ABSTRACT

Title of Dissertation: EFFICIENT TERRAIN ANALYSIS OF POINT CLOUD DATASETS ON A DECOMPOSITION-BASED DATA REPRESENTATION

Yunting Song
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Leila De Floriani,
Geographical Sciences Department
University of Maryland Institute for
Advanced Computer Studies

With a modern focus on LiDAR (Light Detection and Ranging) technologies, which generate precise three-dimensional measurements from the Earth's surface, the amount of spatial data in the form of massive point clouds has dramatically increased. This dissertation addresses the problem of direct terrain analysis using large LiDAR point clouds without interpolating them into gridded Digital Elevation Models (DEMs). Unlike gridded DEMs, Triangulated Irregular Networks (TINs) maintain full information of point clouds and can represent terrains with variable resolution. When using TINs to represent large terrains, the major challenges are the high storage and time costs. To address these, this dissertation introduces a family of decomposition-based data structures, named *Terrain trees family*, for encoding TINs. A Terrain tree employs a nested subdivision strategy, partitioning the domain of the triangle mesh into several leaf blocks.

Each leaf block contains the minimum amount of information required for extracting all connectivity relations that are needed for TIN navigation and terrain analysis. A new library for terrain analysis, the *Terrain trees library* (TTL), is developed based on the Terrain trees. Performance evaluation of TTL shows that a Terrain tree can encode the same terrain with $\sim$36% less storage than the state-of-art, compact data structure while maintaining good computing performance in extracting connectivity relations. Despite the highly efficient data structure, managing large TINs on local machines remains challenging, particularly for complex analyses or simulations. Mesh simplification methods are commonly applied to reduce TIN sizes to enable downstream processing. However, these simplification methods can modify the topology of the underlying terrain in an uncontrolled manner, which affects the results of terrain analysis applications. To address this issue, a topology-aware mesh simplification method based on Terrain trees is proposed. A parallel version of this simplification method is also developed, which simplifies different leaf blocks at the same time using a shared-memory implementation. A leaf-locking strategy is employed to avoid conflicts among leaf blocks during parallel computing.

TTL and the topology-aware mesh simplification methods on Terrain trees effectively lower the memory and time requirements for terrain analysis on TINs. This dissertation demonstrates the effectiveness of TIN-based models in real-world applications using sea ice topography as an example. Studying sea ice topography is crucial as it enhances our ability to monitor sea ice volume changes and comprehend sea ice processes. Besides, timely and precise assessments of sea ice dynamics are critical in the context of climate change and its impacts on polar regions. TIN-based surface models are employed to represent the sea ice surface, and methods are developed for extracting important sea ice topographic features, such as density, regions without measurements, roughness, and pressure ridge structures, from TINs.

EFFICIENT TERRAIN ANALYSIS OF POINT CLOUD DATASETS ON A DECOMPOSITION-BASED DATA REPRESENTATION

by

Yunting Song

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Leila De Floriani, Chair/Advisor
Professor Sinead Farrell
Professor Federico Iuricich
Professor Hanan Samet
Professor Sergii Skakun
Professor Yiqun Xie

© Copyright by
Yunting Song
2024

Foreword

The research presented in Chapter 3 of this dissertation has been published in Springer International Journal on Advances of Computer Science for Geographic Information Systems (GeoInformatica) in 2023 [72]. This is a collaborative work involving extensive algorithmic design and software development. The first three authors are listed in alphabetical order and have contributed equally to the paper, while the fourth author is the senior author and advisor. This work introduces a new family of decomposition-based data structures, named *Terrain trees family*, for Triangulated Irregular Networks (TINs). An open-source library, named *Terrain trees library* (TTL), has been developed based on Terrain trees.

My contributions to this work are substantial and include designing and developing algorithms for morphological terrain analysis and multifield measure calculation, implementing discrete Morse gradient computation in TTL, developing python wrappers of TTL, conducting the entire experiments, and writing sections on the above algorithms and experiments of the paper.

Per the requirement in Doctor of Philosophy Degree Policies, a letter to the Dean of the Graduate School to request the inclusion of a co-authored work is attached in Appendix A.1

Dedication

To my parents, Yiwen Wen and Xinsheng Song, for your love, support, and encouragement throughout every stage of my life. To my grandfather, Tiansheng Song, for your love and understanding, and for being my advisor during my childhood.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Leila De Floriani. I could not have achieved this milestone without her continuous support and encouragement throughout my academic journey. I am grateful for her patience and understanding when I faced challenges in my research. Working with her has not only been instrumental in the completion of my thesis but has also profoundly influenced my life decisions.

I also want to thank the experienced researchers in our group, Dr. Riccardo Fellegara and Dr. Federico Iuricich, for their generous help with my dissertation research. They have provided me with guidance and support that are no less than that of a second advisor. I learned a lot from working with them and from our discussions. I would like to extend my thanks to all other fellow researchers in our group, Xin Xu, Noel Dyer, Yuehui Qian, and Haoan Feng, for their insightful suggestions and kind help throughout this journey.

I extend my profound gratitude to Prof. Sinead Farrell and Kyle Duncan, who not only co-authored the research paper on sea ice topography study with me but also provided invaluable guidance and insightful suggestions throughout the process.

My sincere appreciation also goes to all my committee members, Prof. Sinead Farrell, Prof. Federico Iuricich, Prof. Hanan Samet, Prof. Sergii Skakun, and Prof. Yiqun Xie. Their expertise and insightful feedback have been significant in shaping the direction and focus of my research. Their reviews and constructive suggestions have helped me to improve the quality of my work.

Thanks to all my friends I met in the GEOG department. The time we spent together made this experience far more enjoyable and fulfilling.

To my friends QD, Pomu, and Mint, thank you for your unconditional emotional support during the most challenging times. I could not have come this far without you standing by me.

To my friend Han Zhang, thank you for your patience and understanding during the times I was less available. I will always appreciate the support I received from you.

This research was supported by the US National Science Foundation under grant number IIS-1910766.

Table of Contents

Foreword ii

Dedication iii

Acknowledgements iv

Table of Contents vi

List of Tables ix

List of Figures x

List of Abbreviations xx

Chapter 1: Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Research objectives and dissertation structure . . . 7

Chapter 2: Related work 10
- 2.1 Background . . . 10
  - 2.1.1 Triangle-based terrain representation . . . 10
  - 2.1.2 Surface topology . . . 13
- 2.2 Review of state-of-the-art . . . 18
  - 2.2.1 Data representation for triangle-based terrain models . . . 19
  - 2.2.2 Triangle mesh simplification . . . 21
  - 2.2.3 Topology-based terrain analysis and simplification . . . 24
  - 2.2.4 Sea ice topography measurements . . . 26

Chapter 3: Terrain trees: a decomposition-based data representation for triangulated terrain modeling and analysis 31
- 3.1 Introduction . . . 31
- 3.2 Terrain Trees . . . 33
- 3.3 Implementation of Terrain trees . . . 37
  - 3.3.1 Spatial and connectivity-based queries in a Terrain tree . . . 39
- 3.4 Morphological Terrain Features . . . 41
- 3.5 Topology-based Terrain Segmentation . . . 43
  - 3.5.1 Forman gradient computation . . . 44

3.5.2 Extracting the critical net . . . 46
3.6 Multifield Visualization . . . 50
3.7 Experimental results . . . 52
3.7.1 Selection of capacities for Terrain trees generation . . . 53
3.7.2 Terrain trees evaluation . . . 56
3.7.3 Computing morphological features . . . 59
3.7.4 Computing topology-based segmentations . . . 61
3.7.5 Multifield data visualization . . . 64
3.8 Concluding Remarks . . . 67

Chapter 4: Topology-aware TIN simplification on Terrain trees 69
4.1 Introduction . . . 69
4.2 Topological cancellation . . . 71
4.3 Topology-aware edge contraction . . . 73
4.4 Topology-aware simplification on Terrain trees . . . 76
4.5 Parallel topology-aware edge contraction on Terrain trees . . . 80
4.5.1 Parallel edge contraction algorithm . . . 80
4.5.2 Discussion on the leaf-locking strategy . . . 83
4.6 Terrain tree update after simplification . . . 87
4.7 Terrain mesh evaluation . . . 91
4.8 Experimental results . . . 95
4.8.1 Performance evaluation . . . 96
4.8.2 Compression rate evaluation . . . 102
4.8.3 Mesh quality evaluation after simplification . . . 103
4.8.4 Progressive strategy for improving output mesh quality . . . 109
4.8.5 Selection of leaf capacity on mesh quality . . . 111
4.9 Concluding remarks . . . 113

Chapter 5: TIN-based sea ice surface topography mapping 116
5.1 Introduction . . . 116
5.2 Data description . . . 118
5.2.1 Operation IceBridge Datasets . . . 119
5.2.2 ICESat-2 datasets . . . 121
5.3 Triangulated surface model . . . 122
5.4 Topographic statistics calculation . . . 127
5.4.1 Regional data statistics . . . 127
5.4.2 Surface roughness . . . 131
5.5 Pressure ridges extraction . . . 133
5.5.1 Surface topology and ridge structure . . . 135
5.5.2 Topological simplification . . . 136
5.5.3 Ridge structure extraction . . . 138
5.6 Results . . . 140
5.6.1 OIB ATM point density . . . 142
5.6.2 Sea ice surface roughness . . . 145
5.6.3 Selecting parameters for ridge extraction . . . 149

5.6.4 Results from ridge extraction . . . 155
5.7 Discussion . . . 162

Chapter 6: Conclusions 167
6.1 Major contributions . . . 167
6.2 Additional Research Contributions . . . 171
6.2.1 Critical Features Tracking on Irregular Terrain Meshes by a Scale-Space Method . . . 172
6.2.2 Topology-based Terrain Segmentation Using Apache Spark . . . 174
6.3 Future research . . . 176

Appendix A: Supplementary material 179
A.1 Letter to include co-authored paper . . . 179
A.2 Chapter 3 - Additional results . . . 180
A.3 Chapter 4 - Computing the quadric error matrix on Terrain trees . . . 186
A.4 Chapter 4 - Performing an edge contraction . . . 187
A.5 Chapter 4 - Experiments on leaf capacity selection . . . 188
A.6 Chapter 4 - Additional results . . . 189
A.7 Chapter 5 - Influence of parameters on ridge extraction results . . . 195
A.8 Chapter 5 - Additional results . . . 197

Bibliography 201

List of Tables

3.1 Overview of experimental datasets used in Part I. For each terrain, we list the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$. 52

3.2 Overview of Terrain trees. For each Terrain tree, we list the thresholds k_V and k_T and the number of blocks in the index ($|N|$). 56

3.3 Comparison of storage, expressed in megabytes (MB) and gigabytes (GB), for the underlying TIN, Terrain trees and IA data structure. O.O.M. stands for Out Of Memory. 57

4.1 Overview of experimental datasets. For each terrain, the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$ are listed. 96

4.2 Time **T** (in minutes) and peak memory usage **M** (in Gigabytes) of the simplification on the IA data structure and the Terrain tree (TT) when using different cost threshold ω. **Q1**, **Q2**, and **Q3** represent the first, second, and third quartile edge costs of each dataset, respectively. The best performance value is denoted in bold face and blue color. 98

4.3 Time **T** (in minutes) and peak memory usage **M** (in Gigabytes) of topology-aware mesh simplification on the IA data structure, and on sequential (seq.) and parallel (para.) versions on Terrain trees. The best performance value is denoted in bold face and blue color. 98

5.1 The sum of square error (unit: m^2) and root mean square error (values in the brackets, unit: m) of roughness distributions at different radii and tested models. The tested models include the exponentially modified normal distribution (denoted as *exponnorm*), exponentiated Weibull distribution (denoted as *exponweib*), exponential power distribution (denoted as *exponpow*), Fisk distribution, *log-normal* distribution (denoted as *lognorm*), and normal distribution (denoted asd *norm*). 148

A.1 Time (in minutes) (denoted as **T**), peak memory usage (in Gigabytes) (denoted as **M**), and compression rate (in %) (denoted as **R**) of simplification when using different capacity values (denoted as **C**) in the Terrain tree generation. The capacity value in bold face is the capacity value used to generate the Terrain tree of each dataset. 188

List of Figures

2.1 Example of star and link condition. The *star* of v_a consists of the triangles incident in it (red triangles and blue triangles) and edges incident in it ($\{v_a, v_b\}$, $\{v_a, v_c\}$, $\{v_a, v_d\}$, $\{v_a, v_f\}$, and $\{v_a, v_g\}$). The *link* of v_a is formed by vertices v_b, v_g, v_f, v_c, v_d and the edges connecting them. The link of edge $\{v_a, v_b\}$ consists of v_d and v_g. 11

2.2 (a) Forman gradient. Regions of influence of critical simplices are computed visiting the V-paths (b) at the critical simplices, and (c) until the whole region of influence is visited. (d) V-paths connecting pairs of critical simplices. 16

2.3 Example of Forman gradient calculated from a surface with an elevation function. Spheres on the figure correspond to the centroids of critical vertices, edges, and triangles in the Forman gradient. Red spheres are maxima (critical triangles), blue spheres are saddles (critical edges), and yellow spheres are minima (critical vertices). The green triangles are the triangles along the separatrix V-paths between maxima and saddles. The red lines connecting the centroids of such triangles correspond to the geometry of maxima-saddle arcs in the surface network G. 18

2.4 Example of (a) a persistence diagram formed by four points and (b) its birth-persistence chart. Points with the same color in (a) and (b) correspond to the same point. 19

3.1 Given the triangle mesh of Fig. 3.2, a vertex capacity $k_V = 2$ and a triangle capacity $k_T = 2$, the Figure shows the spatial subdivision obtained with a PR-T Tree (a), a PM-T Tree (b) and a PMR-T Tree (c). In black are highlighted the spatial decomposition caused by the vertices capacity, while in red those caused by the triangles one. For the PMR-T tree we also highlight the triangle insertion order that drives the spatial decomposition (note that k_T is not a bucketing capacity for the PMR-T tree). 34

3.2 (a) Triangle mesh and PR-Tree as encoded in a Terrain tree. (b) The mesh topology is organized by encoding, for each triangle, the boundary relation with its vertices. (c) The PR-T represents a hierarchy where each leaf block encodes the list of vertices contained in the block and the list of triangles intersecting the block. 38

3.3 Homotopy expansion [155] computed on the simplices belonging to the lower star of vertex 6. 45

3.4 Reconstruction of a portion of the critical net connecting a critical saddle with two critical minima. (a) Starting from a critical edge, the boundary vertices are connected with the edge. (b) The edge e_1 paired with v_1 is added to the stack. (c) The edge e_1 is extracted from the stack, connected with its paired vertex v_1. The other vertex on the boundary of e_1 (i.e., v_2) is retrieved and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until both paths reach critical vertices (minima). 47
3.5 Reconstruction of a portion of the critical net connecting a critical saddle with a critical maxima. (a) Starting from a critical edge, the coboundary triangle (i.e., t_1) is visited. (b) The traversal starts by adding the edge e_1, paired with t_1, to the stack. (c) The edge e_1 is extracted from the stack and connected with its paired triangle t_1. The other triangle in the coboundary of e_1 (i.e., t_2) is extracted and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until a critical triangle (maximum) is reached. 49
3.6 The storage costs for storing the hierarchical index of the Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 54
3.7 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 55
3.8 Comparison of total timings, expressed in minutes (m), for generating a Terrain tree or the IA data structure. 58
3.9 Comparison of total timings, expressed in minutes (m), for extracting the VT relations. 59
3.10 Comparison of total timings, expressed in minutes (m), for extracting edge slopes. 59
3.11 Comparison of total timings, expressed in minutes (m), for extracting concentrated curvatures. 60
3.12 Comparison of total timings, expressed in minutes (m), for extracting roughness values. 61
3.13 Comparison of total timings, expressed in minutes (m), for computing the Forman gradient vector. 62
3.14 Comparison of total timings, expressed in minutes (m), for computing the critical net. 63
3.15 The satellite map and the visualization of the multifield comparison measure on an area with few trees and some human artifacts. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), based on elevation paired with green values (c), and based on elevation paired with RGB values (d). 64
3.16 The satellite map and the visualization of the multifield comparison measure on an area with high tree density. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), and based on roughness paired with RGB values (c). 65
3.17 Comparison of total timings, expressed in minutes (m), for computing the multifield measure. 66

4.1 Example of a maximum-saddle-maximum cancellation operation on a surface network G. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines highlight arcs starting from maximum M_2. Note that the regular points are not a part of the surface network and are just displayed for illustration purpose. (a) Original surface network; (b) surface network when a boundary maximum M_4 and a saddle S_2 adjacent to it are removed; (c) surface network when an internal maximum M_1 and a saddle S_1 adjacent to it are removed. 73
4.2 Example of edge contraction not satisfying the link condition (same figure as 2.1 in Section 2.1.1). Given edge $e = \{v_a, v_b\}$, we have $Lk(v_a) \bigcap Lk(v_b) = \{v_d, v_g, v_f, \{v_f, v_g\}\}$ and $Lk(e) = \{v_g, v_d\}$. The condition is not satisfied since $Lk(v_a) \cap Lk(v_b)$ contains vertex v_f and edge $\{v_f, v_g\}$ that are not in $Lk(e)$. . . . 74
4.3 Two possible gradient configurations and corresponding updates after contracting edge $\{v_1, v_2\}$ (in red) to v_1. Black arrows represent gradient pairs and red arrows represent contraction direction. 75
4.4 (a) Example of 1-neighborhood (vertices in green) and 2-neighborhood (vertices in green and vertices in blue) of a vertex v. (b) Example of a vertex v and two vertices in its 2-neighborhood. (c) Example of edge contraction on edge $e = \{v_1, v_2\}$ (in red), where v_2, t_1, and t_2 are removed by the contraction. Gradient pairing information of cyan triangles may be modified due to the contraction of e. Red vertices are reconnected to v_1 after the contraction during which v_2 is removed. 83
4.5 (a) Example of a possible conflict occurring when two edges are contracted at the same time (triangles are not displayed for clarity). Edge $e_1 = \{v_1, v_2\}$ is contracted to v_2 and edge $e_2 = \{v_3, v_4\}$ is contracted to v_3. (b) A case that blocks encoding v_1 and v_4 can be active at the same time under the leaf-locking strategy (c) An invalid case in which v_1 and v_4 cannot be removed at the same time. . . . 86
4.6 (a) Speedup and (b) efficiency achieved by the parallel simplification algorithm when different numbers of threads are used. 99
4.7 The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original *reinsert* method. **Q1**, **Q2**, and **Q3** represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. **All** represents the result when contracting all contractible edges without a cost threshold. The text labels on the boxes in figure indicate the time when **Q1** cost threshold is used and when **All** edges are contracted. 101
4.8 The compression rates of three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (**Q1**, **Q2**, and **Q3**) and when all contractible edges are contracted (**All**). 102
4.9 (a) The maximum vertical distance with respect to the elevation range of the original mesh and (b) the RMSE of vertical distances between the original mesh and the simplified mesh obtained from Para-TT and IA-based simplification. . . 105
4.10 The sliced Wasserstein distance between the original mesh and the simplified mesh when the mesh is simplified with the Geometric-TT method. 106

4.11 The change in the number of critical simplices in two tested dataset when they are simplified with the Geometric-TT method. 108
4.12 The performance and quality evaluation of the parallel simplification on Terrain trees when a progressive optimization is applied to the *Great Smokey Mountains* dataset. 110
4.13 The performance and quality evaluation of the simplification of the *Great Smokey* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds to leaf capacity value used for the Terrain tree generation. 112

5.1 Example of the datasets used in this study. The basemap is an image collected by Operation IceBridge (OIB) Continuous Airborne Mapping By Optical Translator (CAMBOT), the mesh layer colored from blue to red is the TIN surface model generated from OIB Airborne Topographic Mapper (ATM) dataset, and the purple path overlaying the mesh layer is the track of ICESat-2. All datasets were collected during 2019 summer. The inset map displays the flight path of 2019 Summer OIB survey on September 9, 2019. The basemap of the inset map is the topographic basemap offered by Arctic SDI [2]. 118
5.2 OIB ATM point cloud map of a region with open water area. ATM points are displayed as green points in the figure. The basemap layer is the OIB CAMBOT image coincident with the displayed section of ATM point cloud. 119
5.3 The elevation distribution of OIB ATM points (bin width = 0.1 m) (a) within the complete study region and (b) within the segment for ridge extraction experiments. All elevations are relative to he Earth Gravitational Model 2008 (EGM2008) geoid [147]. 120
5.4 (a) CAMBOT image of a region with open water areas. (b) Corresponding TIN generated from the OIB ATM data in the same region and (c) the modified TIN after the underlying mesh is processed with α-shape method (when $\alpha = 20$ m). The triangles in both (b) and (c) are from the underlying triangle meshes of generated TINs. 122
5.5 TIN after the underlying mesh is processed through α-shape method with two different α-values: (a) α-shape when $\alpha = 20$ m; and (b) α-shape when $\alpha = 15$ m. The triangle edges are hidden in both figures for a clearer visualization. The red dashed box highlights the region where triangles overlapping sea ice surface are removed from the TIN unintentionally in (b). 123

5.6 Comparison between OIB ATM-TIN model and ICESat-2 University of Maryland-Ridge Detection Algorithm (UMD-RDA) profiles. (a) The profiles from interpolation on OIB ATM-TIN model along ICESat-2 track (blue line) and the ICESat-2 ATL03 UMD-RDA (orange line) along a 1km segment. (b) Example of a small surface feature that cannot be detected from ICESat-2 UMD-RDA result, but can be detected in OIB ATM-TIN profile. (c) Example of an open water region, which leads to data dropouts in OIB ATM. (d) Distributions of elevations from two profiles in the selected segment (bin width = 0.02 m). Points whose elevation measurement is out of the range ($mean - 2 \times sd.$, $mean + 2 \times sd.$) in either distribution are excluded as outliers. 125
5.7 Possible cases of extracted invalid regions by comparing Σ_{20} and Σ_{25000}. The difference between them are displayed as green regions. The black lines in the map are the boundary of Σ_{25000}. The red lines in the map are the boundary of Σ_{20}. As discussed in Section 5.4.1, (a) shows a case in which the regions of difference indicate regions without measurements, while (b - c) show cases where the regions of difference are caused by the artifacts introduced during the Delaunay triangulation. Σ_i: the mesh after α-shape processing with $\alpha = i$ (unit: meter). . . 128
5.8 The flowchart of steps to extract ridge structures from preprocessed OIB ATM points by using TIN-based surface models. 133
5.9 Maps of maxima and saddles (a) in the original Forman gradient field calculated from the TIN based on discrete Morse theory [76]; (b) after the topological simplification (ω_m = 0.25 m); and (c) after the ridge grouping and roughness filtering (θ_m = 0.09 m) steps. The red lines in (c) are final ridge structures extracted with the proposed method. 136
5.10 (Same figure as Fig. 4.1 in Section 4.2.) Example of a maximum-saddle-maximum cancellation operation on a surface network G. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines highlight arcs starting from maximum M_2. Note that the regular points are not a part of the surface network and are just displayed for illustration purpose. (a) Original surface network; (b) surface network when a boundary maximum M_4 and a saddle S_2 adjacent to it are removed; (c) surface network when an internal maximum M_1 and a saddle S_1 adjacent to it are removed. 137

5.11 Example of ridge grouping. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines are considered the ridge structure of maximum M2. Note that other regular points along the gradient paths are not shown in this figure. The elevation anomaly H_A of a node is the anomaly of the highest vertex of the corresponding geometry. With the given anomalies in the graph, the ridge structures of M_3 should be merged to that of M_2 according to the discussion in Section 5.5.3. 139
5.12 (a) Point density distribution within the study area (bin width = 0.05 pt/m^2). (b) Point density distribution (bin width = 0.1 pt/m^2) and (c) density map of a 5 km segment in the study area. The point density is calculated within a 10 m search radius for each point in the dataset. 141
5.13 (a) An example of the extracted regions without measurements, the black boundary indicates the range of the study OIB ATM data and the red polygons indicate the regions without measurements, which is extracted by comparing Σ_{20} and Σ_{25000}. (b) The distribution of extracted water/thin ice surface regions within the study region. Σ_i: the mesh after α-shape processing with $\alpha = i$ (unit: meter). . . 141
5.14 (a - d) The 10 m roughness maps of four selected regions. (e - h) The corresponding images of the four regions. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image. 142
5.15 Region A: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image. 144
5.16 The distribution histograms of roughness values at vertices in the study region with different measure range radii (bin width = 0.01 m). The curve line represents the best-fitting distribution model, starting at the 1% percentile point function (PPF) and concluding at the 99% PPF of the distribution. The root mean square error (rmse) and parameters for this best-fitting model are listed. The name of the best-fitting model is labeled in the top-right corner of each chart. 145
5.17 Relationship between roughness and persistence thresholds and (a) the percentage and the total number of R_A ridges that intersect with the 5 m buffer zone around the ICESat-2 track; (b) the total number of R_A ridges within the range of six visually inspected images; (c) the total length of R_A ridge parts within the range of six visually inspected images; (d) the percentage of R_I ridges that have a match in R_A; (e) the percentage of R_M ridges that have a match in R_A; (f) the total length of R_M ridge parts matched with R_A. R_I: the ridge peaks extracted from ICESat-2 profile data through the UMD-RDA [54]. R_M: ridge lines visually extracted from three CAMBOT images in the study area. R_A: ridge structures extracted from OIB ATM data using our method. 151

5.18 (a) The profiles from interpolation on OIB ATM-TIN model along ICESat-2 track (blue line) and the ICESat-2 ATL03 University of Maryland-Ridge Detection Algorithm (UMD-RDA) (orange line) along a 1 km segment. Orange circles indicate the ridge peaks identified by UMD-RDA and the blue circles indicate the intersections between the extracted ridges from ATM-TIN model and the ICESat-2 track; (b) elevation map of ATM TIN model with extracted ridges of the 1 km segment shown in (a), the region bounded by yellow box corresponds to the highlighted segment in (a); (c) Zoom-in images and TINs of the selected segment in (a - b). All ridges are extracted by our method when the persistence threshold ω_m = 0.25 m and roughness threshold θ_m = 0.09 m. 156
5.19 Example where the ridge results from the ICESat-2 ATL03 UMD-RDA does not match with the results from our method operated on the OIB ATM-TIN model. The basemap in (a) is the α-shape generated from OIB ATM when $\alpha = 20$ as described in Section 5.3 and the image in (b) is the OIB CAMBOT image coincident with the OIB ATM data. 157
5.20 Example of ridge extraction results when ω_m = 0.25 m, θ_m = 0.09 m. The basemap in (a) is the α-shape generated from OIB ATM when $\alpha = 20$ as described in Section 5.3 and the image in (b) is the OIB CAMBOT image coincident with the OIB ATM data. The blue lines in (b) are ridge lines selected through visual inspection for reference. 158
5.21 Examples where the ridge results from the visual inspection do not match with the results from our method operated on the OIB ATM-TIN model. The upper figures, colored with blue-red legend, are the α-shapes generated from OIB ATM when $\alpha = 20$ and the lower figures are OIB CAMBOT images coincide with the OIB ATM data. The yellow dashed box in (a) highlights a region with complex terrain features and our method is able to identify such features. The black dashed box in (b) highlights a segment of visually selected ridge line l_2 that does not match with ridges extracted by our method. 159
5.22 (a) Joint distribution of ridge lengths and ridge peak elevation anomalies H_A; (b) Ridge length distribution (bin width = 10 m); (c) Ridge peak H_A distribution (bin width = 0.1 m); (d) Diagonal length distribution of ridge Oriented Bounding Box (OBB) (bin width = 10 m); and (e) ridge OBB orientation distribution (bin width = 15°). The ridges results in all five figures are extracted with persistence threshold ω_m = 0.25 m and roughness threshold θ_m = 0.09 m. 161
5.23 Roughness maps calculated with TIN-based method and grid-based method with similar measure scales. For instance, when the search radius γ on TINs equals to 5, the corresponding grid cell size l is twice the radius, which is 10. The region in the map is Region A introduced in Section 5.6.2. 164

6.1 Critical point classes according to the piece-wise linear (PL) Morse theory [13]. The labels indicate the type of the central vertex *v*. Among vertices in the Vertex-Vertex relation of *v*, red vertices have lower elevations than *v* and blue vertices have higher elevations than *v*. 173

A.1 The storage costs for storing the hierarchical index of the Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 180
A.2 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 181
A.3 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 181
A.4 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 182
A.5 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 182
A.6 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 183
A.7 The storage costs for storing the hierarchical index of the Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 183
A.8 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 184
A.9 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 184
A.10 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 185
A.11 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices. 185
A.12 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value. 186
A.13 The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original **Q1**, **Q2**, and **Q3** represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. **All** represents the result when contracting all contractible edges without a cost threshold. The text labels on the boxes in figure indicate the time when **Q1** cost threshold is used and when **All** edges are contracted. 190

A.14 The average triangle shape measure of meshes obtained from three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (**Q1**, **Q2**, and **Q3**) and when all contractible edges are contracted (**All**). 191
A.15 The change in the number of critical simplices in each dataset when it is simplified with the Geometric-TT method. 192
A.16 The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Molokai* dataset. 193
A.17 The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Canyon Lake* dataset. 193
A.18 The performance and quality evaluation of the simplification of the *Molokai* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds leaf capacity value used during the Terrain tree generation. 194
A.19 The performance and quality evaluation of the simplification of the *Canyon Lake* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds leaf capacity value used during the Terrain tree generation. 195
A.20 Comparison between the ridge extraction results when the persistence threshold $\omega_m = 0$ m, the persistence threshold $\theta_m = 0$ m and when $\omega_m = 0.25$ m, $\theta_m = 0$ m. The basemap is (a) the surface model generated from OIB ATM as described in Section 5.3 and (b) the CAMBOT image. The blue lines are manually selected ridge lines for reference. The value after **P** in the legend labels indicates the persistence threshold and the value after **R** indicates the roughness threshold. The black dashed box in (a) highlights the ridge lines removed during the topological simplification. The purple dashed box in (a) highlights the ridge line extended during the topological simplification. 196
A.21 Comparison between the ridge extraction results when $\omega_m = 0.25$ m, $\theta_m = 0$ m and when $\omega_m = 0.25$ m, $\theta_m = 0.09$ m. The basemap is (a) the surface model generated from OIB ATM as described in Section 5.3 and (b) the CAMBOT image. The blue lines are manually selected ridge lines for reference. The value after *P* in the legend labels indicates the persistence threshold and the value after *R* indicates the roughness threshold. 196
A.22 Region B: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image. 198
A.23 Region C: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image. 199

A.24 Region D: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image. 200

List of Abbreviations

ACM	Association for Computing Machinery
AGS	Average Gradient on Star
ATLAS	Advanced Topographic LiDAR Altimeter System
ATM	Airborne Topographic Mapper
CAMBOT	Continuous Airborne Mapping By Optical Translator
CR	Coverage range
DMS	Digital Mapping System
DT	Delaunay Triangulation
EE	Edge-Edge
EGM2008	Earth Gravitational Model 2008
EPSG	European Petroleum Survey Group
ET	Edge-Triangle
EV	Edge-Vertex
exponpow	exponential power distribution
exponnorm	exponentially modified normal distribution
exponweib	exponentiated Weibull distribution
Geometric-TT	Parallel geometric simplification on Terrain trees
GIS	Geographic Information Systems
GPS	Global Positioning System
IA	Indexed data structure with Adjacencies
ICESat	Ice, Cloud, and land Elevation Satellite
ISPC	Implicit SPMD Program Compiler
LiDAR	Light Detection and Ranging
LKF	Linear Kinematic Feature
lognorm	log-normal distribution
LRU	Least-Recent-Used
LS	Lower star
MPI	Message Passing Interface

NASA	National Aeronautics and Space Administration
NOAA	National Oceanic and Atmospheric Administration
norm	normal distribution
NSIDC	National Snow and Ice Data Center
OBB	Oriented Bounding Box
OIB	Operation IceBridge
O.O.M.	Out of Memory
OpenMP	Open Multi-Processing
Para-TT	Parallel topology-aware simplification on Terrain trees
PL	Piece-wise linear
PM	Polygonal Map
PM-T	PM-Terrain
PMR	Randomized Polygonal Map
PMR-T	PMR-Terrain
PM2T	PM_2-Triangle
PPF	Percentile Point Function
PR	Point-Region
PR-T	PR-Terrain
Pro-Para-TT	Parallel simplification on Terrain trees with a progressive strategy
Pro-TT	Sequential simplification on Terrain trees with a progressive strategy
QEM	Quadric Error Metric
Q1	First quartile
Q2	Second quartile
Q3	Third quartile
RGB	Red, Green, Blue
RMSE	Root Mean Squared Error
SAR	Synthetic Aperture Radar
Seq-TT	Sequential topology-aware simplification on Terrain trees
SRE	Sequential Range Encoding
TBB	Threading Building Blocks
2D	Two dimensional
TE	Triangle-Edge
TIN	Triangulated Irregular Network
TT	Triangle-Triangle
TTL	Terrain trees library
TV	Triangle-Vertex

UMD	University of Maryland
UMD-RDA	University of Maryland-Ridge Detection Algorithm
VE	Vertex-Edge
VT	Vertex-Triangle
VV	Vertex-Vertex

Chapter 1: Introduction

1.1 Introduction

With the development of new data collection methods, more and more spatial data represented as raw massive point clouds, such as Light Detection and Ranging (LiDAR) data, become available. Such techniques provide highly accurate elevation measurements from the Earth's surface, which are crucial for detailed terrain analysis. This increased data availability gives rise to new research challenges and technological developments. LiDAR data is used in a variety of different fields, including urban modeling [188], climate study [12], earthquake analysis [127], disaster management [115], flood risk mapping [190], forest analysis [38, 195], and coastal morphology computation [162, 192]. Terrain analysis based on accurate elevation measurements enables the extraction of features relevant to several applications. For example, the slope of terrain is used to map seafloor habitats [112], while the combination of slope, roughness, and curvature allows for the modeling of coral distribution [52]. Also, the segmentation of a terrain according to its critical features (i.e., peaks and pits) provides information about terrain morphology, which is fundamental for assessing the risk of landslides or floods. Research in these fields has been greatly enhanced by the increasing availability of open data repositories.

However, these new opportunities also pose challenges, such as direct terrain modeling and analysis from large, unevenly distributed LiDAR point clouds. While it is possible to extract some

local terrain features, such as roughness and curvature, directly from point datasets, a variety of analytical techniques are not feasible on point clouds. For instance, all topological analyses of terrains, including terrain segmentation and critical feature extraction, require the computation of topology on a surface model. Moreover, visibility studies, such as line-of-sight and visual exposure analysis, as well as the extraction of feature ranges like lakes, ridges, and valleys, also necessitate terrain models generated from LiDAR data. A Triangulated Irregular Network (TIN) is a desirable terrain model for LiDAR point clouds as it can be generated directly from irregularly-distributed points. Terrain models built on the original point datasets can maintain the details of the terrain surface and use the full information content from LiDAR datasets. However, existing Geographic Information Systems (GISs) tools have limited support for TINs loading and processing. *GRASS GIS* [3] only provides a tool to construct a Delaunay triangulation of a set of points, but it does not support processing and analysis operations on TINs. *QGIS* [6] and *ArcGIS Pro* [1] can only process TINs of limited size and they do not support many terrain analysis methods. Even compact data structures for encoding TINs suffer from scalability issues. For instance, using the most compact state-of-art data structure for triangle meshes, we can process meshes up to 150 million vertices (300 million triangles), but still much lower than the size of currently available data sets (see, for instance, the *OpenTopography* repository [143]).

A point dataset larger than the software limits needs to be interpolated into gridded Digital Elevation Models (DEMs) before being used for representing terrains. While it is always possible to transform a point cloud into a raster representation, as a gridded DEM, this process is a computationally intensive, and can account for 70-80% of the time required by the entire terrain analysis pipeline [7, 16]. Also, the resolution of a gridded DEM is constrained by cell size and not by the quality of original point data. After a gridded DEM is generated, the original data will

be discarded and there is no possibility to improve the resolution. Moreover, when represented in a raster format, the boundary information of terrain surface may be lost or roughly approximated. Therefore, TINs are the preferred modeling choice in applications where boundary information is important, such as in coastal and ocean modeling [10, 22]. Considering the increasing need for high-resolution terrain models and the limitations of existing tools, new techniques for the efficient representation of triangulated terrain models are necessary.

This dissertation will address the problem of direct terrain modeling and analysis of LiDAR point clouds by using triangulated surface models, i.e. TINs. The first proposal for a new data structure, called *Terrain tree*, which encodes the topology of a triangulated terrain combined with a spatial index built on the triangle mesh, has been introduced in a 4-page paper presented at ACM SIGSPATIAL in 2017 [69]. In this dissertation, I will introduce a set of three indexes forming the *Terrain trees family* which is developed based on the concept of Terrain tree [69]. A Terrain tree is a spatial-decomposition-based data structure for TINs. In another word, it contains a list of blocks obtained from spatial decomposition and each block encodes a subset of the vertices and triangles of a TIN Θ. It encodes the minimal connectivity information required for navigation on Θ with a hierarchical spatial index. Other connectivity relations are extracted on the fly within each block of the decomposition and discarded after the processing. This design helps to reduce the memory footprints of operations on Terrain trees and allows for the development of efficient TIN-based terrain analysis methods. Based on the Terrain trees data structures, a new library for terrain analysis called the *Terrain trees library* (TTL) has been developed.

TTL contains a kernel for connectivity and spatial queries, and modules for extracting morphological features, including edge and triangle slopes, roughness, and curvature. Such features are measures to quantitatively describe the terrain and can help people to locate significant fea-

tures. TTL also contains a technique to integrate multiple scalar fields defined on the same terrain by computing a multi-field measure based on the strategy defined by Nagaraj et al. [135]. A terrain with multiple fields defined on it is referred to as a *multi-field dataset*, and can be found in several domains, like forest management, weather predictions and geological sciences [100].

Another important module of TTL is the one for topological structures extraction based on discrete Morse theory [76]. Discrete Morse theory [76] is a combinatorial counterpart of smooth Morse theory [133], which is a powerful mathematical framework for computing the terrain topology. It enables the segmentation of a scalar field according to the regions of influence of its critical points. This general task has proved fundamental in many application domains, including material science [90], chemistry [141], environmental science [186], forest monitoring [195], and urban analysis [51], to mention a few. In terrain analysis, particularly, the segmentation of a terrain according to its critical points (i.e., peaks and pits) provides information regarding terrain morphology, which are fundamental for assessing the risk of landslides or floods.

Spurious critical points, naturally occurring in data due to noisy acquisitions, can severely affect terrain analysis. For this reason, several simplification approaches capable of removing spurious features, while maintaining important critical points, have been defined in the literature [18, 33, 122]. These approaches reduce the topological complexity of the dataset, while leaving the underlying terrain model unchanged. However, when working with large terrain datasets, a notable issue arises: the complexity of extracting, representing, and visualizing topological features and structures is directly related to the resolution of the terrain model. For instance, triangulated terrain models are widely used in storm surge and tide simulation models due to their advantages at maintaining line features, like coast lines or barriers. While using the finest mesh can ensure the simulation accuracy, it contains a huge amount of triangles and makes large-extent

simulations computationally expensive. In practice, fast simulation of a large domain is needed to provide timely prediction while the available computing resources are likely to be limited [15].

Therefore, a mesh simplification method is often used to reduce the size of a TIN in order to enable or accelerate downstream processing in simulations. However, mesh simplification can affect the topology of underlying terrain in an uncontrolled way, thereby deteriorating the simulation and segmentation results. Topology-aware operators [50, 99] have been defined to solve this issue by coarsening a TIN without affecting the topology of its underlying terrain, i.e., without modifying critical features describing pits, saddles, peaks, and their connectivity. By combining such an operator with a topological simplification operator, users are able to simplify the resolution of both the topology and the geometry of a terrain in a completely controlled way. However, when processing large terrains, multiple issues arise. First, encoding the original TIN requires significant memory resources, and, thus, there is the need to use efficient representations, like Terrain trees, to reduce memory usage. Secondly, performing a large number of simplifications sequentially is time-consuming [99], directly affecting user interactions during data exploration. Thus, there is a need to develop a parallel simplification strategy. This dissertation addresses both issues by designing and implementing a new topology-aware simplification approach for triangulated terrains on Terrain trees.

The proposed methods for efficient TIN-based terrain analysis and processing are beneficial for all fields that require accurate descriptions of the Earth's surface. For instance, accurate surface model of sea ice in polar regions is crucial for monitoring the sea ice changes and for understanding their impacts on various sea ice processes, such as ice melting [36] and momentum transfer to the ocean [126]. Moreover, the rapid changes in sea ice cover and thickness [34] make timely measurements and analysis of sea ice topographic features more important.

Every year a large amount of point cloud data is collected by satellite altimeters, like Ice, Cloud, and land Elevation Satellite (ICESat)-1, -2 [125], and airborne altimeters, like the Airborne Topographic Mapper (ATM) [104] used in the NASA Operation IceBridge (OIB) campaign. Those data provide an accurate measurement of the ice elevation, but existing studies on sea ice surface topography rely on the profiles [67] or raster grids [151] generated from point clouds. They either only reflect the one-dimensional change of the terrain surface, or cannot fully utilize the information from high-resolution point clouds. Considering the fact that the LiDAR points are distributed irregularly, it is natural to use TINs instead of raster grids as sea ice surface models. As mentioned above, the major limitation hindering the use of TINs is their high storage costs and the challenges associated with conducting efficient analysis on large TINs. This problem can be addressed by using the proposed data representation, Terrain trees, as it is efficient in both time and space.

Based on the sea ice surface models, different measures or features can be calculated for more insights. For instance, the knowledge of sea ice roughness is crucial for understanding snow distribution [97], parameterizing momentum transfer to the ocean [126], and processing of remote sensing data collected from sea ice surfaces [109, 110]. Generally, higher sea ice roughness indicates more complex topographic features, such as pressure ridges, melt ponds, and leads [139]. Pressure ridges are formed from the collisions between ice floes. The mapping of ridges are important factors to be considered in traffic-ability analysis [39] and in evaluating the degree of deformation [154]. While previous studies have proposed methods for calculating topographic features from sea ice profiles [67] and raster grids [151], such methods cannot be directly used on TINs. Therefore, this dissertation will address this gap by introducing methods to extract sea ice topographic features, like roughness and ridges, from TIN-based surface models.

To reach this goal, existing functions in TTL, which are not tailored for sea ice studies, are modified and extended to compute interested topographic features from sea ice surface models.

1.2 Research objectives and dissertation structure

This dissertation aims to address the issue of direct terrain modeling and analysis on LiDAR point clouds without interpolating them into grids. As discussed in Section 1.1, TIN-based terrain models can avoid interpolation on point clouds, but their application is limited by their large storage costs and high time costs during processing. A straightforward solution is to use compact data representations for TINs to reduce such costs. But even with a compact data representation, the size of a TIN may still need to be reduced to enable or expedite downstream processing. This reduction could be essential for performing complex analyses or simulations more quickly and at larger scales. Thus, this dissertation also introduces a new method for TIN simplification on the proposed compact data representation. The proposed solution for efficient TIN-based terrain analysis is evaluated in this dissertation by being applied in a real-world scenario, the study of sea ice topography. By employing these techniques in a practical setting, we aim to explore the possibility of extracting information that could not be effectively extracted from raw LiDAR point sets or gridded DEMs.

Specifically, this dissertation focuses on the following three research questions, which help to address the major objective.

Question 1. How to encode and directly process point cloud data for representing terrain surface efficiently? As mentioned above, TIN-based terrain models are used to connect point clouds and to represent terrain surfaces. This solution leads to a few more questions to be

addressed in the dissertation:

1. How to reduce the storage cost of a TIN?

2. How to avoid the high memory and time requirements for terrain analysis on TINs?

3. Can the proposed methods be scaled to large datasets?

Question 2. How can TIN-based terrain models built from point clouds be simplified efficiently while preserving their topological features? To develop an efficient topology-aware TIN simplification, there are a few questions that the dissertation needs to address:

1. How can topology-based simplification be performed efficiently in terms of memory requirements and time performance on a TIN?

2. Can the proposed method be extended to a parallel computing environment to further reduce time costs? If so, how will the parallel algorithm affect the simplification outputs or the memory overhead?

3. How can we handle potential noise in the original TINs?

Question 3. How to employ TIN-based representations to improve topography feature extraction from sea ice surfaces? In this dissertation, the techniques developed to address the first two questions are employed to study sea ice surface in polar regions. Specifically, it addresses the following questions:

1. How can accurate TIN-based sea ice surface models be generated? Are the generated models consistent with other independent datasets collected in the same region?

2. Which sea ice topographic features can be extracted from TIN-based surface models, and how can this be achieved?

3. How can roughness be accurately quantified on sea ice surfaces, and how does the measure scale affect the interpretation of roughness results?

4. How can pressure ridge sails be extracted from TIN-based surface models?

This dissertation includes six chapters. Chapter 1 introduces the topic and motivations of the research and outlines the primary objectives. Chapter 2 reviews the background and state-of-art studies related to this dissertation. Chapter 3 addresses the first research question by introducing terrain analysis and processing methods designed based on a new scalable data structure named *Terrain trees*. Chapter 4 addresses the second question by introducing an efficient topology-aware mesh simplification method and its parallelized version developed on Terrain trees. Chapter 5 discusses the third question by introducing the strategy to generate TIN-based sea ice surface models from point clouds, and methods to extract sea ice topographic features. Chapter 6 concludes with a summary of the major contributions of this dissertation, a brief overview of my other research contributions during the PhD study, and a discussion of potential directions for future research.

Chapter 2: Related work

2.1 Background

In this section, we review some background notions on triangle-based terrain representation, including the notions related to triangle meshes and connectivity relations on triangle meshes. We also review discrete Morse theory, which is the basis of the topological terrain analysis methods presented in the dissertation.

2.1.1 Triangle-based terrain representation

Terrain surfaces are usually described by either TINs or raster-based DEMs. A TIN is a digital terrain model defined by a finite set of irregularly distributed points in the plane, each of which has an elevation value associated with. A TIN consists of a triangle mesh connecting the points in the plane and of a piecewise linear interpolating function defined on the triangles of such mesh.

To define a triangle mesh, we need to introduce the concept of a simplex. A *k-simplex* σ is the convex hull of $k+1$ independent points in the Euclidean space $\mathbb{E}^n$ (with $k \geq 0$). A 0-simplex is a vertex, a 1-simplex is an edge and a 2-simplex is a triangle. Any simplex τ, which is the convex hull of a non-empty subset of the points generating a simplex σ, is a *face* of σ.

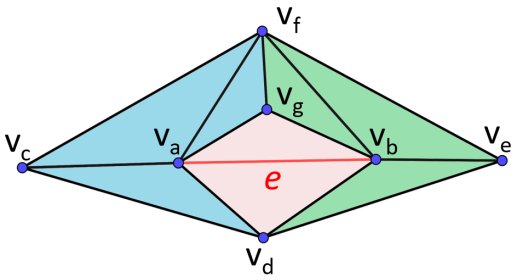


Figure 2.1: Example of star and link condition. The *star* of v_a consists of the triangles incident in it (red triangles and blue triangles) and edges incident in it ($\{v_a, v_b\}$, $\{v_a, v_c\}$, $\{v_a, v_d\}$, $\{v_a, v_f\}$, and $\{v_a, v_g\}$). The *link* of v_a is formed by vertices v_b, v_g, v_f, v_c, v_d and the edges connecting them. The link of edge $\{v_a, v_b\}$ consists of v_d and v_g.

Conversely, σ is a *coface* of τ.

Given a simplex σ, the *star* of σ, denoted as $St(\sigma)$, is the set of cofaces of σ, i.e., the set of simplices with σ as a face. In Fig. 2.1, the *star* of v_a consists of the triangles incident in it (red triangles and blue triangles) and edges incident in it ($\{v_a, v_b\}$, $\{v_a, v_c\}$, $\{v_a, v_d\}$, $\{v_a, v_f\}$, and $\{v_a, v_g\}$). The *star* of edge e consists of two red triangles incident in it. The *link* of σ, denoted as $Lk(\sigma)$, is the set of faces of the simplices in $St(\sigma)$ that do not intersect σ. More specifically, on a triangle mesh, the link of a vertex v consists of all vertices adjacent to v and edges opposite to v in the triangles incident in v. For instance, in Fig. 2.1, $Lk(v_a)$ is formed by vertices v_b, v_g, v_f, v_c, v_d and the edges connecting them. The link of edge e consists of the other two vertices in the triangles adjacent to e. In Fig. 2.1, the link of edge $\{v_a, v_b\}$ consists of v_d and v_g.

A *triangle mesh* Σ is a collection of vertices (0-simplices), edges (1-simplices), and triangles (2-simplices) such that, given any two triangles in Σ, either they have an empty intersection or they intersect at a common simplex (edge or vertex).

The Delaunay triangulation (DT) [42] is the most widely-used method to generate a TIN since it ensures the exclusion of elongated triangles in the resulting mesh. Avoiding such triangles

is crucial for the accurate interpolation of elevation on the TIN. The DT of a point set *P* defines a triangle mesh Σ on the plane, of which the vertices are the points in *P*, and the circumcircle of any triangle belonging to Σ does not contain other vertices of Σ in its interior. The domain of the DT of *P* is the convex hull of *P* [42]. A TIN consists of a mesh Σ generated with DT, elevations defined on the vertices of Σ, and an interpolating piecewise linear function [183]. The DT has been used for surface reconstruction, scientific visualization, and terrain modeling. Many algorithms have been proposed for efficient calculation of a DT [27, 28, 61, 180].

The connectivity relations of a triangle mesh describe the adjacency and incidence of simplices on the meshes. Given a vertex v, an edge e, and a triangle t on a triangle mesh:

- the *Vertex-Triangle* (VT) relation of v is the set of triangles incident on v;
- the *Vertex-Edge* (VE) relation of v is the set of edges incident on v;
- the *Vertex-Vertex* (VV) relation describes the adjacency among vertices on a triangle mesh, that is, given a vertex v, the set of mesh vertices adjacent to v;
- the *Edge-Triangle* (ET) relation of edge e describes the incidence relation between e and two triangles sharing e;
- the *Edge-Vertex* (EV) relation of edge e refers to the two vertices forming edge e;
- the *Edge-Edge* (EE) relation describes the adjacency relation among edges, that is, given an edge e, the set of edges sharing one vertex with e on the mesh;
- the *Triangle-Vertex* (TV) relation of triangle t refers to three vertices forming t;
- the *Triangle-Edge* (TE) relation of triangle t refers to three edges forming t;

- the *Triangle-Triangle* (TT) relation describes the adjacency relation among triangles, that is, given a triangle t, the set of triangles sharing one edge with t on the mesh.

2.1.2 Surface topology

2.1.2.1 Discrete Morse theory

Let us consider a domain $D \subseteq \mathbb{R}^2$ and a smooth scalar function f defined over D. A point $p \in D$ is called a *critical point* of f if and only if the gradient of f vanishes at p [189]. The determinant of the Hessian matrix $Hess_p(f)$ of the second order partial derivatives of f, evaluated in p, provides additional information about the critical points of f. The number of negative eigenvalues of $Hess_p(f)$, called the *index* of p, defines the type of the critical point. For a bivariate function, a critical point of index 0 is a *minimum*, a critical point of index 1 is a *saddle*, and a critical point of index 2 is a *maximum*. For each critical point p, the eigenvectors of $Hess_p(f)$ define the directions in which function f decreases. An *integral line* of a function f is a maximal path that is everywhere tangent to its gradient. Each such line follows the direction in which the function has the maximum growth, starting at a critical point of f and ending at another critical point of f. *Morse theory* has been developed for smooth functions f such that all the critical points of f are non-degenerate, i.e., $Hess_p(f) \neq 0$. In such cases f is said to be a *Morse function* [133].

The set of critical points and integral lines of a Morse function define a decomposition of D based on the regions of influence of the critical points. If we consider a critical point p of index k, the integral lines converging at p form a k-cell, called the *descending k-manifold* of p. The collection of all descending manifolds form the *descending Morse complex*. Dually,

integral lines originating at p form a $(2-k)$-cell, called the *ascending k-manifold* of p. For a bivariate function, a descending 2-manifold corresponds to the region of influence of a maximum, a descending 1-manifold corresponds to the region of influence of a saddle, and an ascending 2-manifold corresponds to the region of influence of a minimum.

Discrete Morse Theory (DMT) [76] is a combinatorial counterpart of Morse theory which extends the results of Morse theory to discrete data, and thus it has been used for analysis of 2D and 3D scalar fields [69, 155, 191]. We consider a pair (Σ, F), where Σ is a triangle mesh and $F : \Sigma \to \mathbb{R}$ is a scalar function defined on all the simplices of Σ. Function *F* is a *discrete Morse function* (also called a *Forman function*) if and only if, for every k-simplex $\sigma \in \Sigma$, all the $(k-1)$-simplices on the boundary of σ have a lower function value than σ, and all the $(k+1)$-simplices bounded by σ have a higher function value than σ, with at most one exception. If there is such an exception, it defines a pairing of cells, called a *gradient pair*. A gradient pair can be viewed as an *arrow* formed by a head (k-simplex) and a tail ($(k-1)$-simplex). A simplex involved in no pair is called a *critical* simplex. We call a *V-path* a sequence of simplices $[\sigma_0, \tau_0, ..., \sigma_i, \tau_i, ..., \sigma_q, \tau_q]$ such that σ_i and σ_{i+1} are on the boundary of τ_i and (σ_i, τ_i) are paired, where $i = 0, ..., q$. A V-path is said to be *closed* if $\sigma_0 = \sigma_r$, and *trivial*, if $r = 0$. The collection of all paired and critical simplices of Σ forms a *discrete Morse gradient* (also called a *Forman gradient*) if there are not closed V-path. A *separatrix V_j-path* is a V-path connecting two critical simplices of dimension $j+1$ and j, respectively.

If we consider a scalar function f defined at the vertices of a triangle mesh Σ, we can extend it to a Forman function F defined on all simplices of Σ by setting the function value of a simplex as the highest function value among its vertices [155]. In a computation environment, we compute Forman gradient V without computing the Forman function F explicitly. In a triangle

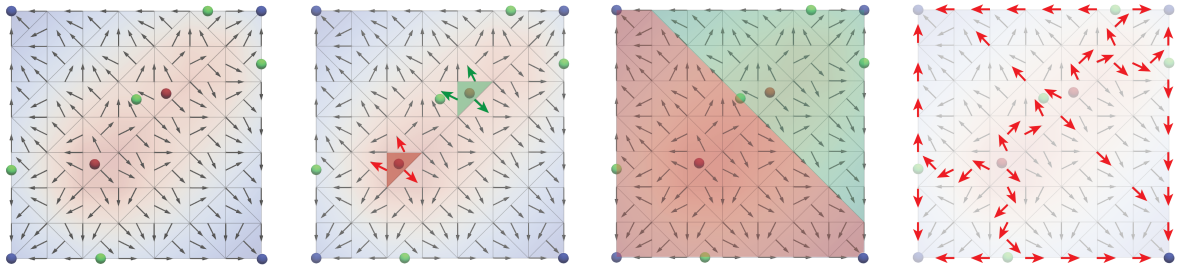
mesh, we have arrows formed by a triangle and an edge (*triangle-edge pair*) and by an edge and a vertex (*edge-vertex pair*). There are three types of critical simplices in a triangle mesh: critical triangles indicating maxima, critical edges indicating saddles, and critical vertices indicating minima. Fig. 2.2(a) shows an example of a discrete Forman gradient computed on a triangle mesh. Red, green, and blue dots indicate critical triangles, edges, and vertices, respectively. Arrows indicate gradient vectors.

Critical simplices are the discrete counterpart of critical points. Note that separatrix V_j-paths are the discrete counterpart of the integral lines in the smooth case. In a triangle mesh Σ, we have separatrix V_0-paths connecting a critical edge to a critical vertex and separatrix V_1-paths connecting a critical triangle to a critical edge.

2.1.2.2 Topological features

Topological features are defined in the discrete case in terms of the Forman gradient and its paths. Any descending k-manifold is the collection of the k-simplices of Σ reached by the separatrix V-paths starting from a critical k-simplex. Dually, an ascending k-manifold is the collection of the $(2-k)$-simplices reached by the separatrix V-paths (visited backward) starting from a critical $(2-k)$-simplex. The collection of the descending and ascending manifolds forms the discrete ascending and descending Morse complexes, respectively.

On a triangle mesh Σ, the descending and ascending 2-manifolds can be considered as the regions of influence of critical simplices (i.e., maxima and minima), which can be used to define a segmentation of the triangle mesh Σ. Fig. 2.2(c) shows the regions of influence for two critical triangles (maxima) and Fig. 2.2(b) shows the process of computing such regions.



(a) (b) (c) (d)

Figure 2.2: (a) Forman gradient. Regions of influence of critical simplices are computed visiting the V-paths (b) at the critical simplices, and (c) until the whole region of influence is visited. (d) V-paths connecting pairs of critical simplices.

This segmentation technique has revealed fundamental in many application domains, including material science [90], chemistry [141], environmental science [186], forest monitoring [195], and urban analysis [51], to mention a few. In terrain analysis, in particular, the segmentation of a terrain according to its critical points (i.e., peaks and pits) provides information regarding terrain morphology, which are fundamental for assessing the risk of landslides or floods.

On a triangle mesh Σ, an descending 1-manifold corresponds to a separatrix V_0-path and an ascending 1-manifold corresponds to a separatrix V_1-path. The collection of all critical vertices, edges, and triangles plus the separatrix V_0- and V_1-paths connecting them forms the *critical net*. Fig. 2.2(d) shows the gradient paths connecting pairs of critical simplices (i.e., critical net).

Based on the critical net, we define a labeled graph G called *surface network*, in which the nodes correspond to critical simplices, and have their highest elevation associated with, and the arcs represent separatrix V-paths connecting them. According to discrete Morse theory, the arcs connect saddles to minima or saddles to maxima [133], referred to as saddle-minimum and maximum-saddle arcs, respectively. The geometry of a maximum-saddle arc is a chain of line segments connecting the centroids of triangles, defined by the triangle-edge pairs in a triangle mesh. Such a chain goes from one of the three triangles adjacent to a maximum (critical triangle),

proceeds along the separatrix V_1-paths, and ends with a triangle incident in a saddle (critical edge). For instance, the green triangles in Fig. 2.3 are triangles along the separatrix V_1-paths connecting maxima to saddles in the Forman gradient and the red lines connecting the centroids of such triangles are the geometries of all maximum-saddle arcs in the surface network *G*. Similarly, the geometry of a saddle-minimum arc is a chain of line segments connecting the vertices and centroids of edges, defined by the edge-vertex pairs in a triangle mesh. Such a chain goes from the two extreme vertices of a saddle, proceeds along the separatrix V_0-paths, and ends with an edge incident in a minimum (critical vertex).

2.1.2.3 Persistence diagram

Topological descriptors offer representations of a terrain topology well suited for analytical comparisons. The most widely used topological descriptor is the *persistence diagram*, a compact representation rooted in the theory of *persistent homology* [58].

Formally, a persistence diagram is a multi-set of points representing all critical simplices of the terrain. Each point p, with coordinates (b, d), in the persistence diagram corresponds to a pair of critical simplices p_1 and p_2 connected by a *separatrix V-path* (e.g., a minimum-saddle pair or a saddle-maximum pair). The x-coordinate is called the birth of the point p, which is defined as $b = f(p_1)$; the y-coordinate is called the death of the point p, which is defined as $d = f(p_2)$. The difference between the death and the birth of the point p, denoted as *persistence*, measures the importance of the pair of critical simplices p_1 and p_2. Fig. 2.4 shows an example of a persistence diagram containing four points. For example, the blue point has coordinates (3, 4) and its persistence value is 1.

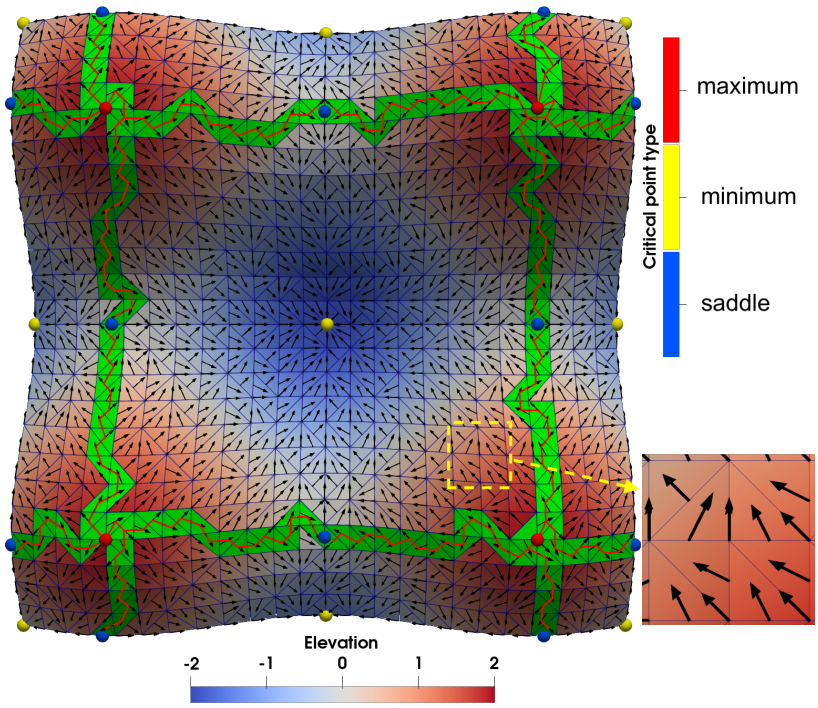
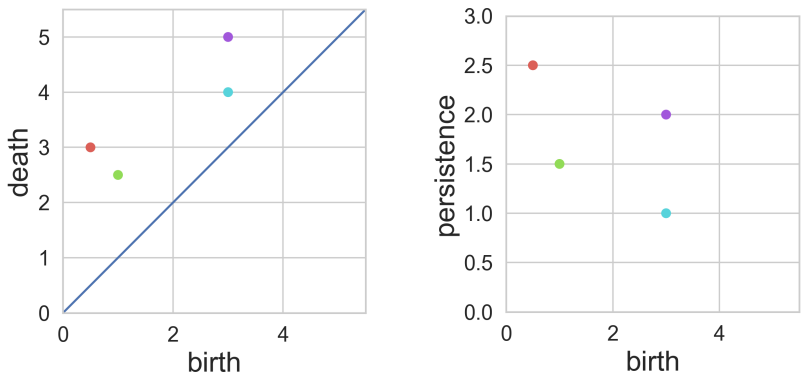


Figure 2.3: Example of Forman gradient calculated from a surface with an elevation function. Spheres on the figure correspond to the centroids of critical vertices, edges, and triangles in the Forman gradient. Red spheres are maxima (critical triangles), blue spheres are saddles (critical edges), and yellow spheres are minima (critical vertices). The green triangles are the triangles along the separatrix V-paths between maxima and saddles. The red lines connecting the centroids of such triangles correspond to the geometry of maxima-saddle arcs in the surface network G.

2.2 Review of state-of-the-art

In this section, we review state-of-the-art research related to this dissertation, including data representation for TINs, triangle mesh simplification methods, topology-based terrain analysis and simplification, and sea ice topography feature extraction.



(a) (b)

Figure 2.4: Example of (a) a persistence diagram formed by four points and (b) its birth-persistence chart. Points with the same color in (a) and (b) correspond to the same point.

2.2.1 Data representation for triangle-based terrain models

TINs can be generated from LiDAR point clouds without being interpolated into raster-based terrain models, but their usage are limited by the large storage costs. Using more efficient data structures can encode the connectivity relations on a TIN with less storage costs while efficiently supporting terrain analysis.

Connectivity-based data structures extend graph-based data structures for supporting the efficient extraction of connectivity relations (see [45] for a survey). A variety of connectivity-based data structures have been developed in the literature for triangle meshes [45]. The most widely used data structures are *triangle-based* ones, which encode the vertices and the triangles of the mesh but not the edges. *Edge-based* data structures are also used for triangle meshes, but they have been shown to be more verbose than triangle-based ones [45], while providing the same computational performances.

The most compact triangle-based representation is the *indexed data structure*, which encodes the vertices and the triangles of the mesh, and, for each triangle, the references to its ver-

tices. The *Indexed data structure with Adjacencies* (IA data structure) [145] extends the indexed data structure by explicitly encoding the adjacencies between triangles through the *TT* relation and, for each vertex, the index of a triangle incident into it. Other triangle-based data structures, such as the *Corner Table (CoT)* [158], the *Sorted Opposite Table (SOT)* [89], and the *SQuad* data structures [88], either require more storage or cannot support the efficient retrieval of certain topological relations.

Hierarchical spatial indexes use a recursive subdivision of the space on which the objects of interest are embedded according to different refinement rules. In 2D space, there are two types of refinement rules, one is provided by quadtrees in which each time a block is subdivided into four regular subblocks, another is provided by kD-trees in which each time a block is bisected by axis-aligned lines. The hierarchical decompositions have been originally defined for indexing point sets. They subdivide the space into blocks of equal size, generating *Point-Region (PR)* quadtrees and kD-trees [144]. The class of *Polygonal Map* (PM)-quadtrees [161] extends the PR-quadtree to represent polygonal maps in 2D space. There are three variants of a PM-quadtree, namely the *PM_1-quadtree*, the *PM_2-quadtree* and the *PM_3-quadtree*, which adapt different subdivision rules. The *Randomized Polygonal Map (PMR)-quadtree* [93, 136] is an index for collection of line segments in the plane (not necessarily forming a polygonal map).

A first attempt to extend the quadtree to index triangle meshes is the *PM_2-Triangle quadtree (PM2T-quadtree)* [43], in which triangles guide the partition into blocks. However, the PM2T-quadtree has two fundamental limitations. The first one is that each block indexes just one vertex, leading to a very deep hierarchy. The second limitation is that the spatial index is stored on top of the IA data structure, leading to a verbose representation, which greatly limits its scalability.

2.2.2 Triangle mesh simplification

The task of mesh simplification has been extensively studied in the literature [28, 91, 160]. In this section, we provide an overview of some techniques and challenges in triangle mesh simplification.

Popular techniques for mesh simplifications include vertex decimation [163], edge collapse [95], vertex clustering [157], and triangle collapse [96]. *Vertex decimation*, also referred to as *vertex removal*, consists of removing from the mesh a vertex and all the triangles incident at it and triangulating the "hole" left behind. *Edge collapse*, also known as *edge contraction*, involves contracting an edge to a single vertex. When the edge is contracted to one of its endpoints, this operation is called *half-edge-collapse*. *Vertex clustering* consists of grouping nearby vertices into clusters and representing each cluster with a few new points. *Triangle collapse* removes a triangle from the mesh by collapsing it into one vertex.

After selecting a simplification operator, one should define the order in which simplifications are performed. When considering edge contraction, a simple metric is to always contract the shortest edge [194]. Several more sophisticated metrics have been defined for optimizing the quality of output meshes, such as minimizing an energy function [95], setting a threshold on the Hausdorff distance between the original and the simplified models [103], minimizing the error quadrics [83], or setting a threshold on the error volume [86]. The *Quadric Error Metric* (*QEM*) [83] is one of the few approaches which keeps a good balance between computational costs and resulting mesh quality. These metrics are designed to optimize the geometric quality of the simplified mesh, but a simplification operator following them can still create non-valid meshes. To avoid such a situation, quality conditions, like the *link condition* [49] and *fold condi-*

tion [25], have been proposed.

Processing large terrains presents multiple issues. First, encoding the original TIN is memory-demanding. Second, performing a large number of simplifications sequentially is time-consuming [99] and directly affects user interactions during data exploration.

One solution for handling memory limitation is to use out-of-core partitioning methods to produce sub-meshes that can be processed by a single computer [20, 118, 119]. While these methods overcoming the limitation on memory, they do not reduce the memory costs of the simplification process itself and incur extra I/O time compared to in-core methods. In our research, we focus on in-core methods and use efficient representations to minimize memory requirements.

To expedite mesh simplification, many parallel mesh simplification algorithms have been developed. While a few of them focus on vertex decimation [77] and vertex clustering [46], most parallel approaches rely on the edge contraction operator due to its intrinsic flexibility [20,47,77, 85,114,146]. These latter methods can be roughly categorized into two classes.

The first class defines heuristics to prevent concurrent contractions of adjacent edges. Franc et al. [77] firstly introduced the concept of *super-independent* vertices, and a set of such vertices, named a *super-independent set*, can be contracted simultaneously. Papageorgiou and Platis [146] improved this algorithm by parallelizing the identification of super-independent vertices. However, such an algorithm is still complex and time-consuming, since only a subset of the vertices is processed at each iteration, and a pre-processing step is required every time to find a super independent set. Grund et al. [85] devised a new method in which an edge e can be contracted only if all the other edges adjacent to e have higher costs according to QEM. This method ensures that no adjacent edges of e (i.e., edges sharing a common vertex with e) can be contracted at the same time. While this approach is fast and efficient when only the edge cost is considered, it cannot

avoid possible conflicts in parallel mesh simplification when the check of quality conditions (e.g., link condition and fold condition) involves more than the adjacent edges.

A different strategy is partitioning the mesh into sub-meshes that can be then processed independently. The main advantage of this strategy is that it supports better quality control for the simplified mesh. Dehne et al. [47] partitioned the mesh by dividing the vertices of a mesh into subsets without duplicates, ensuring edges with endpoints in different subsets are not contracted. Lee and Kyung [114] introduced a new parallel edge contraction method with a lazy-update technique. The idea is to symbolically simplify the mesh while working in parallel and then encode the updates at the end of the simplification process. Cabiddu et al. [20] designed a parallel algorithm for distributed systems. The mesh is divided by a binary space partitioning [78] and the resulting sub-meshes are then grouped into independent sets. As a result, each group of sub-meshes does not share any element with the others. Recently, Mousa and Hussein [134] proposed a method for parallel mesh simplification, in which the mesh is subdivided into disjoint blocks with a k-d tree. Each block is processed independently by removing edges based on the order of edge costs encoded in a global queue. This method has then the same contraction order as a method simplifying directly the triangle mesh. If an edge e intersects two blocks, then only one of the two contracts e, while the other is locked and cannot contract any other edge until the contraction of e is completed. However, the locking strategy defined in this method does not avoid conflicts between blocks when the check of quality conditions involves also the triangles in the neighborhood of e.

2.2.3 Topology-based terrain analysis and simplification

The key aspect of the methods rooted in discrete Morse theory is that of computing a discrete Morse (Forman) gradient. Among the many algorithms defined for computing a Forman gradient [44], the most efficient ones are those computing the gradient from a function sampled at the vertices of a triangle or tetrahedral mesh, or of a regular grid. The algorithm described in [102] is based on a divide-and-conquer approach and has the main drawback of introducing many spurious critical simplices. A similar approach, based on a weighted discrete function, has been defined by Shivashankar and Natarajan for computing a Forman gradient on 2D regular grids [166]. The algorithm is well suited for parallelization and significantly reduces the number of spurious critical cells. Robins et al. proposed an algorithm for 3D regular grids that processes the *lower star* of each vertex independently [155]. The function value of a cell (i.e., an edge, an unit square, or an unit cube) is set as the highest function value of its vertices. The *lower star* of a vertex v is the set of cells in the star of v, whose function values are the same as v. This algorithm does not generate spurious critical cells. The algorithm has been extended to triangle [70] and tetrahedral meshes [191] by using a compact and efficient implicit encoding of the Forman gradient.

In Section 2.2.2, we reviewed different triangle mesh simplification methods with a focus on parallel simplification. However, those simplification methods do not consider the topology during the process. In another word, they can modify the number or connectivity of critical simplices of a terrain in an uncontrolled way. This problem is addressed by applying topology-preserving simplifications, i.e., a simplification combining the need to reduce the size of a mesh with the need to maintain its topological features, such as peaks, pits, ridges, and valleys.

Bajaj and Schikore [11] first introduced a simplification operator which preserves the critical points of the terrain. The operator removes a vertex from the terrain and re-triangulates the neighborhood in such a way that the remaining vertices maintain the same classification (i.e., minimum, saddle, maximum, non-critical). The method preserves the topology of the terrain, but it lacks an efficient implementation for re-triangulating the terrain.

The first efficient topology-aware operator based on edge contraction was introduced by Danovaro et al. [40]. This method preserves critical points connectivity by checking the separatrix lines incident at the endpoints of a contracted edge. However, while preventing the removal of existing critical points, it does not avoid the creation of new ones. The first topology-aware simplification based on discrete Morse theory was introduced by Iuricich and De Floriani [99]. The basic operator, called *gradient-aware edge contraction*, is able to preserve both the critical simplices and their connectivity by using a Forman gradient as the underlying descriptor of the terrain topology. Before contracting an edge e, the operator checks the gradient pairs in the neighborhood of e. If these pairs are organized in a valid configuration, the contraction guarantees the preservation of the terrain topology. On top of that, a multi-resolution model called a *Hierarchical Forman Triangulation (HFT)* is introduced. This model combines geometric and topological operators to enable mesh simplification or refinement by varying the resolution of both topology and geometry on-demand. This is fundamental for interactive exploration of a mesh dataset in real time.

Building upon the gradient-aware simplification operators, Dey and Slechta [50] relaxed the original criteria to allow the removal of more critical simplices. While this operator does not lead to a multiresolution model like HFT, it increases the number of admissible edge contractions, and thus, the compression factor. Recently, a new approach based on vertex removal was proposed

by Fugacci et al. [79]. The simplification operator is similar to the one introduced by Bajaj and Schikore [11], but in this case, the topology is preserved by checking a descriptor whose definition is rooted in algebraic topology, i.e., persistent homology [58]. A vertex removal is valid only if the link of the removed vertex can be re-triangulated while preserving the persistence diagram, which is proven to be equivalent to preserving critical points.

2.2.4 Sea ice topography measurements

2.2.4.1 Sea ice surface modeling

A variety of surveying methods, both in situ and remote sensing methods, have been applied to measure the topography of sea ice surfaces (see [62] for a detailed summary of those methods). In situ surveying methods, like laser profiling [64], drilling and level-based surveying [175], can provide a high-precision measurement of sea ice surface features. Most of them are one-dimensional surveying which cannot characterize two-dimensional surface properties [21, 187]. Besides, in situ surveying is labor-intensive and has a limited sampling extent, which cannot support large-scale topography analysis [62].

Laser altimeters deployed on the ICESat-1, -2 and Synthetic Aperture Radar (SAR) altimeters deployed on CryoSat-2, are important sources for sea ice elevation and thickness. Topography feature studies using satellite altimeters focus mainly on the sea ice surface roughness [105, 107]. More recent research demonstrates the ability of ICESat-2 data for extracting other topography features including pressure ridges, melt ponds, floes, and leads [67]. Those feature extraction methods are based on the profile along the scanning track. They cannot be simply extended to 2D feature extraction on ICESat-2 data considering the much larger distance

between adjacent tracks compared to the point distance along the track. Also, the difference between across-track and along-track resolution makes the direct generation of a 2D terrain model from the satellite data more difficult since suitable spatial and temporal interpolation methods are required. Another data source for sea ice elevation is the airborne laser altimeters. NASA Operation IceBridge (OIB) campaign collects a large volume of sea ice surface measurements by using the Airborne Topographic Mapper (ATM) [104]. The ATM swath, which can reach a width of 200-300 meters across the flight track, provides us with opportunities for extracting two-dimensional features from sea ice surfaces that cannot be easily extracted from linear profiles [150, 151].

2.2.4.2 Sea ice topographic features

From sea ice surface models, several topographic features can be extracted. In this section, we focus on sea ice topographic features that can be identified from surface models and review widely-used methods to extract such features.

Sea ice surface roughness is an important sea ice topography feature extensively studied in previous research [14, 67]. The knowledge of sea ice roughness is crucial for understanding snow distribution [97], parameterizing momentum transfer to the ocean [126], processing of remote sensing data collected from sea ice surfaces [109, 110], and evaluating sea ice surface trafficability [39]. Generally, higher sea ice roughness indicates more complex topographic features, and thus, lower trafficability. Sea ice roughness is influenced by ice deformation and, in return, has been used to identify topographic features such as ridges and leads [139] or to determine level sea ice [54].

There are different definitions of roughness depending on the scale studied and on the data format. In most sea ice studies, roughness is defined as the standard deviation of elevations, but the scales and ranges of elevations vary. For instance, in [67], roughness is defined as the standard deviation of sea ice heights within 25-km-long segments. In [14], roughness is the standard deviation of elevations for each scan line measured by the laser scanner, or the standard deviation of a five-data-point moving window for the single-beam altimeter. In [101], roughness is calculated as the standard deviation of elevation measurements within a 1.1 km resolution pixel.

Leads, defined as narrow linear fractures in the sea ice, are important for estimating local sea-level height. Leads are easy to identify in high-resolution imagery due to their significantly lower surface albedo and darker color compared to adjacent sea ice surfaces [117, 142]. Onana et al. [142] proposed an automated method to detect leads from Digital Mapping System (DMS) visible imagery by integrating minimal signal transforms across RGB channels. The advent of LiDAR altimeters gives rise to a new dimension to this field. Farrell et al. [66] proposed a method to identify leads in visible imagery by locating pixels with low brightness and to estimate lead elevations based on elevations of ATM points fall into such pixels. When only altimetry data is available, the lead elevation can be estimated by finding the lowest mode of the elevation distribution [68]. In [108] and [67], leads are identified from ICESat and ICESat-2 profiles by detecting flat areas around local minima.

Melt ponds are formed by snow melted and pooled in lower spots on sea ice surface. These ponds have low albedo [148] and affect the absorption of solar radiation into the sea ice-ocean system [63]. Therefore, understanding the distribution and depth of melt ponds is significant for the prediction of sea ice melting. Similar to leads, melt ponds can be easily identified from visible

images due to their low surface albedo. Melt ponds and open water (including leads) are usually distinguished based on whether water has melted through ice [19]. Classification methods have been proposed to automatically identify melt ponds and open water regions from visible [19] and optical imagery [193]. When only altimetry data is used for melt pond detection, a common strategy is to utilize the bimodal distribution of photons within the range of melt ponds [67]. This distribution pattern happens because the primary mode corresponds to photon returns from the water surface and the secondary mode corresponds to photon returns from the pond bottom [67]. Building on the same principle, Fair et al. [65] developed an algorithm to detect the meltwater surface from OIB ATM and ICESat-2 data based on the flatness of high-confidence photons, as these are likely collected from the water surface, while others may be collected from the pond bottom.

The collision between ice floes causes ice to pile up into *pressure ridges*, which form small "mountains" above the local sea-ice surface. Most large-scale studies on ridges rely on images [54,55,120,132] or LiDAR data [23,53,67,151,182]. Edge detection and object detection are the main approaches used in image-based studies to identify ridge lines. Miao et al. [132] detected the shadows from the aerial images to derive the locations and heights of ridges. A similar method was used by Duncan et al. [54, 55] to derive ridge sail heights from OIB DMS images, and the results are validated with OIB ATM data. Linow and Dierking [120] used edge detection to find the Linear Kinematic Features (LKFs), which can be leads, cracks and pressure ridges, from images. There is no elevation information in the images used by those methods, and thus, the ridge height can only be derived but not directly measured.

The direct measurements of ridge heights are usually collected with laser altimeters. Most existing studies on laser altimeters use laser profiles to identify ridges. One reason behind it is

that the laser profiler had been the major data source for LiDAR data in the past, and in some missions, only laser profile data is available. The most common way to identify ridges from profiles is to find local maxima along the profile [23, 53, 67, 182]. Depending on the objective, different cutoff heights are used to select valid ridges. The most widely-used one is 0.6 m above the level sea ice. Ridge extraction based on profiles, however, is limited to one direction. Thus, such methods suffer from the limitation that only ridges intersecting the profile track are observed and measured. Therefore, they cannot extract the ranges of complex, multi-directional ridges. With only laser profile data, a comprehensive view of ridges in a region can only be obtained by extrapolating from the one-dimensional result. Such an extrapolation relies on the assumption that the two-dimensional distribution of ridges has the same pattern as the distribution along the profile track, which may not always be the case.

Compared to profiles, LiDAR swaths offer a more comprehensive coverage, allowing for the extraction of two- or three-dimensional features of ridges. One recent study that leverages the three-dimensional coverage of LiDAR data to extract ridges was conducted by Petty et al. [151]. They grid OIB ATM points using a 2 m resolution horizontal grid and extract peaks as local maxima within the grid. A watershed method [168] was applied to extract the ranges of topography features whose peaks are such maxima. The features extracted by their method are considered as an intermediate between ridge links and full ridge structures [151]. While this method can effectively extract surface features, the ranges of extracted features are larger than those of ridges and it is not clear how to derive more accurate ridge ranges from such features.

Chapter 3: Terrain trees: a decomposition-based data representation for triangulated terrain modeling and analysis

3.1 Introduction

This chapter introduces a new compact data representation for triangle meshes, the *Terrain trees*, and a library for terrain analysis based on Terrain trees, named the *Terrain Trees Library (TTL)*.

The underlying idea behind Terrain trees was firstly introduced in a short paper presented at ACM SIGSPATIAL in 2017 [69]. In the work presented in this chapter, we extend this previous work by introducing and comparing three spatial indexes that together form the Terrain trees family. A Terrain tree encodes the topology of a triangulated terrain combined with a spatial index built on the triangle mesh. By encoding the vertices incident in each triangle as well as the field values associated with each vertex, a Terrain tree provides the minimum amount of information required for extracting the full mesh connectivity and for processing such fields locally. On the other hand, the spatial index provides the ability to navigate the triangle mesh at a global scale. We introduce methods for extracting morphological features, such as slope, curvature, and roughness, on Terrain trees. Moreover, we have developed distributed algorithms for computing and analyzing the topology of a terrain by using discrete Morse theory [76]. Such algorithms

require an efficient navigation of a triangle mesh, which is a challenging task in hierarchical and modular mesh data structures like the Terrain trees. Moreover, we discuss a multivariate measure for visual analysis of terrains with multi-field information. All those methods have been implemented in an open-source library TTL [73].

The work presented in this chapter is a collaborative work. To be more specific, my contributions to this work is listed as follow:

1. Designed algorithms for morphological terrain analysis and calculating multifield measure on Terrain trees and the *Indexed data structure with Adjacencies* (IA data structure) [145].
2. Designed and developed the algorithm for Forman gradient computation on Terrain trees.
3. Contributed to the development of TTL by implementing algorithms in (1) and (2) and optimizing connectivity relation extraction functions.
4. Have been maintaining TTL.
5. Have contributed to the development of python wrappers of TTL.
6. Conducted the entire experimental performance evaluation of Terrain trees.
7. Made substantial contributions to the writing and revising of the manuscript.

Based on our work on TTL, I gave a lightning talk at AGU 2021 Fall meeting [173]. A co-authored paper on Terrain trees and the library was published on Springer International Journal on Advances of Computer Science for Geographic Information Systems (GeoInformatica) in 2023 [72].

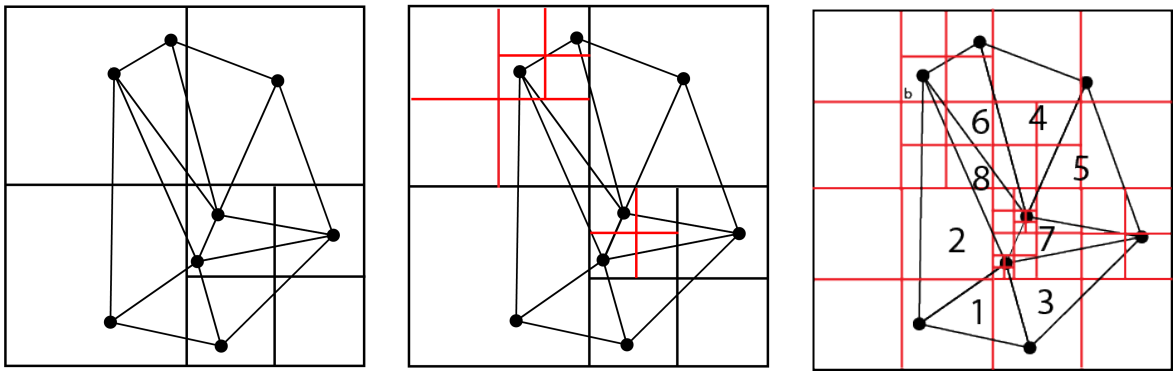
The remainder of the chapter is organized as follows. In Section 3.2, we present the Terrain trees, describing the different subdivision rules and how they are generated, and, in Section 3.3, their encoding structure and how to execute basic spatial and topological queries on them. In Section 3.4, we describe how to extract classical morphological terrain features, namely *triangle* and *edge slope*, *curvature* and *roughness* in the Terrain trees framework. In Section 3.5, we define how to perform topology-based terrain analysis on Terrain trees by using a Forman gradient. First, we introduce a distributed algorithm for extracting a Forman gradient field, and, then, a distributed procedure for extracting the critical net on the critical points of the gradient. In Section 3.6, we depict how multivariate visualization is performed in Terrain trees. In Section 3.7, we provide an experimental evaluation of the Terrain trees by comparing the performances of the different spatial indexes also against a state-of-the-art compact data structure for meshes. Finally, in Section 3.8, we draw some concluding remarks.

3.2 Terrain Trees

Terrain trees are a family of spatial data structures for triangulated terrains based on a nested subdivision of the terrain domain. Given a set P of data points, with each point being characterized by x and y coordinates plus an elevation value), we consider the projections of the points of P on the plane and we call D the square domain in the plane containing such projections. A *Terrain tree* built on P consists of:

1. a triangle mesh Σ connecting the projections on the plane of the points in P;

2. a quadtree describing the nested subdivision of the domain D into square blocks in such a way that the vertices and triangles of Σ are associated with the leaf blocks of the quadtree



(a) PR-T subdivision (b) PM-T subdivision (c) PMR-T subdivision

Figure 3.1: Given the triangle mesh of Fig. 3.2, a vertex capacity $k_V = 2$ and a triangle capacity $k_T = 2$, the Figure shows the spatial subdivision obtained with a PR-T Tree (a), a PM-T Tree (b) and a PMR-T Tree (c). In black are highlighted the spatial decomposition caused by the vertices capacity, while in red those caused by the triangles one. For the PMR-T tree we also highlight the triangle insertion order that drives the spatial decomposition (note that k_T is not a bucketing capacity for the PMR-T tree).

subdivision.

The association of the vertices and triangles with a leaf block is defined as follows. A vertex is associated with the only block containing it. A triangle is associated with all leaf blocks having a non-empty intersection with it. Note that a block is considered *closed* at the two edges incident in its lower-left corner, and *open* at the remaining two edges. More precisely, a block consists of all points (x, y) such that $x_1 \leq x < x_2$ and $y_1 \leq y < y_2$, where (x_1, y_1) is the lower-left corner and (x_2, y_2) is the upper-right corner of the block. Blocks having the upper or the rightmost edge on the boundary of *D* are closed on the corresponding edge.

Similarly to spatial indexes for 2D maps, we have defined different criteria for domain subdivision, based only on the TIN vertices, on the TIN triangles and on both vertices and triangles. Thus, the Terrain trees family consists of three spatial data structures, namely the *PR-Terrain tree (PR-T tree)*, the *PM-Terrain tree (PM-T tree)* and the *PMR-Terrain tree (PMR-T tree)*, which are bucketed versions of the PM_3-quadtree, of the PM_2-quadtree, and of the PMR-quadtree for maps,

respectively [136, 138, 161]. As we demonstrate in Section 3.7, bucketing is a crucial aspect for our spatial indexes, since it allows the indexing of much larger datasets, compared to existing representations in the literature.

A *PR-T tree* subdivides the domain D based on the vertices of Σ. Its subdivision rule uses a capacity k_V on the number of vertices contained in a leaf block b. If b contains more than k_V vertices, then b is recursively split into four blocks until this condition is met. An example of a PR-T tree is shown in Fig. 3.1(a). The generation of a PR-T tree is entirely guided by the vertices of Σ, and, thus, the first step in the generation process is exactly the same as for the PR-quadtree [159]. Then, each triangle t of Σ is added to all the leaf blocks intersecting t, without affecting the spatial decomposition.

A *PM-T tree* uses the same subdivision rule defined for the vertices of Σ as the *PR-T tree*. A splitting rule on the triangles is also defined, based on a capacity k_T on the number of triangles per leaf block, as follows:

(1) a block b containing up to k_T triangles is a leaf block;

(2) a block b that contains more than k_T triangles is a leaf block if and only if all triangles intersecting b are incident in the same vertex v, which can be either inside or outside b;

(3) otherwise, the block is recursively split until either condition (a) or (b) is met.

An example of the subdivision obtained with the PM-T tree is shown in Fig. 3.1(b). Note that a *PM-T tree* extends the PM_2 quadtree defined for maps, by adding bucketing capacities k_V and k_T for both vertices and triangles. The generation of the initial hierarchical decomposition in a PM-T tree is entirely guided by the vertices, like in PR-T trees. Then, each triangle t in Σ is

added to a leaf block *b* intersecting *t*, if and only if *b* contains less than k_T triangles. Otherwise, *b* is split until either condition (1) or (2) is met.

A PM-T tree is a sort of *bucketed* version of the PM2T-quadtree with some fundamental differences. The PM-T tree has a bucketing capacity for both vertices and triangles. The lack of a bucketing capacity in the PM2T-quadtree produces much deeper decompositions with a number of leaf blocks that is at least four times the number of vertices in Σ (see [43] for details). The PM-T tree uses a simple indexed representation encoding only *Triangle-Vertex* relations while the PM2T-quadtree encodes the triangle mesh through the IA data structure (see Section 3.3).

The subdivision strategy for the *PMR-T tree* is driven only by the triangles of Σ. It extends to triangle meshes the subdivision approach defined for sets of segments in the plane in [136]. A leaf block *b* is split if *b* intersects more than k_T triangles, where k_T is a user-defined capacity, but b is split only once, not recursively. The decomposition and, thus, the shape of the tree depends on the insertion order of the triangles. In [136] it has been proven that in a PMR quadtree built on the edges of a map, the number of edges intersected by a leaf block cannot exceed the sum of the splitting capacity and of the depth of the leaf block. For a PMR-T tree this result still holds and, thus, the number of triangles in a leaf block of a PMR-T tree can be at most equal to $d_b + k_T$, where d_b is the depth of the leaf block and k_T is the splitting capacity. An example of PMR-T tree is shown in Fig. 3.1(c). For instance, in leaf block *b* there are three triangles indexed by it (i.e., triangles 2, 6 and 8), but no split operation is triggered, as the space is decomposed once when inserting triangle 8. Also condition $d_b + k_T$ is verified as *b* is at depth 3. Notice that the split condition may trigger unnecessary splits. For example, in Fig. 3.1(c) the insertion of triangles 7 and 8 causes unnecessary split operations on the vertices sharing these two triangles.

A *PMR-T tree* is generated as follows. For each triangle *t*, and for each leaf block *b* inter-

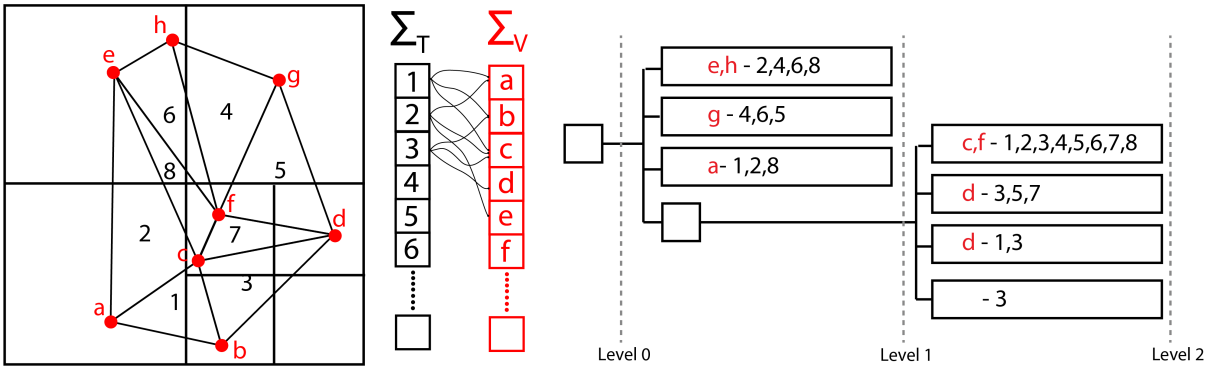
secting *t*, *t* is added to *b* if and only if *b* contains less than k_T triangles. Otherwise, *b* is split and its triangles are distributed to the newly generated leaf blocks (i.e., the children of *b*). Triangle *t* is also added to the children of *b* intersecting it.

3.3 Implementation of Terrain trees

In this section, we describe the implementation of the Terrain trees in the *Terrain trees library* (TTL). The kernel of the tool contains the implementation of the three Terrain trees, the *PR-Terrain tree (PR-T tree)*, the *PM-Terrain tree (PM-T tree)* and the *PMR-Terrain tree (PMR-T tree)*, plus their generation algorithms and algorithms for answering basic spatial and connectivity-based queries. Other functions currently implemented and discussed in the following sections are the extraction of morphological features (see Section 3.4), the extraction of topology-based features (see Section 3.5), and the analysis of multivariate terrain data (see Section 3.6).

The encoding of the triangle mesh Σ in a Terrain tree consists of two arrays Σ_V and Σ_T, storing the vertices and the triangles of Σ, respectively. Each vertex *v* and triangle *t* is represented by a unique index, i_v and i_t within arrays Σ_V and Σ_T, respectively. Σ_V encodes the geometry of the terrain Σ by storing the longitude, latitude, and elevation and the other field value(s) associated with each vertex *v* in Σ. Σ_T encodes the connectivity of each triangle by storing its three vertex indexes (see Fig. 3.2(b)).

We use a pointer-based representation for the hierarchy describing the nested subdivision of a Terrain tree (see Fig. 3.2(c)). Each internal block of a Terrain tree contains a reference to its parent block and a reference to its children. Each leaf block contains a reference to its parent



(a) PR-T tree (b) Mesh (c) Spatial hierarchy

Figure 3.2: (a) Triangle mesh and PR-Tree as encoded in a Terrain tree. (b) The mesh topology is organized by encoding, for each triangle, the boundary relation with its vertices. (c) The PR-T represents a hierarchy where each leaf block encodes the list of vertices contained in the block and the list of triangles intersecting the block.

block plus the information about the vertices and triangles, or only the triangles in the case of a PMR-T trees.

To encode the information associated with the leaf blocks, we use the compact encoding proposed in [74] for an arbitrary-dimensional connectivity-based data structure for simplicial complexes based on a vertex clustering. Such encoding uses the *sequential range encoding (SRE)*, a variant of the *run-length encoding* [92], that represents a run of consecutive indexes using two integers. The first (negative) index encodes the starting index of the run, while the second encodes the number of remaining elements of the run. The effectiveness of this compression increases with longer runs. This is exploited by representing all vertices inside a leaf block with a single run. Once we obtain the spatial decomposition, a single tree traversal is sufficient to reindex the vertices. The vertices indexed into the same leaf block get a contiguous range of indexes in the reindexed vertices array Σ_V. Within each leaf block, this range is represented as a pair of integers. Exploiting the spatial coherence for the triangles is more involved. The reindexing and compression of triangles is performed in such a way that, at the end, triangles indexed by the

same set of leaves have contiguous indices in Σ_T. To obtain this representation, we traverse first the tree to extract, for each triangle *t*, the tuple of leaf blocks indexing *t*. Then, we extract the dual relation, i.e., we associate the list of triangles with each tuple of leaf blocks. Given this inverted relation, we extract a coherent ordering for the triangles of Σ_T, where triangles indexed by the same leaf tuple have contiguous indexes. Once we have this spatial ordering on the triangles, we apply it to the triangle list of each leaf block, and we compress this list by using the SRE compression. Finally, we update Σ_T to be consistent with this spatial ordering.

3.3.1 Spatial and connectivity-based queries in a Terrain tree

We have developed algorithms on Terrain trees for answering two fundamental spatial queries, namely a *point location* and a *window* query, and for extracting connectivity-based relations (described in Section 2.1.1), which enable the local traversal and processing of the underlying mesh. The point location query consists of finding the triangle (or triangles) containing a given query vertex, while the window query consists of finding all the triangles which intersect an axis-aligned rectangular window.

We have implemented algorithms for connectivity-based queries which extract the connectivity relations discussed in Section 2.1.1. Since the indexed TIN representation underlying a Terrain tree encodes the TIN vertices and triangles, extracting the vertices and edges of a triangle (these latter being expressed as vertex pairs) does not require the use of the tree structure unless we combine the extraction of such information with a window query when focusing on portions of the TIN. Extracting co-boundary relations at the vertices of the mesh efficiently and massively, as our experiments show, are fundamental for computing morphological features, the

discrete Forman gradient estimators, and the critical net (see Sections 3.4 and 3.5). We describe here how we extract some fundamental co-boundary relation, namely the *Vertex-Triangle (VT)*, the *Vertex-Vertex (VV)*, and the *Vertex-Edge (VE)* relations, and the *Edge-Triangle (ET)* relation, as well as the *Edge-Vertex (EV)* relation inside a leaf block of the Terrain tree. These are used in the terrain analysis algorithms presented in the following sections.

Extracting the *VT* relations in a block *b* requires knowing the set of vertices contained in *b*. The range of indexes of the vertices contained in any given block *b* is explicitly encoded in the PR-T and PM-T trees. A PMR-T tree encodes only the triangles indexed by a block *b*. In this case, the set of vertices in *b* are extracted by performing a *point-in-block* test for each of the bounding vertices of the triangles in *b*. Then, the *VT* relation for the vertices in block *b* is extracted by cycling over the set of triangles in *b*. For each triangle t, the algorithm iterates through the vertices of t. For each vertex v of t, if v is indexed by *b*, t is added to the list of triangles incident in v. The strategy for extracting the *Vertex-Vertex (VV)* and *Vertex-Edge (VE)* relations in a block *b* combines the VT relation with either the *Triangle-Vertex (TV)* relation, which provides the list of the vertices of a given triangle, or the *Triangle-Edge (TE)* relation, which provides the list of the edges of a given triangle. For the *VV* relation, for each vertex v of a triangle t, if v is indexed by *b*, we add the other two vertices in the boundary of t, namely v_i and v_j, to the set of vertices adjacent to v. Similarly, for the *VE* relation of v, we pair v_i and v_j with v to get the two edges of t that are incident in v.

Extracting the *Edge-Triangle (ET)* relations in a block *b* is slightly more involved, since the edges are not explicitly encoded in a Terrain tree. The algorithm iterates over the triangles in a block *b* and extracts the edges on their boundary provided by the *TE* relation. An edge e belonging to the boundary of a triangle t is considered *internal* to *b* if it has at least one vertex

indexed by *b*. Each internal edge e is encoded in a local associative array having as key e and as value a pair containing the index of the two triangles in the co-boundary of e. These two triangles are identified during a single iteration on the triangles in *b*. The strategy for extracting the *Edge-Vertex (EV)* relation in a block *b* is similar to the one for extracting the ET relation. The algorithm iterates over the triangles of *b* and extracts their edges. Since each edge is encoded as a pair of vertex indices, we add those edges having at least one vertex indexed by *b* to the output.

3.4 Morphological Terrain Features

We have developed and implemented in the *Terrain trees library* algorithms which extract classical morphological terrain features, namely *triangle* and *edge slope*, *curvature* and *roughness*. An experimental comparison among the various Terrain trees and the IA data structure in computing such features is presented in Section 3.7.

The *slope* of an edge or a triangle in a TIN represents the steepness and the direction of its extent, where the steepness is the absolute value of the slope. A zero value of the slope indicates horizontality. The direction is increasing if the slope is positive and decreasing if the slope is negative. Specifically, the *edge slope* of an edge e is the angle between e and its projection on the horizontal plane defined by the z-coordinate of its lowest endpoint. In other words, given $v_i = \{v_{i_x}, v_{i_y}, v_{i_z}\}$ and $v_j = \{v_{j_x}, v_{j_y}, v_{j_z}\}$ the endpoints of e. Let v_i the endpoint of e with the minimum z-value, we consider the projection v'_j of vertex v_j on the plane $z = v_{i_z}$. The slope of edge e is the angle $\widehat{v_j v_i v'_j}$. For computing edge slopes, we need to extract the edges as pair of vertices, by using the *TV* relation, since the edges are not explicitly encoded. Extracting the edges requires an auxiliary data structure. The decomposition of the mesh defined by the

blocks becomes computationally relevant, since the auxiliary data structure is created in each block independently and discarded after processing the block. This makes it possible to compute slopes in a region of interest without iterating through the entire domain.

In a similar way, the *triangle slope* is defined as the angle between the normal to the plane to which the triangle belongs and a vector aligned with the z-axis. The computation of triangle slopes requires only the *Triangle-Vertex (TV)* relation for each triangle, which is stored in the triangle array of a Terrain tree.

We approximate the curvature at the vertices of a TIN by using a discrete approach. In our previous work [129,130], we have developed three discrete curvature approximations of Gaussian and mean curvature, and compared and evaluated them in [128] for curvature estimation on a TIN. Our results showed that all curvature estimators provide similar results, also when used as the basis for TIN segmentation, and that *concentrated curvature* is the least sensitive to noise.

We consider a TIN Θ and a vertex v of Θ. Let $t_1, ..., t_n$ the triangles incident in v. Let v_i and v_i' the two vertices in the triangle t_i, different from v. The concentrated curvature is defined as $K_c(v) = 2\pi - \Lambda_v$, for internal vertices, and $K_c(v) = \pi - \Lambda_v$, for boundary vertices, where $\Lambda_v = \sum_{i=1}^{n} \widehat{v_i v v_i'}$.

The computation of curvature requires extracting, for each vertex v of the TIN, the set of triangles incident at v, i.e., extracting the *Vertex-Triangle* relation, which is performed as discussed in Section 3.3.1. As the internal vertices and boundary vertices have different equations for estimating concentrated curvature, a preprocessing step to identify boundary vertices is performed. For each vertex v, we check the number of edges $|e|$ and the number of triangles $|t|$ incident in v. A vertex is on the boundary when $|e| \neq |t|$.

There are several ways to define surface roughness, and the most commonly used is the

standard deviation of local elevation at each vertex, evaluated based on the neighbors of the vertex itself [165]. We have extended the definition that is given for raster grids to TINs by considering the vertices adjacent to a vertex v, which are the ones sharing an edge with v. Based on that definition, the *roughness* at a vertex v in a TIN is computed as:

$$R(v) = \sqrt{\frac{\sum_{i=1}^{m}(z_i - \overline{z})^2}{m-1}} \tag{3.1}$$

where m is the number of vertices adjacent to v plus v itself, z_1, z_2, ..., z_m are the elevations at such vertices, and $\overline{z}$ is the average of those elevations. From Equation 3.1, the roughness computation requires the extraction of the VV relation of v (see Section 3.3.1 for details).

3.5 Topology-based Terrain Segmentation

The basis for terrain analysis is a segmentation of the terrain based on its critical points, their regions of influence and how they are connected together in the critical net. The approach we consider here is rooted in discrete Morse Theory, which supports an efficient computation of a discrete gradient on large meshes and the efficient computation of topological descriptors like the Morse decompositions and the critical net.

In Subsection 3.5.1, we present the general strategy for computing a discrete Morse (Forman) gradient and a distributed approach based on Terrain trees. In Subsection 3.5.2, we discuss how to compute the critical net and present an algorithm for Terrain trees.

3.5.1 Forman gradient computation

We consider a triangle mesh Σ and an elevation function $f : \Sigma_V \longrightarrow \mathbb{R}$ defined on the vertices of Σ. The algorithm for computing the Forman gradient is based on the extension to TINs of the algorithm proposed for regular grids [155]. It consists of three major steps, which are described below and illustrated by referring to Fig. 3.3.

Step 1 (Indexing). The first step requires computing a total order I on the vertices of Σ. The total order will serve as guiding schema for the subdivision of the triangles, edges and vertices of Σ in independent sets. This is done by *Simulation of Simplicity* [60], i.e., by sorting the vertices of Σ in ascending order and by assigning a unique index to each of them. In Fig. 3.3(a) we indicate the index in I of each vertex of a triangle mesh. On Terrain trees, *Step 1* is executed by sorting the vertices stored in the global vertex array.

Step 2 (Partition). Σ is then subdivided by associating each vertex v with the set of edges and triangles having the same value of I as v. I is extended to the edges and triangles of Σ via $I(\sigma) := \max_{v \in \sigma} I(v)$, where σ is either an edge or a triangle of Σ and v is a vertex on the boundary of σ. For each edge or triangle σ, we denote as v the vertex of σ with maximum value of I. For this reason, this set of triangles and edges associated with v is called the *lower star* of v according to I and denoted $L_I(v)$. It can be proved that each triangle or edge in Σ belongs to exactly one lower star. Then, the lower stars associated with the vertices form a partition of Σ and thus they can be processed in parallel. In Fig. 3.3(a) the lower star of vertex 6 is depicted with bold lines.

On Terrain trees, *Step 2* is implemented through a tree traversal where each leaf block *b* is processed once. In a leaf block *b*, the algorithm extracts the lower star $L_I(v)$ for each vertex v

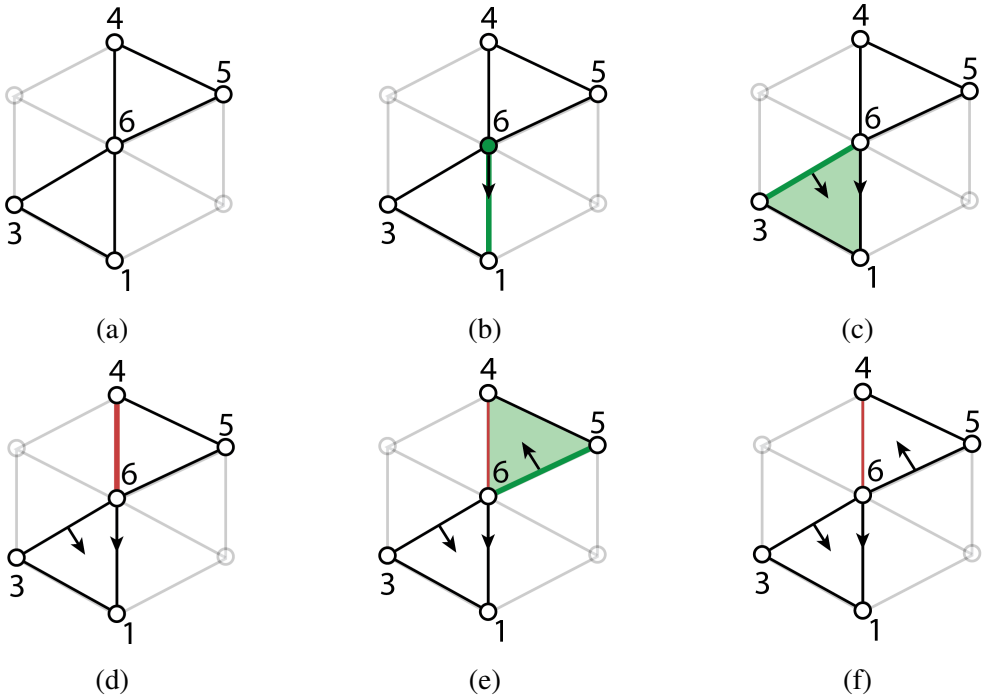


Figure 3.3: Homotopy expansion [155] computed on the simplices belonging to the lower star of vertex 6.

in b by retrieving the set of triangles incident in v (*Vertex-Triangle (VT)* relation) and the set of edges *(Vertex-Edge (VE)* relation), and by computing their values of I at runtime.

Step 3 (Pairing). Pairings of edges and vertices, and edges and triangles are computed through a process called *homotopy expansion* on each lower star. Recall that the Forman gradient is a collection of *vertex-edge* and *edge-triangle* pairs.

We initialize a set LS with $L_I(v)$. If $LS = \{v\}$, v is declared as a critical vertex. Otherwise the pair (v, e) is created by pairing v with the edge e in LS having $\min_{v \in e} I(v)$. In Fig. 3.3(b) edge $(6, 1)$ is selected to be paired with vertex 6.

Then, for each triangle t in LS, we compute the number of unpaired edges on the boundary of t, which are also in LS. We are interested in two cases:

- if t has no unpaired edges on its boundary, then it is classified as critical,

- if t has exactly one unpaired edge e on its boundary, then it is paired with e

If by cycling over all triangles in LS, no triangle is paired, at the end of the cycle a new edge is classified as critical, and we start again.

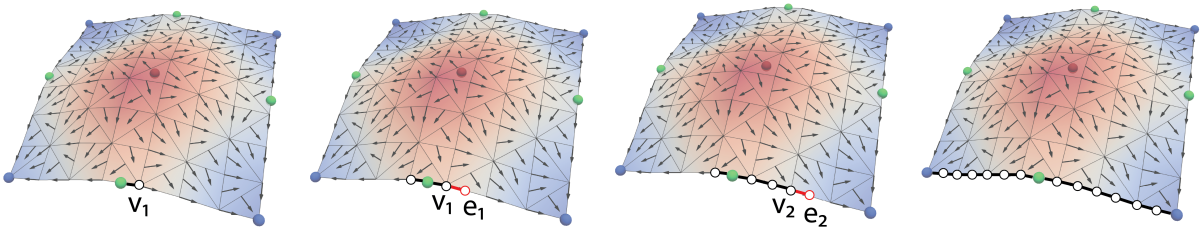
In Fig. 3.3(c), triangle $(6, 3, 1)$ is paired with its unique unpaired boundary edge $(6, 3)$. In Fig. 3.3(d), no triangle has either zero or exactly one unpaired edge, then edge $(6, 4)$ is declared as critical. In Fig. 3.3(e), triangle $(6, 5, 4)$ gets paired with its unique unpaired boundary edge $(6, 5)$. Fig. 3.3(f) shows the Forman gradient computed within the lower star of vertex 6.

On Terrain trees, *Step 3* is performed by considering each leaf block independently, and performing computation locally to the block on the lower stars of the vertices which belong to the block.

3.5.2 Extracting the critical net

Computing the critical net means visiting all the separatrix V_1-paths connecting critical vertices and edges, and all the separatrix V_2-paths connecting critical edges and triangles (see Section 2.1.2.2). The geometrical representation of the critical net is computed by connecting the barycenters of the triangles and edges visited in the separatrix V-paths. For the vertices of the triangle mesh, we consider the points themselves.

Extracting separatrix vertex-edge paths (V_1-paths). Given a critical edge, the two vertices in its boundary are first extracted. For each vertex, we extract its paired edge and we insert such edges into a stack *St*. The stack is used to implement a depth-first traversal of the path. At each iteration, we extract an edge e from *St* and we compute its boundary vertices. For each vertex v, we compute its paired edge e' and we add e' to *St* if $e' \neq e$. This retrieves the connections of



(a) (b) (c) (d)

Figure 3.4: Reconstruction of a portion of the critical net connecting a critical saddle with two critical minima. (a) Starting from a critical edge, the boundary vertices are connected with the edge. (b) The edge e_1 paired with v_1 is added to the stack. (c) The edge e_1 is extracted from the stack, connected with its paired vertex v_1. The other vertex on the boundary of e_1 (i.e., v_2) is retrieved and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until both paths reach critical vertices (minima).

critical edges with the critical vertices. The lines of the critical net reconstructed at this stage are obtained by connecting the barycenters of each edge with the boundary vertices.

Fig. 3.4 shows the extraction of the critical net limited to a critical edge and two critical vertices. Starting from the critical edge (Fig. 3.4(a)) the two vertices in its boundary are extracted and connected with the critical edge. Edges paired with such vertices are inserted in the stack *St*. In Fig. 3.4(b), edge e_1 is extracted from the stack and connected with its boundary vertex v_1. In Fig. 3.4(c), the other vertex on the boundary of e_1 (i.e., v_2) is connected to e_1 and its paired edge e_2 is added to the stack *St*. The depth-first traversal continues in the same manner until two critical vertices are encountered (Fig. 3.4(d)).

Extracting separatrix edge-triangle paths (V_2-paths). Given a critical edge, all the triangles in its co-boundary are extracted. For each such triangle, we extract its paired edge and we insert the edge into a stack *St*. Each time we extract an edge e from *St*, we compute the triangles in its coboundary. For each of such triangles t, we compute the edge e' paired with t and we enqueue e' if $e' \neq e$. The lines of the critical net reconstructed at this stage are obtained by connecting the barycenters of each triangle with the barycenters of the boundary edges encountered during the

visit.

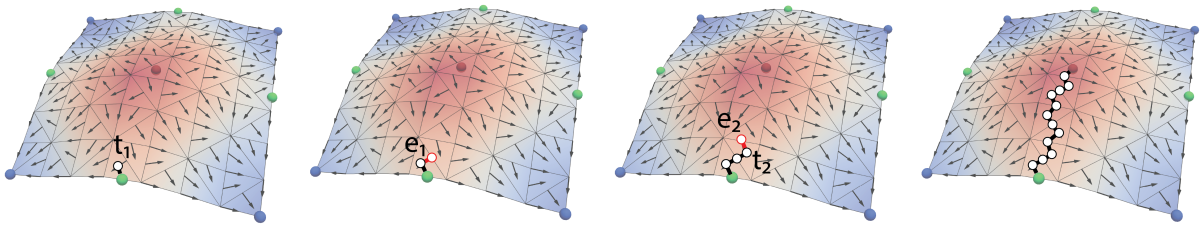
Fig. 3.5 shows the extraction of the critical net limited to a critical edge and a critical triangle. Starting from a critical edge (Fig. 3.5(a)) the triangle t_1 in its coboundary is retrieved. The edge e_1 paired with t_1 is inserted in the stack *St*. In Fig. 3.5(b), the edge e_1 is extracted from the stack and connected with the triangle paired with it (i.e., triangle t_1). In Fig. 3.5(c), the next triangle on the coboundary of e_1 (i.e., t_2) is connected to e_1 and its paired edge e_2 is added to the stack *St*. The depth first traversal continues in the same manner until a critical triangle is encountered (Fig. 3.4(d)).

As the extraction of the separatrix V-paths involves an intense mesh traversal, the leaf blocks of Terrain trees need to be visited multiple times. Thus, for efficiency, we use an auxiliary cache for encoding a subset of the connectivity relations required. Extracting separatrix V_1-paths, which are composed of edges and vertices, requires the *Edge-Vertex (EV)* relation. Extracting separatrix V_2-paths, which are composed of edges and triangles, requires the *Edge-Triangle (ET)* relation. The cache uses a Least-Recent-Used replacement policy (*LRU-cache*) which let us improve processing times with a negligible storage overhead.

Within each leaf block *b* of a Terrain tree, we execute the following steps:

1. expand the leaf block representation by computing and storing in the block the connectivity relations required, as discussed above;
2. extract the separatrix V-paths in *b*;
3. save in cache the connectivity relations of *b*.

During the computation of the separatrix V-paths, it can happen that a V-path will go



(a) (b) (c) (d)

Figure 3.5: Reconstruction of a portion of the critical net connecting a critical saddle with a critical maxima. (a) Starting from a critical edge, the coboundary triangle (i.e., t_1) is visited. (b) The traversal starts by adding the edge e_1, paired with t_1, to the stack. (c) The edge e_1 is extracted from the stack and connected with its paired triangle t_1. The other triangle in the coboundary of e_1 (i.e., t_2) is extracted and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until a critical triangle (maximum) is reached.

outside of the leaf block *b* currently processed. To deal with this situation, we introduced the notion of *dangling path*, where a dangling path is a V-path whose continuation is outside the block currently processed. Our strategy uses the dangling paths for postponing the construction of certain V-paths, thus limiting the number of times we have to enter and exit a leaf block.

The extraction of the gradient V-paths within a leaf block *b* is performed as follows:

- the new V-paths starting from the critical edges indexed by *b* are visited. Notice that an edge could be shared by two blocks. To process each critical edge once, we use the following convention: a critical edge is indexed by *b* if and only if *b* indexes the vertex with the higher label value in the vertex array Σ;

- the *dangling paths* that are entering *b* are then expended. Each time a dangling path is expanded, the corresponding entry in the auxiliary data structure storing dangling paths is removed. In this way, the storage requirement of this structure is kept negligible during the extraction process.

Notice that the visit of a V-path can be interrupted several times, as it can cross multiple

leaf blocks. Once the visit of the V-paths in b is terminated, the connectivity relations computed are saved into the cache.

3.6 Multifield Visualization

Multifield data are scientific data characterized by multiple field values. An example of this type of data is airborne LiDAR data where, for each point, multiple measures are recorded such as the intensity of the laser pulse, the point classification (i.e., ground, canopy, water, etc.), RGB bands, scan angle, and direction.

Extracting and visualizing descriptive information for multifield data is a major challenge. The technique implemented in this work relates to the notion of *Jacobi set* [57]. The Jacobi set of a collection of real-valued Morse functions defined on a common manifold is the set of all points where the function gradients are linearly dependent, which is directly related to the rank of the Jacobian matrix. This definition inspired numerical techniques aimed at rendering a single function built out of the multiple scalar fields in a way compatible with the relationships among scalar fields. In [135], a comparative measure is defined as measure for the evaluation of the local coherence among different scalar fields based on the gradient of the fields.

Given a point v, we can write the matrix of partial derivatives as follows:

$$dF(v) = \begin{bmatrix} \frac{\delta f_1}{\delta x_1}(v) & \cdots & \frac{\delta f_1}{\delta x_n}(v) \\ \vdots & \ddots & \vdots \\ \frac{\delta f_m}{\delta x_1}(v) & \cdots & \frac{\delta f_m}{\delta x_n}(v) \end{bmatrix}$$

The multifield comparison measure in [135] is defined as the norm of such matrix $\eta^F(v) =$

$||dF(v)||$. To speed up the computation, the estimation of $\eta^F(v)$ is reduced to the root of the maximum eigenvalue of the matrix $(dF(v))^T(dF(v))$. To compute $\eta^F(v)$, when v is a vertex of a TIN, we need to estimate partial derivatives at v. In [124] several methods have been analyzed, and the best method has been shown to be the *Average Gradient on Star (AGS)* method [131], which is both accurate and efficient. In AGS, the gradient at a vertex v is approximated by taking the average of the gradients estimated at the triangles incident in v.

In particular, given a scalar function f defined on the vertices of a TIN Σ, the gradient at a triangle t of Σ, denoted as ∇f_t, is calculated as follows:

$$\nabla_{f_t} = (f(v_j) - f(v_i)) \times \frac{(p_i - p_k)^{\perp}}{2A_t} + (f(v_k) - f(v_i)) \times \frac{(p_j - p_i)^{\perp}}{2A_t}$$

where $\perp$ denotes the 90 degrees rotation of a vector, A_t is the area of the triangle t, v_i, v_j, v_k are the three vertices of t, and p_i, p_j, p_k are vectors representing the x- and y- coordinates of v_i, v_j and v_k, respectively.

To compute the gradient at vertex v, we need to compute the so-called *mixed area* [131]. Let t be a triangle and p_i, p_j, p_k the vectors of coordinates of its three vertices. If t is non-obtuse, the contribution of t to the mixed area is $\frac{1}{8}(|p_ip_k|^2 cot\angle p_j + |p_iq_j|^2 cot\angle p_ip_k)$. If t is obtuse, there are two cases: if the angle at v is obtuse, the mixed area will be half of the triangle area, while if the angle at v is not obtuse, then the area will be a quarter of the triangle area. Then, the gradient at v is the weighted average of gradients computed at each triangle incident in v weighted by the corresponding mixed area.

Since the computation of the gradient at vertex v relies on the gradients at all the triangles intersecting at p, the computation of such gradient in the Terrain trees requires extracting the

Vertex-Triangle (VT) relation, as discussed in Section 3.3.1.

3.7 Experimental results

In this section, we study the performances of the Terrain trees family. TINs used are computed from LiDAR point clouds by means of a Delaunay triangulation algorithm from the *CGAL* library [24].

The characteristics of each TIN are reported in Table 4.1. We have used a total of seven TINs with a number of vertices ranging from 34 million to 193 million. GREAT SMOKEY MOUNTAIN, CANYON LAKE GORGE and BIG CREEK are three datasets provided by the OpenTopography repository [143]. For each of them we have computed a single TIN. The original SONOMA COUNTY dataset includes more than 60 billion points. We created four different datasets by subsampling the original point cloud at four different resolution levels. For each point cloud obtained we have computed a TIN that is used in our experiments.

Table 3.1: Overview of experimental datasets used in Part I. For each terrain, we list the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$.

	GREAT SMOKEY MOUNTAIN	CANYON LAKE	SONOMA COUNTY 1	SONOMA COUNTY 2	BIG CREEK	SONOMA COUNTY 3	SONOMA COUNTY 4
$\|\Sigma_V\|$	34M	49M	105M	135M	151M	154M	193M
$\|\Sigma_T\|$	68M	98M	210M	271M	303M	309M	386M

Our experimental evaluation addresses five aspects: (i) calibration of the capacities guiding the construction of Terrain trees, (ii) evaluation of the requirements for initializing Terrain trees, (iii) extraction of connectivity-based relations, (iv) computation of morphological and topology-based features, and (v) analysis and visualization of multifield data. The hardware configuration used for these experiments is a dual Intel Xeon E5-2630 v4 CPU at 2.20Ghz, and 64GB of RAM.

The source code of the Terrain trees library implementing the Terrain trees is available at [73]. The source code of the LibTri library implementing the IA data structure is available at [71].

3.7.1 Selection of capacities for Terrain trees generation

Terrain trees are generated based on, at most, two input values. One is the maximum number of vertices per leaf block, denoted as k_v. The second one is the maximum number of triangles per leaf block, denoted as k_t. Since the number of triangles in a TIN is about twice the number of its vertices, in the following, we set $k_t = 2k_v$. To efficiently calibrate such capacities, we performed a preliminary evaluation to identify non-optimal values, that create either too deep or too coarse hierarchies. This evaluation also established an initial test range such that each leaf block contains between 1 millionth and 10 millionth of vertices of the TIN. For each dataset, we create a total of 20 spatial indexes within this test range. All triangle meshes are generated over irregularly distributed LiDAR point clouds. Since the observed performance trend is similar on all datasets, in this section, we show just the plots from GREAT SMOKEY MOUNTAIN dataset to evaluate the experimental results on capacity selection. We provide the plots describing the performance trends of other datasets in Appendix A.2. In order to evaluate the effects of varying k_v and k_t, we analyze the following parameters: (i) storage costs, (ii) time requirements for generating a Terrain tree, and (iii) time requirements for answering the most common connectivity-based query, i.e., the extraction of *Vertex-Triangle* relation.

Since all Terrain trees encode the TIN with the same indexed representation, we focus only on the storage costs of the hierarchical index without considering the storage costs for encoding information on the vertices and on the triangle connectivity. Fig. 3.6 shows these results. The

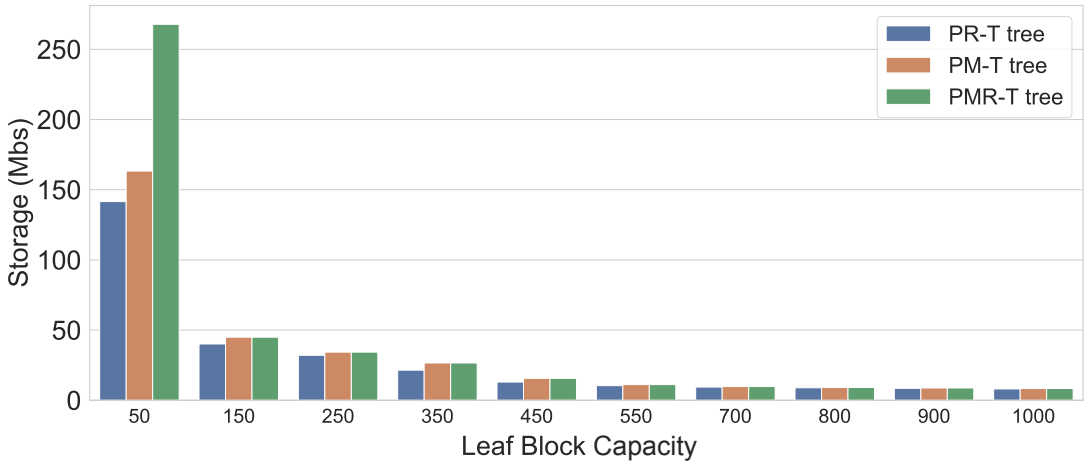
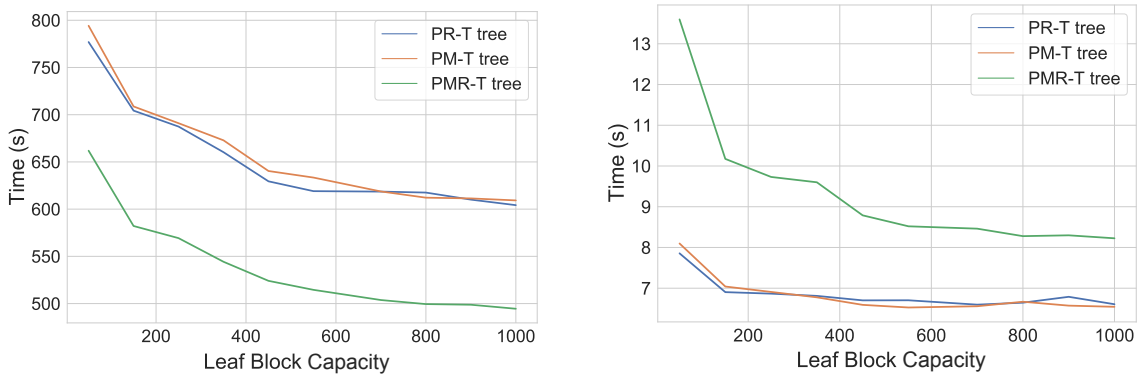


Figure 3.6: The storage costs for storing the hierarchical index of the Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.

storage costs show a sharp decrease with smaller capacities, while they remain nearly constant with larger ones. The differences in storage costs among the three indices are more noticeable when the capacities are small, while the costs become nearly identical for larger ones. The storage cost of the hierarchical index is closely related to the number of nodes in a Terrain tree. When using larger capacities (i.e., k_V greater than 550), Terrain trees have the same number of nodes, and this means that the spatial index is losing its effectiveness at decomposing the embedding space. This latter result highlights that the capacity on the triangles is the one guiding the spatial decomposition when both capacities become larger.

Figs. 3.7(a) shows the time required for generating the spatial decomposition of a Terrain tree on GREAT SMOKEY MOUNTAIN dataset. Generation times decrease for all types of Terrain trees when using larger capacities since each leaf block can contain more vertices and triangles, and the spatial decomposition is obtained by executing fewer split operations. Also, since the subdivision rule for the PMR-T tree is not recursive, its construction is always faster than the other two rules, that, on average, have similar generation times.



(a) Generation time

(b) VT extraction time

Figure 3.7: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

We evaluate now how different capacities affect the extraction of the *Vertex-Triangle (VT)* relation. This relation is key to most of the applications defined in our framework. Fig. 3.7(b) summarizes the results we have obtained. We notice that the larger the capacity used, the faster the extraction of the *VT* relation is. The larger time drop is noticeable on smaller capacities, while for larger capacities the time difference becomes smaller. Since PR-T and PM-T trees explicitly encode the range of vertices, they show similar performances, and are always faster than PMR-T trees, using from 30% to 70% less time, as the latter has to compute such vertex ranges at run-time.

These results highlight how the performances of Terrain trees are affected by the two user-defined capacities. By considering only the storage requirements and generation timings, we see that by using larger capacities, we reduce such quantities, since the decomposition is coarser and the compressed encoding becomes more effective. However, for larger values of k_V and k_T, the spatial index becomes less efficient, being more similar to a global representation. For such capacities, leaf blocks encode more entities, leading to a reduced speedup gain and to higher

storage requirements for encoding application-specific auxiliary data structures. Based on the results and evaluations above, we choose capacities that generate trees having a number of nodes between 300K and 500K, as we have noticed that such spatial indices are neither too coarse nor too deep, with overall good performances at generating and encoding Terrain trees and answering connectivity queries.

3.7.2 Terrain trees evaluation

Table 3.2: Overview of Terrain trees. For each Terrain tree, we list the thresholds k_V and k_T and the number of blocks in the index ($|N|$).

	GREAT S. MOUNT.			CANYON LAKE			SONOMA COUNTY 1			SONOMA COUNTY 2		
	k_V	k_T	$\|N\|$	k_V	k_T	$\|N\|$	k_V	k_T	$\|N\|$	k_V	k_T	$\|N\|$
PR-T	350	-	304K	450	-	295K	825	-	302K	950	-	476K
PM-T	350	700	377K	450	900	330K	825	1650	328K	950	1900	510K
PMR-T	-	700	377K	-	900	330K	-	1650	328K	-	1900	510K
	BIG CREEK			SONOMA COUNTY 3			SONOMA COUNTY 4					
	k_V	k_T	$\|N\|$	k_V	k_T	$\|N\|$	k_V	k_T	$\|N\|$			
PR-T	1000	-	401K	1100	-	445K	1300	-	474K			
PM-T	1000	2000	442K	1100	2200	482K	1300	2600	493K			
PMR-T	-	2000	442K	-	2200	482K	-	2600	493K			

In this section, we compare the storage costs and timings for generating the spatial indices of Terrain trees and IA data structure, as well as their performance at extracting the Vertex-Triangle (VT) relations.

We generate a PR-T tree, a PM-T tree, and a PMR-T tree for each TIN. A single value for k_v and k_t is selected for each dataset according to the results discussed in Section 3.7.1. Table 3.2

Table 3.3: Comparison of storage, expressed in megabytes (MB) and gigabytes (GB), for the underlying TIN, Terrain trees and IA data structure. O.O.M. stands for Out Of Memory.

	GREAT S. MOUNT.	CANYON LAKE	SONOMA COUNTY 1	SONOMA COUNTY 2	BIG CREEK	SONOMA COUNTY 3	SONOMA COUNTY 4
PR-T	21.5MB	20.7MB	21.4MB	33.7MB	28.4MB	31.4MB	33.4MB
PM-T	26.6MB	23.2MB	23.3MB	36.1MB	31.4MB	33.9MB	34.7MB
PMR-T	26.6MB	23.2MB	23.3MB	36.1MB	31.4MB	33.9MB	34.7MB
IA	908.0MB	1.31GB	2.80GB	3.62GB	4.04GB	4.12GB	O.O.M.
TIN	1.56GB	2.24GB	4.80GB	6.20GB	6.93GB	7.06GB	8.83GB

shows the capacities selected and the total number of internal and leaf nodes of each Terrain tree. Notice that PR-T trees and PMR-T trees use only one capacity value (i.e., k_v and k_t, respectively), while PM-T trees use both. As shown in Table 3.2, the number of nodes in a PM-T tree and a PMR-T tree of the same dataset is always similar, which leads to comparable storage costs. The number of nodes in a PR-T tree is always smaller than in the other two trees. These results match the ones of Section 3.7.1, which show that a PR-T tree always has a lower storage cost compared to the other two Terrain trees.

Table 3.3 shows the storage costs of the Terrain trees and of the IA data structure. Since both Terrain trees and IA data structure encode an indexed representation of the TIN, we represent this storage requirement separately, and thus, the storage costs shown in Table 3.3 consider only the requirements for encoding the spatial index in a Terrain tree, and for encoding adjacency and coboundary relations (i.e., *Triangle-Triangle* and partial *Vertex-Triangle* relation) in the IA data structure. The total storage requirements can be easily computed by adding the storage cost of the TIN to the corresponding Terrain trees or IA data structure overhead cost. Note that the overhead cost of Terrain Trees is between 1% and 3% of the overhead cost of the IA data structure. When considering also the cost of the underlying indexed TIN representation, a Terrain tree can encode

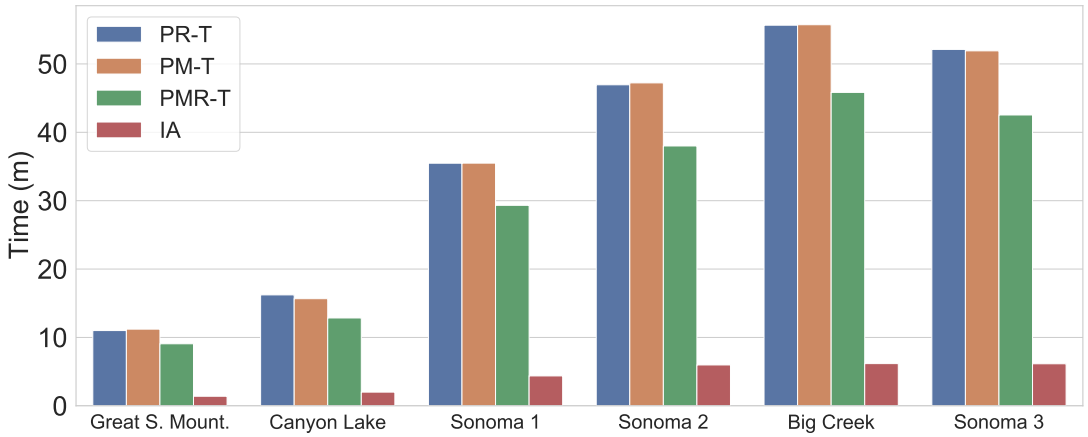


Figure 3.8: Comparison of total timings, expressed in minutes (m), for generating a Terrain tree or the IA data structure.

the same dataset using, on average, 36% less storage than IA data structure. The differences between the three Terrain trees are minimal. The PR-T tree is the most compact since it generates fewer leaf blocks compared to the other two.

Fig. 3.8 shows the time requirements for generating a Terrain tree or the IA data structure. The generation times do not consider the time needed to load the TIN from file, but only the time for generating the corresponding data structure. The generation times for the IA data structure are about 10% of those of Terrain Trees. This is expected since the IA computes only the adjacencies between triangles and a partial *VT* relation. Terrain Trees, instead, are created by first computing the spatial decomposition and then compressing its representation, as described in Section 3.3. Also in this case, the differences between the three Terrain trees are minimal, and the generation of a PMR-T tree is always 20% faster than that of the other two, which is consistent with the findings in Section 3.7.1.

As shown in Fig. 3.9, Terrain trees can always extract *VT* relations faster than the IA data structure. PR-T and PM-T trees use from 57% to 72% less time than the IA data structure. PMR-T trees use, on average, from 30% to 70% more time compared to the PR-T and PM-T trees, still

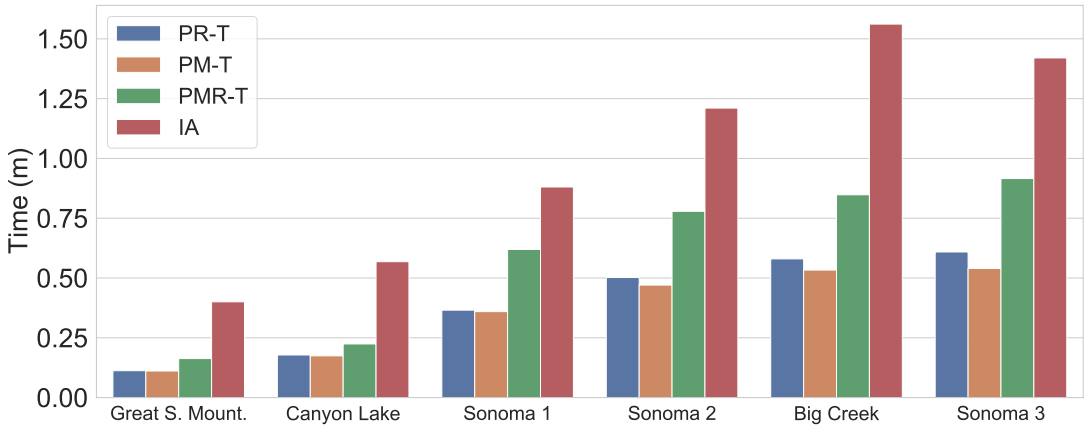


Figure 3.9: Comparison of total timings, expressed in minutes (m), for extracting the VT relations.

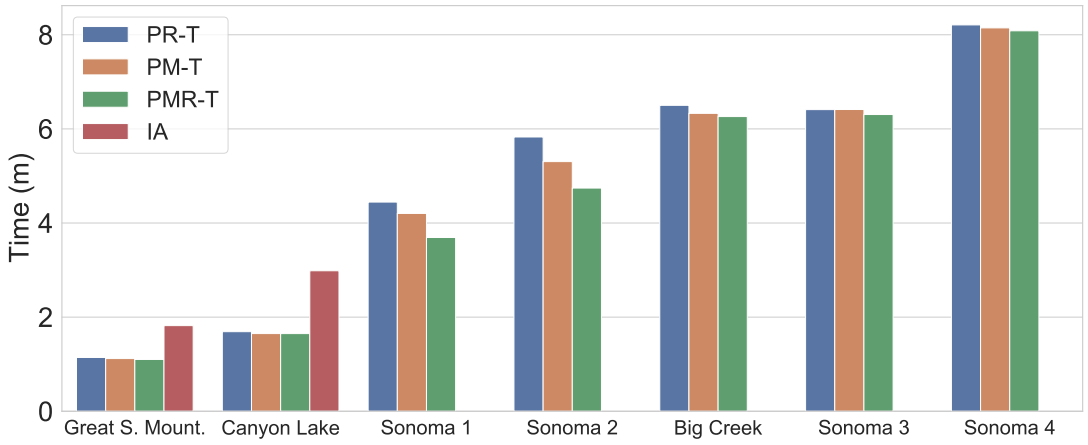


Figure 3.10: Comparison of total timings, expressed in minutes (m), for extracting edge slopes.

saving at least 30% time compared to IA data structure. The differences among Terrain Trees are due to the encoding of the vertex ranges. In PR-T and PM-T trees, such ranges are explicitly encoded, while in PMR-T trees, they are computed at run-time on a block-by-block basis.

3.7.3 Computing morphological features

In this section, we evaluate the performances for computing morphological features, as described in Section 3.4. Results are shown in Figs. 3.10, 3.11, and 3.12.

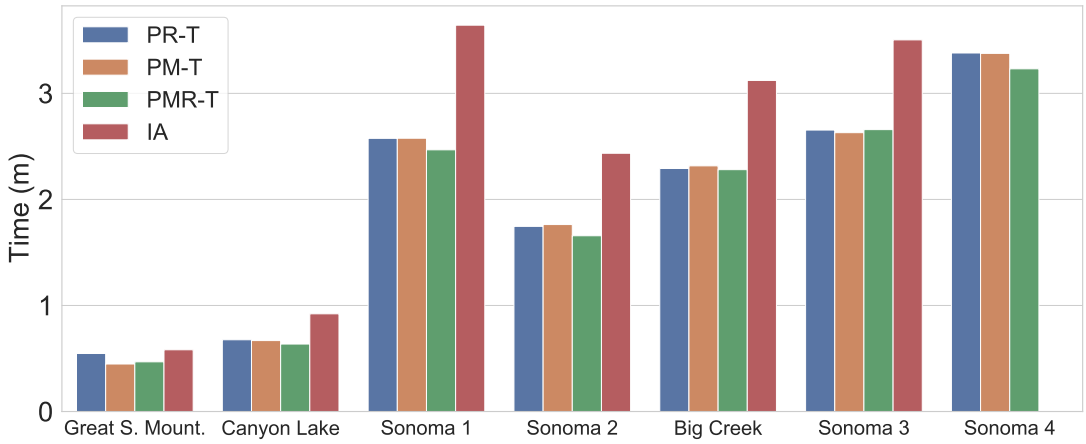


Figure 3.11: Comparison of total timings, expressed in minutes (m), for extracting concentrated curvatures.

Computing the *triangle slope* requires the *Triangle-Vertex* relation, while computing the *edge slope* requires the (dual) *Vertex-Triangle* relation. Since both Terrain trees and the IA data structure store the *Triangle-Vertex* relation explicitly we only compare their performances in computing the edge slope.

As edges are not explicitly encoded neither in Terrain trees nor in the IA data structure, we have to use an auxiliary lookup table in both implementations for encoding the slope values without duplicates. Terrain trees enable the usage of a local data structure within each leaf block, and this reduces both the cost of encoding and accessing the lookup table. As shown in Fig. 3.10, the slope estimation benefits by the use of a spatial index and a modular structure. We notice that computing edge slopes on Terrain trees requires from 37% to 45% less time and less memory than by using the IA data structure (considering also that the IA data structure goes out of memory five times). The difference among the three Terrain Trees is limited, since the extraction time for the *VT* relation accounts for a small portion of the overall slope computation time.

As discussed in Section 3.4, estimating the *curvature* requires visiting the star of each vertex (i.e., extract the *Vertex-Triangle* relation). As in Fig. 3.11, the implementation based on

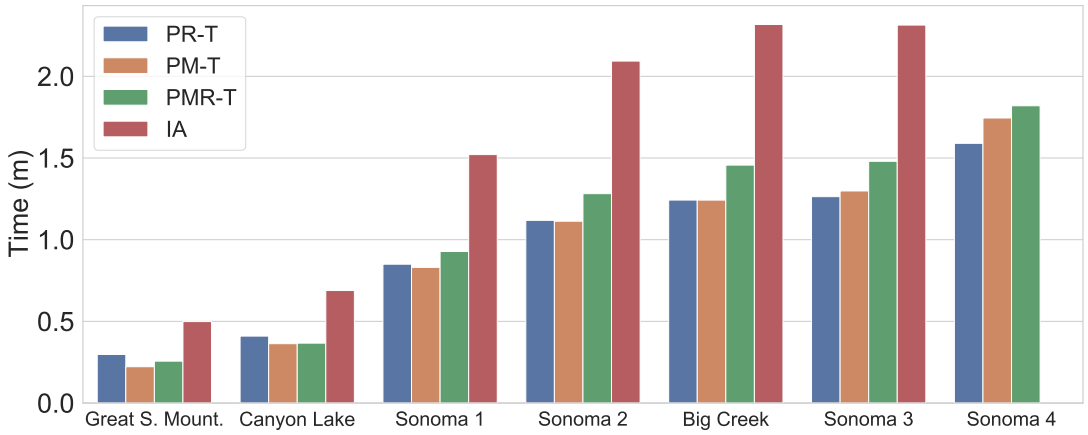


Figure 3.12: Comparison of total timings, expressed in minutes (m), for extracting roughness values.

Terrain trees is always faster than the one based on the IA data structure, as it requires from 25% to 30% less time. Both Terrain trees and the IA data structure require the same amount of space for encoding curvature values, while the size of the auxiliary data structures is negligible. The performances of the three Terrain trees are similar. This shows that extracting the vertices in a block at run-time does not affect significantly the performances of the spatial index.

Estimating *roughness* requires computing the *Vertex-Vertex* relation. In Terrain trees, the roughness computation is pretty efficient (see Fig. 3.12), being from 36% to 55% faster than the corresponding procedure on the IA data structure. On larger datasets, PMR-T trees are always slower than PR-T and PM-T trees, since they are less efficient at extracting the *VV* relation.

3.7.4 Computing topology-based segmentations

In this section, we evaluate the performances in computing the Forman gradient, and in extracting the critical net.

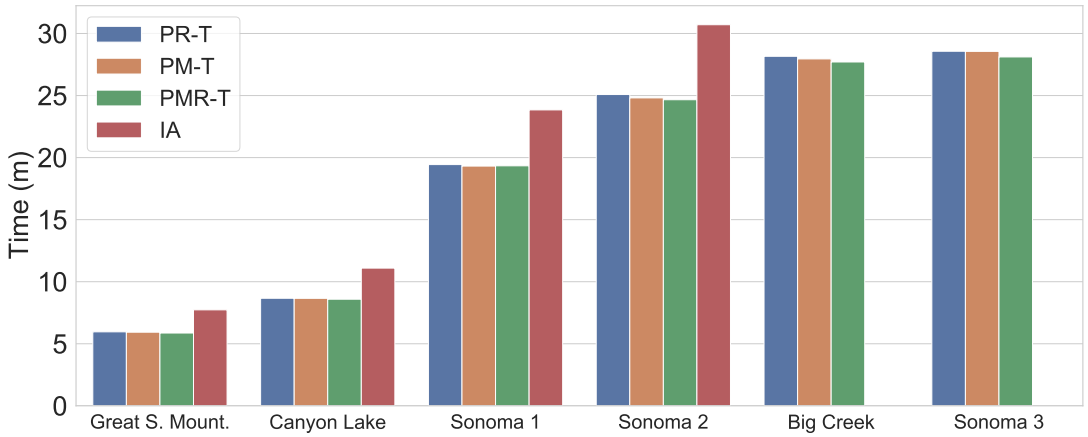


Figure 3.13: Comparison of total timings, expressed in minutes (m), for computing the Forman gradient vector.

Forman gradient computation

As discussed in Section 3.5, computing the Forman gradient on a Terrain tree requires extracting the star of each vertex of the TIN, i.e., the *Vertex-Triangle* relation. As shown in Fig. 3.13, Terrain trees are always faster than the IA data structure requiring about 20% less time. PMR-T trees are usually slightly faster than PR-T and PM-T trees, but overall the time difference between the three Terrain trees is small.

Since Terrain trees have lower storage requirements than the IA data structure, they can complete the Forman gradient computation on all datasets, i.e., there is enough memory for encoding the Forman gradient and the auxiliary data structures used in the process. This does not apply to the IA data structure, which goes out of memory on the two larger datasets.

Critical net extraction

In a Terrain tree, the extraction of the critical net requires an intense navigation of the spatial index, and the extraction at run-time of connectivity relations that, in the IA data structure,

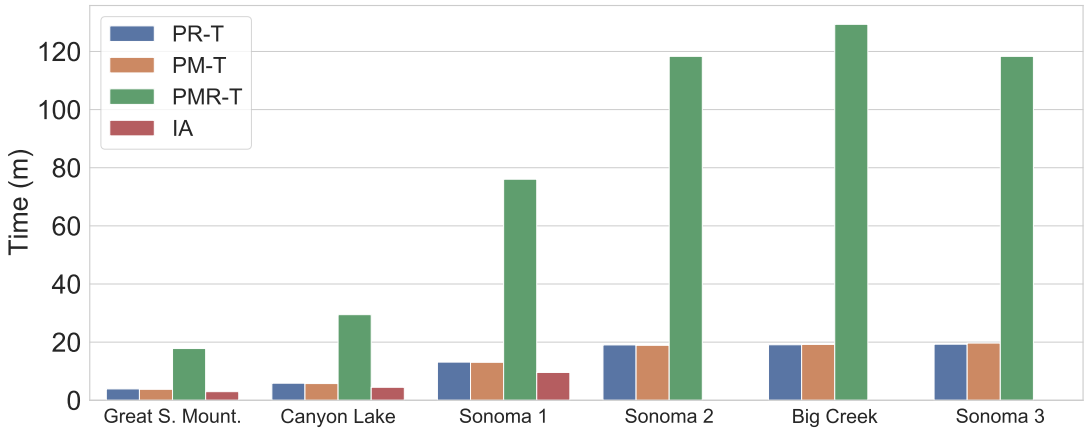


Figure 3.14: Comparison of total timings, expressed in minutes (m), for computing the critical net.

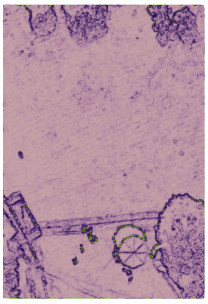
are either explicitly encoded (*Triangle-Triangle* relation) or efficiently extracted (*Vertex-Vertex* relation). This application represents an interesting worst-case scenario for Terrain trees.

Both Terrain trees and the IA data structure use auxiliary data structures for extracting the critical net. The IA data structure uses a global stack to perform the TIN traversal. Conversely, a Terrain tree uses a cache of leaf blocks with expanded connectivity information as well as a list of dangling paths, plus a local stack within each leaf block (see Section 3.5.2). As shown in Fig. 3.14, thanks to the lower storage requirements of Terrain trees, such auxiliary structures can be effectively encoded in memory, while the IA data structure goes out of memory on the three larger datasets.

Comparing timings, PR-T and PM-T trees have comparable performances with respect to the IA data structure, and use up to 10% more time, while PMR-T trees perform significantly worse being up to 8 times slower than the IA data structure. Comparing Terrain trees, PR-T and PM-T trees have similar performances and are at least 5 times faster than PMR-T trees. As we have already observed with the other applications, this speedup is due since a PMR-T tree has to compute the range of vertices at run-time.



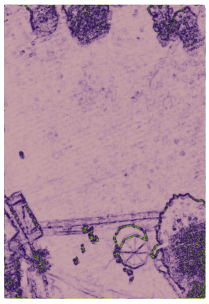
(a) Satellite



(b) RGB



(c) Elevation+Green



(d) Elevation+RGB

Figure 3.15: The satellite map and the visualization of the multifield comparison measure on an area with few trees and some human artifacts. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), based on elevation paired with green values (c), and based on elevation paired with RGB values (d).

3.7.5 Multifield data visualization

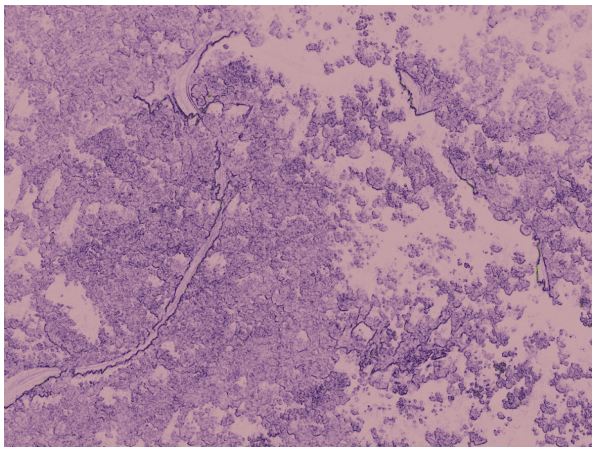
We evaluate the performances of Terrain Trees for computing the multifield measure described in Section 3.6. We test our implementation for a visual analysis on two small datasets (shown in Figs. 3.15 and 3.16), and for a performance analysis against the IA data structure on three different areas of the SONOMA COUNTY dataset. Each dataset has a total of five scalar fields: elevation, a color field (encoded as a RGB triple), and roughness.

First, we visually analyze the algorithm performance on an area with few trees and some human artifacts (a fence and a small building). Fig. 3.15 presents the raster image of this area (Fig. 3.15(a)), and three output images obtained with our algorithm in which we used as input scalar fields: (i) the RGB values (Fig. 3.15(b)), (ii) the green band paired with the elevation (Fig.

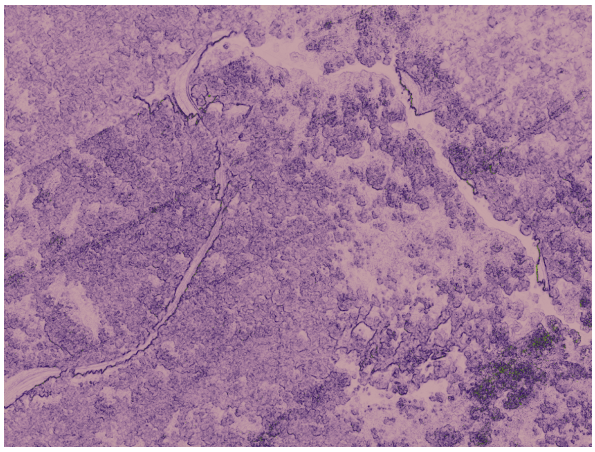
3.15(c)), and (iii) the RGB values paired with the elevation (Fig. 3.15(d)). Just using the RGB values, the algorithm can identify the boundary of the buildings and their shadows clearly, but it does not highlight precisely the trees. Pairing the green band with the elevation improves the identification of trees, while the human artifacts result smoothed and less clear. Lastly, if we pair the three RGB values and the elevation, the algorithm can correctly highlight both trees and human artifacts.



(a) Satellite



(b) RGB



(c) Roughness+RGB

Figure 3.16: The satellite map and the visualization of the multifield comparison measure on an area with high tree density. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), and based on roughness paired with RGB values (c).

In order to understand the performance of our strategy in distinguishing forest areas from other land cover types (like rivers or streets), a region with higher tree density has been used (see Fig. 3.16). We compare the visualization results when using just the RGB values (Fig. 3.16(b)), and when pairing them with roughness (Fig. 3.16(c)). In this case, we pair the RGB values with roughness instead of the bare elevation, since roughness has been proven to be a better estimator for identifying surface deformations in a terrain [165]. The visual comparison of the outputs shows that the multifield strategy is more precise in highlighting the different cover types, when also a geometric field is added to the identification procedure. Pairing roughness with the color values enables the identification of both the road crossing the forest and areas of low vegetation (Fig. 3.16(c)), that cannot be clearly identified by just using the color (Fig. 3.16(b)). Also, a narrow band representing the road in the forest can be identified clearly only if we include roughness values in our input fields.

The visual analysis of the results shows that the multifield strategy can be effectively used for highlighting key areas in satellite datasets, and that the identification improves when pairing a geometric attribute with other scalar fields that are not spatial or geometric.

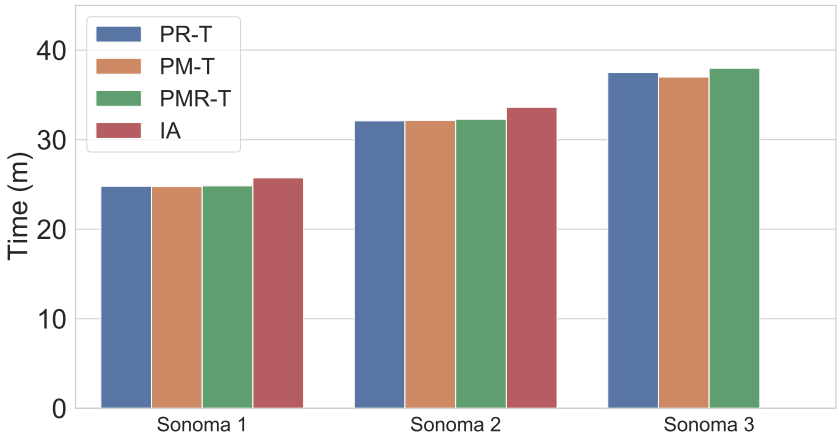


Figure 3.17: Comparison of total timings, expressed in minutes (m), for computing the multifield measure.

Finally, we compare the performance of Terrain trees and IA data structure at extracting the multifield measure on three datasets based on SONOMA COUNTY. Results are reported in Fig. 3.17. Overall, the timing performances of the Terrain trees and IA data structure are very similar, even if we notice that Terrain trees are about 5% faster than the IA data structure. The performance difference is minimal if we compare the Terrain trees. Thanks to the initial lower storage requirements, Terrain trees can compute the multifield measure on all three datasets, while the IA data structure goes out of memory on the largest one.

3.8 Concluding Remarks

We have presented a family of spatial data structures, the *Terrain trees*, for the efficient representation, analysis and visualization of triangulated terrains. A Terrain tree combines a minimal connectivity-based encoding of the underlying triangle mesh with a hierarchical spatial index, thus implicitly encoding other connectivity relations. Terrain trees consist of three spatial indexes that use different bucketed subdivision rules. By borrowing an idea presented in [74] for a distributed data structure for simplicial complexes in arbitrary dimensions, we use spatial coherence to reorder the indexed data, thus achieving the compression of both vertex and triangle information inside the spatial index. This enables high storage reduction and optimized algorithms.

We have proven the effectiveness of our proposal by designing and implementing state-of-the-art morphological estimators for terrain analysis, like slope, curvature, and roughness, as well as a distributed technique based on discrete Morse theory for topology-based segmentation of such terrains. Lastly, as it is common to study a terrain in combination with additional fields

attached to it, we have defined a distributed strategy for visualizing multifield data.

We have experimentally demonstrated how the bucketing capacities on the Terrain trees affect generation times, storage requirements, and performances in extracting a basic connectivity relation. This has enabled the optimal identification of the most appropriate capacity ranges. We have then experimentally demonstrated the efficiency of our data structure by comparing it against the most common and compact connectivity-based data structure, the IA data structure. Terrain trees require always less storage, they generally perform better than the IA data structure, and their effectiveness increases with the dataset size. Conversely, the difference in performances among the three Terrain trees is minimal. We have noticed, however, that spatial indexes explicitly encoding the vertices in each leaf block have shown better performances at computing those estimators which require the efficient extraction of the triangles incident in a vertex. The source code of Terrain trees library and of the library implementing the IA data structure, called *LibTri*, are available in the public domain [71, 73].

Chapter 4: Topology-aware TIN simplification on Terrain trees

4.1 Introduction

This chapter introduces methods for efficient topology-aware TIN simplification on Terrain trees. As discussed in Section 1.1, most mesh simplification methods do not consider the topology of underlying mesh and existing topology-aware mesh simplification methods are not efficient when processing large terrains. Thus, we propose a new topology-aware mesh simplification on Terrain trees that uses a batched processing strategy to reduce both the memory and time requirements of the simplification process. The simplification method employs the topology-aware edge contraction operator defined by Iuricich and De Floriani [99], which allows for constructing the HFT multi-resolution model. Moreover, thanks to the spatial decomposition on the basis of Terrain trees, it can be easily parallelized to a multi-core environment. In Section 2.2.2, we reviewed parallel mesh simplification methods in previous studies and categorized them into two classes. Our parallel simplification method introduced in this study extends the second strategy by considering the conflicts between different blocks in a hierarchical decomposition of the domain and processing independent blocks at the same time.

The major contributions of this study include:

1. the design and implementation of a topology-aware mesh simplification method on a com-

pact data representation for triangle meshes, Terrain trees;

2. the definition of a leaf-locking strategy on Terrain trees and the design of a parallel topology-aware mesh simplification algorithm;

3. the design of an algorithm for updating a Terrain tree after simplification, which can be applied independently of the way the mesh encoded in the Terrain tree is modified;

4. an experimental evaluation on the new simplification methods in terms of computing performances and quality of the output mesh;

5. a discussion on leaf capacity selection for efficient mesh simplification on Terrain trees;

6. an optimized simplification strategy on Terrain trees for improving output mesh quality.

While the proposed topology-aware mesh simplification method ensures that all critical simplices are preserved during the process, in some applications, it may be ideal to remove noise or insignificant terrain changes. In such cases, a preprocessing step to reduce the complexity of terrain topology is necessary. Therefore, in Section 4.2, we review a topological simplification algorithm [99] that can be used for this purpose.

A preliminary research result on the simplification algorithm and the evaluation of the computational performance was presented at ACM SIGSPATIAL in 2021 [171]. Later, the comprehensive study was published on ACM Transactions on Spatial Algorithms and Systems in 2024 [172]. An open-source library for topological simplification and topology-aware mesh simplification on Terrain trees has been developed [170].

The remainder of this chapter is organized as follows. Section 4.3 provides the definition of a topology-aware edge contraction operator. Section 4.4 describes the proposed topology-

aware simplification algorithm, and, Section 4.5 defines a new parallel simplification algorithm relying on the distributed nature of Terrain trees. Section 4.6 describes an algorithm for updating a Terrain tree as a post-processing step, which is necessary to keep the downstream operations efficient. Section 4.7 discusses several mesh quality metrics for evaluating the output meshes generated by the proposed simplification algorithms. In Section 4.8, we experimentally evaluate both the sequential and the parallel topology-aware simplification on Terrain trees and compare them against an implementation of the topology-aware simplification on the IA data structure. Finally, in Section 4.9, we draw some concluding remarks.

4.2 Topological cancellation

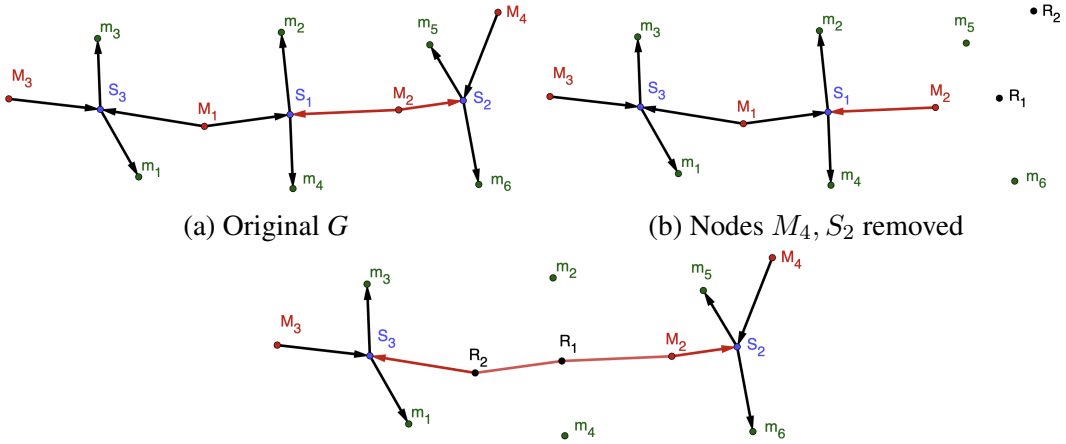
As introduced in Section 2.2.3, noisy data can affect the analysis of a terrain, which creates the need for simplifying the topology of the terrain while maintaining important critical simplices. In this section, we introduce a discrete-Morse-theory-based operator called *topological cancellation* [76] for simplifying the topology of the terrain. In the following, we use a surface network G to represent the critical simplices and their connectivity on a terrain surface Θ. For a surface network G, there are two cancellation operators, *maximum-saddle-maximum* and *minimum-saddle-minimum*.

Given two maxima M_a, M_b, and a saddle S adjacent to both maxima in a surface network G, the maximum-saddle-maximum cancellation removes M_a, S, and the arc connecting them. Other saddles adjacent to M_a, if exist, are reconnected to M_b after the removal of arc $M_a \rightarrow S$. If such saddles exist, then we call M_a an *internal* maximum. Otherwise, we call M_a a *boundary* maximum. For instance, M_4 in Fig. 4.1(a) is a boundary maximum, and thus, no other saddle is reconnected

to M_1 after M_4 and S_2 being removed (as shown in Fig. 4.1(b)). The arcs connected to removed saddle and removed maximum in G are removed at the same time. Conversely, when an internal maximum like M_1 in Fig. 4.1(a) and its adjacent saddle S_1, are removed, arcs connected to S_1 are removed and all other saddles adjacent to M_1 (i.e., S_3) are reconnected to the other maxima M_2 (as shown in Fig. 4.1(c)). After the cancellation, the geometry of arc $M_2 \rightarrow S_3$ becomes a chain merged from the geometries of three arcs in the original graph ($M_2 \rightarrow S_1$, $M_1 \rightarrow S_1$, and $M_1 \rightarrow S_3$). The change in the surface network made by the maximum-saddle-maximum cancellation corresponds to the changes in the underlying Forman gradient field of Θ. All arrows along the separatrix V-path corresponding to the removed arc $M_a \rightarrow S$ are reversed, and thus, the separatrix V-path no long exists.

The minimum-saddle-minimum cancellation, *boundary minimum*, and *internal minimum* are similarly defined. Both cancellation operators operate only on the Forman gradient field and surface network G, and the underlying mesh is not modified.

A common criterion to decide if an arc should be removed or not is the elevation difference between the two nodes connected by the arc [81]. The value of absolute elevation difference between a pair of adjacent nodes in G is also called the *persistence* of this pair of nodes, denoted as ω. When the persistence of a pair of adjacent nodes is small enough, we remove the two nodes from G since they are not significant surface features. Therefore, the topological simplification of the surface network G consists of two steps: (1) sorting all arcs in G based on the persistence ω of the two nodes connected by each arc and (2) removing pairs of nodes with ω lower than a threshold ω_m through topological cancellation. This process helps to maintain only the essential critical simplices, which can avoid over-segmentation and allows more edges to be contracted during a topology-aware mesh simplification.



(a) Original G (b) Nodes M_4, S_2 removed

(c) Nodes M_1, S_1 removed

Figure 4.1: Example of a maximum-saddle-maximum cancellation operation on a surface network G. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines highlight arcs starting from maximum M_2. Note that the regular points are not a part of the surface network and are just displayed for illustration purpose. (a) Original surface network; (b) surface network when a boundary maximum M_4 and a saddle S_2 adjacent to it are removed; (c) surface network when an internal maximum M_1 and a saddle S_1 adjacent to it are removed.

4.3 Topology-aware edge contraction

Edge contraction is a widely used operator for triangle mesh simplification [95]. Given an edge $e = \{v_1, v_2\}$ in a triangle mesh Σ, e is contracted to one of its endpoints, and edge e, one of its endpoints, and the triangles incident in e are removed from Σ. When operating on the underlying mesh of a TIN, an edge contraction operator can modify the shape of the TIN, potentially creating non-valid meshes. To prevent this, we verify that the edge contraction satisfies two fundamental validity conditions, namely the link and fold conditions.

The *link condition* [49] ensures that the simplified mesh has the same homological properties as the original one. Recall that the *link* $Lk(v)$ of a vertex v consists of all vertices adjacent to

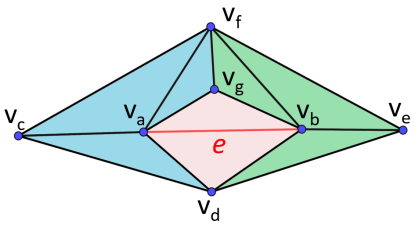


Figure 4.2: Example of edge contraction not satisfying the link condition (same figure as 2.1 in Section 2.1.1). Given edge $e = \{v_a, v_b\}$, we have $Lk(v_a) \bigcap Lk(v_b) = \{v_d, v_g, v_f, \{v_f, v_g\}\}$ and $Lk(e) = \{v_g, v_d\}$. The condition is not satisfied since $Lk(v_a) \cap Lk(v_b)$ contains vertex v_f and edge $\{v_f, v_g\}$ that are not in $Lk(e)$.

v in the mesh and of all the edges opposite to v bounding the triangles incident in v. Similarly, the link $Lk(e)$ of an edge e consists of the two vertices of the triangles incident in e that are not endpoints of e. An edge $e = \{v_1, v_2\} \in \Sigma$ is said to satisfy the *link condition* if and only if $Lk(v_1) \cap Lk(v_2) \subseteq Lk(e)$. Fig. 4.2 shows an example of an edge contraction that violates the link condition. If edge *e* is contracted to v_a, the resulting mesh becomes invalid since more than two triangles would be incident in the same edge (i.e., edge $\{v_a, v_f\}$).

The *fold condition* [25] ensures that for every edge e' in $Lk(v_2)$, v_1 and v_2 lie on the same side of the line l passing through e'. If this condition is not verified, then Σ will have at least one triangle folding over itself after contracting e to v_1.

Our purpose is to preserve the topological features of the scalar field defined on Σ while simplifying the underlying mesh. This translates into maintaining the Forman gradient and, thus, the critical simplices and their connectivity. To this aim, we apply the *gradient-aware condition* [99]. Given a mesh Σ endowed with a Forman gradient V, an edge $e = \{v_1, v_2\}$ can be contracted to vertex v_1 if and only if: (1) all simplices to be removed (v_2, e, and two triangles incident in e) are not critical; and (2) either v_1 or v_2 is paired with *e* in *V*. For instance, as shown in Fig. 4.3(a), it is fine to contract edge $e = \{v_1, v_2\}$ since none of *e*, t_1, t_2, or v_2 is critical, and *e* is paired with v_1.

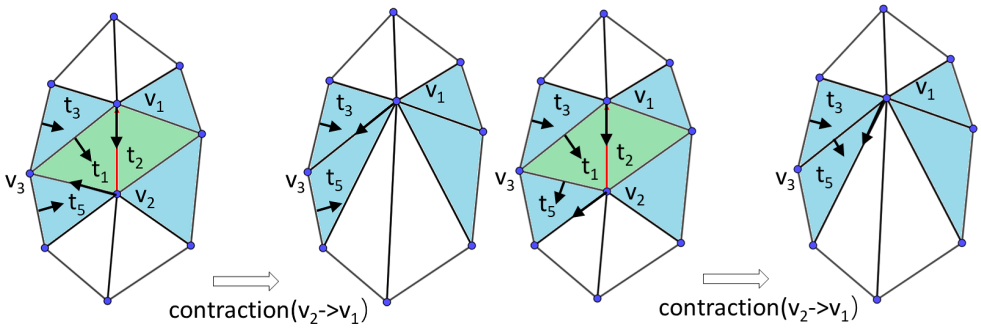


Figure 4.3: Two possible gradient configurations and corresponding updates after contracting edge $\{v_1, v_2\}$ (in red) to v_1. Black arrows represent gradient pairs and red arrows represent contraction direction.

A topology-aware edge contraction requires, in addition to the modification of the mesh, also the update of the Forman gradient V. When contracting edge e, two triangles t_1 and t_2 adjacent to e are removed. The updates of V involve at most four triangles, which share an edge different from e with either t_1 or t_2.

Since the updates are symmetric with respect to e, we only discuss the updates on the left part of e, where edge e is considered as oriented from v_2 to v_1. We denote the other two triangles adjacent to t_1 as t_3 and t_4 and the vertex opposite to e in t_1 as v_3. The updates on the left part need to ensure that vertices v_1, v_3, edge $\{v_1, v_3\}$, and triangles t_3 and t_4 are still paired after simplification. We know that edge e is paired with either v_1 or v_2. Thus, if e is paired with v_2, after the contraction, the pairing of v_1 does not change; otherwise, v_1 will be paired with the simplex paired originally to v_2.

Now we consider edges $\{v_1, v_3\}$ and $\{v_2, v_3\}$. Before the contraction, t_1 should have been paired with either $\{v_1, v_3\}$ or $\{v_2, v_3\}$, since edge e was paired and t_1 was not a critical triangle. If t_1 was paired with $\{v_1, v_3\}$, then $\{v_2, v_3\}$ should have been paired either with one of its endpoints (see Fig. 4.3(a)), or with another triangle, t_4 (see Fig. 4.3(b)). In both cases, t_3 and t_4 are

paired with the same simplices after the contraction. After the removal of t_1 because of the edge contraction, edge $\{v_1, v_3\}$ is paired with the simplex originally paired with edge $\{v_2, v_3\}$, i.e., either with one of its endpoints (see Fig. 4.3(a)), or with another triangle t_4 (see Fig. 4.3(b)). The same reasoning applies when t_1 was paired with $\{v_2, v_3\}$. The same update strategy is applied to the simplices on the right of the edge e to maintain the topology of the Forman gradient [99].

4.4 Topology-aware simplification on Terrain trees

In this section, we present a new topology-aware simplification algorithm developed on a Terrain tree T to simplify a triangle mesh Σ. In our experiments, the PR Terrain tree has been shown to be slightly more compact and efficient than the *PM* and *PMR Terrain trees*. Therefore, in this work, we use the PR Terrain tree to encode meshes, and we refer to it as the Terrain tree in the rest of the chapter, for the sake of simplicity.

To define the terrain topology, this algorithm uses a Forman gradient V computed on Σ inside the Terrain tree, which is encoded as a bit vector using the same indexing of Σ_T, resulting in a storage cost of one byte per triangle [191]. As an error metric for edge contraction, we use the *Quadric Error Metric* (QEM) [83]. Appendix A.3 provides a comprehensive explanation of the process for computing the initial error quadrics associated with each vertex v, representing a set of planes incident in v.

To simplify Σ, all leaf blocks in T are visited through a depth-first traversal. Algorithm 1 provides a pseudo-code description of the simplification procedure within a leaf block *b* in T. The cost of each edge e, which is the error introduced if e is contracted, is computed from the initial error quadrics of its endpoints. In our implementation, edge $e = \{v_1, v_2\}$ is contracted to

either v_1 or v_2 depending on which vertex leads to the smallest cost for edge e. We consider e as a *candidate edge* for leaf block *b* only if the vertex to be removed is contained in *b*, and the cost of e is lower than a user-defined threshold ω. Edge e is an *internal edge* for *b* if also the other vertex of e is in *b*, otherwise e is a *cross edge*.

For each leaf block *b*, the algorithm performs the following steps:

1. *Extract the Vertex-Triangle (VT) relations for the vertices in* b (row 1): the *Vertex-Triangle (VT) relation* for a vertex v in *b* is defined as the set of triangles incident in v.

2. *Build a priority queue* Q *of candidate edges* (row 3): the edges in the queue are sorted in ascending order based on their costs.

3. *Simplify candidate edges* (rows 4-21): for each candidate edge e, the three validity conditions discussed before are checked. If these conditions are satisfied, edge e is contracted, and the Forman gradient updated together with the Terrain tree. This step is described in details below.

The link, fold, and gradient-aware conditions are checked for each edge $e = \{v_1, v_2\}$ extracted from *Q*. These checks require the VT relations for v_1 and v_2 and the *Edge-Triangle (ET) relation* of e (rows 9-11). $ET(e)$ consists of the two triangles sharing edge e. If e is an internal edge, function GET_VT directly retrieves $VT(\nu_1)$ and $VT(\nu_2)$ from array *local_vts*. Conversely, if e is a cross edge, and v_1 is contained by another leaf block b_1, GET_VT must extract the VT relations of b_1. To optimize this latter step, we use an auxiliary Least Recent Used (LRU) cache LRU for encoding a subset of the extracted VT relations. When v_1 is in block b_1, GET_VT looks first if the VT relations of block b_1 are in LRU. If such relations are not in LRU,

Algorithm 1 LEAF_SIMPLIFICATION(b, Σ, F, E, ω, LRU, b_R)

Input:

b: current leaf block
Σ: the triangle mesh
F: the Forman gradient on Σ
E: the array of vertex error quadrics
ω: the edge cost threshold
LRU: LRU cache
b_R: root block of the hierarchy

// Extract the local VT relations for the vertices in b
1: $local\_vts \leftarrow$ LOCAL_VT(b, Σ_T)
// Create an array for encoding the updated edges costs
2: *updated_edges* $\leftarrow$ [ ]
// Create a priority queue of candidate edges
3: $Q \leftarrow$ CANDIDATE_EDGES(b, Σ_T, E, ω)
4: **while** $Q \neq \emptyset$ **do**
5: $e \leftarrow$ DEQUEUE(Q) // $e = \{\nu_1, \nu_2\}$
// Check if e has been updated and if its cost is updated
6: **if** $e \in$ *updated_edges* **and not** SAME_COST(e, *updated_edges*) **then**
7: skip e // If its cost is not updated, then skip this edge
8: **end if**
9: $VT(\nu_1) \leftarrow$ GET_VT(ν_1, $local\_vts$, LRU, b_R, Σ)
10: $VT(\nu_2) \leftarrow$ GET_VT(ν_2, $local\_vts$, LRU, b_R, Σ)
11: $ET(e) \leftarrow$ GET_ET(e, $VT(\nu_1)$)
// Check three conditions introduced in Section 4.3 for e
12: **if** LINK_CONDITION(e, $VT(\nu_1)$, $VT(\nu_2)$, $ET(e)$)
13: **and** FOLD_CONDITION(e, $VT(\nu_2)$, $ET(e)$)
14: **and** GRADIENT_CONDITION(e, $VT(\nu_2)$, $ET(e)$, F) **then**
15: CONTRACT(e, $VT(\nu_2)$, $ET(e)$, E, Σ)
16: UPDATE_GRADIENT(e, $VT(\nu_1)$, $VT(\nu_2)$, $ET(e)$, F)
17: UPDATE_INDEX(e, $VT(\nu_2)$, b, b_R)
// Update the VT relation of ν_1
18: VT(ν_1) $\leftarrow VT(\nu_1) \cup VT(\nu_2)$ - $ET(e)$
// Update the cost of edges, and add these edges to Q
19: *updated_edges* $\leftarrow$ UPDATE_COSTS(ν_1, $VT(\nu_1)$, E, Q)
20: **end if**
21: **end while**
22: $LRU \leftarrow LRU \cup local\_vts$ // Add $local\_vts$ to the LRU-cache

then they are extracted and saved to LRU. The ET relation of a candidate edge e is extracted by traversing $VT(v_1)$ and finding triangles incident to e (GET_ET procedure at row 11). To check the link condition (row 12), the set of vertices adjacent to ν_1 or ν_2 are extracted on the fly in

LINK_CONDITION by traversing the VT relations of ν_1 and ν_2.

If e satisfies all three conditions, then it is contracted to its optimized position (i.e., v_1) by function CONTRACT (row 15). This procedure takes as input edge *e*, the VT relation of ν_2, the ET relation of *e*, the array of vertex error quadrics *E*, and underlying mesh Σ of the TIN Θ. It removes vertex ν_2 as well as the two triangles adjacent to *e*. In each remaining triangle in $VT(\nu_2)$, it replaces ν_2 with ν_1. After the contraction, the error quadric of v_1 is updated by adding the quadric of ν_2 to it. The pseudo-code of function CONTRACT is in Algorithm 3 (see Appendix A.4).

After the contraction, both the Forman gradient *F* and the Terrain tree *T* are updated (rows 16-18). The update of gradient *F* (UPDATE_GRADIENT procedure at row 16) follows the method introduced in Section 4.3. This involves up to four triangles adjacent to triangles in $ET(e)$ (see Fig. 4.3 for an example). Such triangles are retrieved by using the corresponding VT and ET relations.

The update of *T* is performed by function UPDATE_INDEX (row 17). If e is an *internal edge*, the current leaf block *b* is updated by removing the index of ν_2 and the indexes of the triangles incident in e. If e is a *cross edge* and ν_1 is indexed in leaf block b_1, then both *b* and b_1 are updated in a similar way. In this latter case, the indexes of those triangles that were incident in ν_2 but not encoded in b_1 are also added to b_1. The VT relation of vertex ν_1 is updated by adding the triangles in the VT relation of ν_2 and by removing the triangles adjacent to *e* (row 18).

Since the error quadric of v_1 is updated after the contraction of e, the costs of all edges currently incident in v_1 need to be updated accordingly (row 19). A local auxiliary array *updated_edges* which is initialized in row 4, is used to keep track of the updated edge costs. All updated edges are added to *Q* again. Note that we do not update the costs of edges in *Q* directly,

and, therefore, each time we process an edge *e* from *Q*, we check if e has been updated in previous contractions (row 6). If e has been updated and the cost stored with *e* is not the one stored in *updated_edges* then we discard *e* and process the next edge in *Q*.

Finally, after the simplification of leaf block *b*, the $local\_vts$ array, containing the updated vertex-triangle relations, is inserted into LRU (row 22).

4.5 Parallel topology-aware edge contraction on Terrain trees

In this section, we propose a parallel algorithm that extends and enhances the algorithm described in Section 4.4. In subsection 4.5.1, we introduce the design of this parallel algorithm. The main challenge of parallel mesh simplification is to prevent conflicts that may occur if two threads modify the same vertex or triangle concurrently. In subsection 4.5.2, we discuss why the proposed parallel algorithm can avoid such conflicts.

4.5.1 Parallel edge contraction algorithm

The hierarchical domain decomposition of Terrain trees makes them well-suited for parallel computation since different leaf blocks can be processed at the same time. The key idea behind our parallel simplification strategy is to assign each leaf block to a single thread from a set of available threads. To avoid conflicts between two different threads, we have devised a *leaf-locking strategy*, which is based on the definition of *conflict block*. A leaf block b_0 is called a *conflict block* for another leaf block b_1 if there exists a cross edge $e = \{v_1, v_2\}$, where v_1 is in b_0 and v_2 is in b_1. Clearly, in this case, b_1 is a conflict block for b_0 as well.

There are four possible statuses a leaf block *b* may have: (1) *active*; (2) *default*; (3) *conflict*;

and (4) *processed*. A leaf block *b* is *active* when it is being processed. A leaf block *b* is in the *default* status if *b* is neither being processed nor a conflict block of any *active* block. A conflict block of an *active* block is set to the *conflict* status. When the simplification process is completed, block *b* is set to the *processed* status. A leaf block *b* can be processed only if its status is *default* and none of its conflict blocks has *conflict* status.

The parallel simplification strategy performs the following steps:

1. *Generating auxiliary data structures*: The list of all conflict blocks of a leaf block *b*, denoted as $Cl(b)$, is computed by traversing all triangles encoded in *b*. Given a triangle t with at least one vertex in *b*, we check if the other two vertices of t are also in *b*. If a vertex v of t is not in *b*, then, we locate the block b_i containing v, and add b_i to $Cl(b)$.

2. *Computing the initial error quadrics*: The initial error quadric of each vertex is computed using a parallel version of the algorithm introduced in Appendix A.3.

3. *Simplification*: Each leaf block is simplified by a thread following the steps described in Algorithm 1 with one difference. Each thread needs to update the lists of conflict blocks which change due to the undergoing simplifications, as described below.

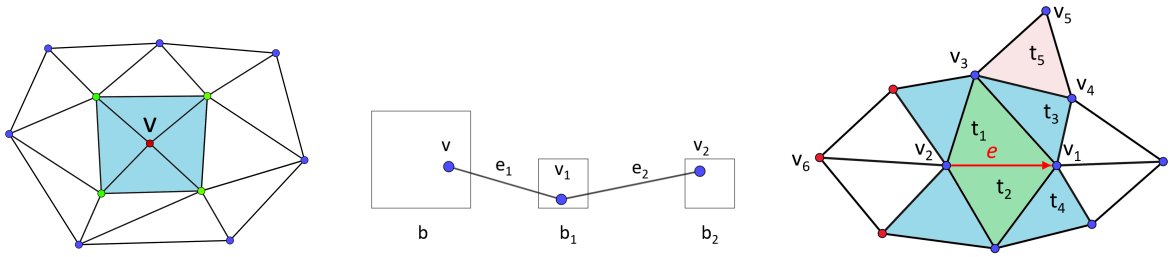
Assume a *cross edge* $e = \{v_1, v_2\}$ is contracted to vertex v_1, with v_1 in block b_1 and v_2 in block b_2. Note that e is simplified only when b_2 is an *active* block. Vertices adjacent to v_2, but not to v_1 (for instance, red vertices in Fig. 4.4(c)), are connected to v_1 after contracting edge e to v_1. For example, if vertex v_6 is not encoded in either b_1 or b_2, the edge connecting v_6 and v_1 is a cross edge and may create a new conflict block for b_1.

After contracting a cross edge, the list of conflict blocks is updated through a modified LINK_CONDITION procedure (row 12 of Algorithm 1). The modified procedure extracts also an

auxiliary array vv_{outer}, which encodes the vertices adjacent to v_2 that are not contained in either b_1 or b_2. After the CONTRACT procedure (row 15 of Algorithm 1), we add a step for updating the conflict block list. To update $Cl(b_1)$ after the contraction of e, we find, for each v' in vv_{outer}, the leaf block b' that contains v', and add it to $Cl(b_1)$ if it has not been added yet. Similarly, b_1 is added to $Cl(b')$ when b' is added to $Cl(b_1)$. The update of $Cl(b_1)$ and $Cl(b')$, when processing b_2, does not affect the concurrent simplification of other blocks. Thanks to the definition of conflict block, both b' and b_1 are conflict blocks of b_2 and, thus, they cannot be active when b_2 is active. Also, being b' and b_1 in a *conflict* state, none of leaf blocks in their conflict lists can have an *active* state.

In contrast to the sequential algorithm, the parallel one does not use a global LRU cache C for storing VT relations, since it could raise resource conflicts when multiple threads access C at the same time. Instead, a local cache at a thread level provides a safe way to encode just the VT relations of blocks in $Cl(b)$ when processing b. Similar to the sequential case, the local cache is accessed and updated only when simplifying a *cross edge*. Once the simplification of b is finished, the local cache is discarded. In subsection 4.5.2, we show why the local caching strategy is thread-safe thanks to the leaf-locking strategy.

We use OpenMP [37] to process multiple leaf blocks in parallel in a Terrain tree. It is noteworthy that, while each step makes use of multi-threading internally, the three steps are organized sequentially, i.e., each step of the pipeline is executed only when the previous one is completed. Since the computations performed in steps 1 and 2 are entirely local to a leaf block, they can be processed in a perfectly parallel manner. In step 3, conflicts among threads can prevent the simplification of a leaf block and, thus, the list of the blocks is traversed multiple times until all blocks are simplified.



(a) (b) (c)

Figure 4.4: (a) Example of 1-neighborhood (vertices in green) and 2-neighborhood (vertices in green and vertices in blue) of a vertex v. (b) Example of a vertex v and two vertices in its 2-neighborhood. (c) Example of edge contraction on edge $e = \{v_1, v_2\}$ (in red), where v_2, t_1, and t_2 are removed by the contraction. Gradient pairing information of cyan triangles may be modified due to the contraction of e. Red vertices are reconnected to v_1 after the contraction during which v_2 is removed.

4.5.2 Discussion on the leaf-locking strategy

We prove here that the *leaf-locking strategy* introduced in subsection 4.5.1 ensures that no conflict occurs between threads during a parallel simplification process. Given a vertex v in a triangle mesh Σ, we call the set of vertices adjacent to v in Σ as the *1-neighborhood* of v. We then define the *2-neighborhood* of v as the set of vertices that share an edge with any vertex in the 1-neighborhood of v excluding v itself. By this definition, the 1-neighborhood of v is a subset of its 2-neighborhood. An example of 1-neighborhood and 2-neighborhood of v in Σ is displayed in Fig. 4.4(a).

Let us consider an edge $e = \{v_1, v_2\}$ being contracted to v_1 on Σ. We need to ensure that: (1) the check on e is not affected by other threads, (2) the update of Σ, the Forman gradient F, and the vertex error quadrics after contracting e do not conflict with any update operations started by other threads, and (3) the local cache does not contain conflicting information with other threads.

When a vertex v in leaf block b_1 is to be removed in an edge contraction operation in the parallel simplification, from the definition of 2-neighborhood, we know that a vertex v' in the

2-neighborhood either is adjacent to v (e.g., v_1 in Fig. 4.4(b)) or has a sharing adjacent vertex with v (e.g., v_2 in Fig. 4.4(b)). We first consider the case that a vertex v_2 shares an adjacent vertex v_1 with v, with two edges $e_1 = \{v, v_1\}$ and $e_2 = \{v_1, v_2\}$ between v and v_2. There are three possible cases for e_1 and e_2: (1) both are internal edges, (2) one is an internal edge and the other one is a cross edge, (3) both are cross edges. In case (1), v and v_2 belong to the same block and cannot be removed at the same time. In case (2), v_2 belongs to a conflict block of b, while in case (3), v_2 belongs to a block b_2 which shares a conflict block b_1 with b as shown in Fig. 4.4(b). In both cases, the block encoding v_2 cannot be processed when b is in status *active* according to the definition of leaf-locking strategy. Recall that for a leaf block b, an edge is only considered as candidate if the vertex to be removed is encoded in b. Therefore v_2 cannot be removed when the block encoding is not set to *active*.

Similarly, when v' is adjacent to v, v' is either in b or in a conflict block of b. In both cases, v' cannot be considered in an edge contraction operation. Therefore, when the parallel simplification uses the leaf-locking strategy, we have the following:

Proposition 4.5.1. *Any vertex belonging to the 2-neighborhood of v_2 cannot be removed by any thread while edge $e = \{v_1, v_2\}$ is being processed and v_2 is the vertex to be removed.*

From Proposition 4.5.1, we have that no triangle in the VT relations of v_1 or v_2 can be modified by other threads. Therefore the validation of link and fold conditions, and the update of Σ after contracting e cannot be affected by other threads. Similarly, the error quadric of vertex v_1 can only be updated by a single thread, otherwise the other vertex being removed is in the 1-neighborhood of v_1 and in the 2-neighborhood of v_2.

We discuss now how to check and update the Forman gradient F during parallel simpli-

fication. From the description of the gradient-aware edge contraction in Section 4.3, we have that:

Proposition 4.5.2. *The gradient pairing information associated with a triangle t can be modified during the contraction of edge e only if t is adjacent to a triangle incident in e, i.e., t is adjacent to a triangle to be removed during the contraction of e.*

The validation of the gradient condition involves only the triangles in VT(v_2). It is straightforward to prove that if another edge contraction operation is modifying the gradient pairing information of a triangle in VT(v_2), then the vertex to be removed by that operation is in the 2-neighborhood of v_2 and, thus, breaks the leaf-locking strategy condition.

Now we prove that the gradient pairing information associated with a triangle cannot be modified by two threads at the same time. Suppose triangle t_3 in Fig. 4.4(c) is modified by two threads Th_1 and Th_2 at the same time and edge e is being removed by Th_1. From Proposition 4.5.2, we know that t_3 should be adjacent to two triangles being removed by Th_1 and Th_2, respectively. Without loss of generality, we assume that t_5 (pink triangle in Fig. 4.4(c)) is a triangle to be removed by Th_2. Then, either $\{v_4, v_5\}$ or $\{v_3, v_5\}$ is the edge to be removed by Th_2. In both cases, the vertex to be removed is in the 2-neighborhood of v_2, which violates Proposition 4.5.1.

We have proved that the leaf-locking strategy ensures that the validation of three conditions and most of the update within an *active* block will not be affected by other threads. But it is possible that the cost of one edge is updated by different threads at the same time. Let us consider an edge $e_1 = \{v_1, v_2\}$ being contracted to v_2 and another edge $e_2 = \{v_3, v_4\}$ being contracted to v_3 on Σ. If v_2 and v_3 are connected by an edge e_0 (see Fig. 4.5(a)), it is still possible that e_1 and

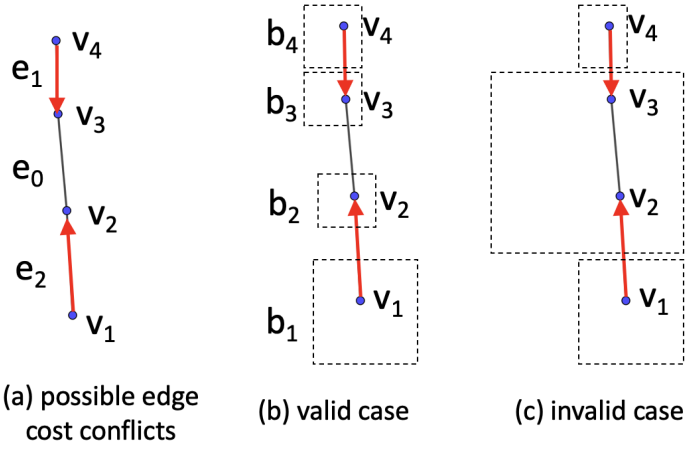


Figure 4.5: (a) Example of a possible conflict occurring when two edges are contracted at the same time (triangles are not displayed for clarity). Edge $e_1 = \{v_1, v_2\}$ is contracted to v_2 and edge $e_2 = \{v_3, v_4\}$ is contracted to v_3. (b) A case that blocks encoding v_1 and v_4 can be active at the same time under the leaf-locking strategy (c) An invalid case in which v_1 and v_4 cannot be removed at the same time.

e_2 are contracted by different threads at the same time since v_1 and v_4 are not in each other's 2-neighborhood. Assume that e_1 is contracted on Th_1 and e_2 is contracted on Th_2. If Th_1 updates the error quadric of v_2 and the cost of e_0 before error quadric of v_3 is updated on Th_2, then Th_2 will have a different updated cost of e_0 since it calculates with two updated error quadrics.

But such a conflict will not affect the simplification on either thread, since, this case can only happen when e_0, e_1, and e_2 are all cross edges (see Fig. 4.5(b)). Otherwise, like the example in Fig. 4.5(c), leaf blocks encoding v_1 and v_4 must be conflict block of each other and so that e_1 and e_2 cannot be simplified at the same time. When all three edges are cross edges, neither endpoints of e_0 is encoded in the same block as v_1 or v_4, and thus, it is not a candidate edge in b_1 or b_4. Therefore, even if the cost of an edge is updated by different threads, such edge is not a candidate edge of current *active* blocks (i.e., b_1 and b_4) and will not cause a conflict between these threads.

The proof of the thread safety of the local caching strategy is straightforward. Given a leaf

block b_1, the local cache of b_1 only encodes the VT relations of its conflict blocks. If b_2 is one block encoded in the local cache of b_1, then all conflict blocks of b_2 except for b_1 itself, cannot be *active*. Therefore, the information of b_2 is encoded only in the local cache of the thread that simplifies b_1.

4.6 Terrain tree update after simplification

After the simplification, the Terrain tree is no longer as compact or efficient. Since the number of vertices within each leaf block is reduced, the original tree structure becomes too loose with respect to the simplices encoded in the simplified mesh. Besides, while initially, each block *b* in the Terrain tree encodes vertices that fall in *b* and triangles intersecting with *b*, this property is not guaranteed during the simplification.

Recall that we mentioned in Section 4.4 that when an edge $e = \{v_1, v_2\}$ is contracted to v_1, the UPDATE_INDEX procedure updates related blocks by removing v_2 and adding triangles that are incident to v_2 to the block containing v_1. However, this procedure does not update the triangle lists of each leaf block b when the intersection relation between a triangle and b is changed after the contraction. This design decision is motivated by the fact that: (i) removing an index from ranges encoded through SRE and performing the triangle-in-block intersection tests are time-consuming operations, and (ii) keeping this outdated information does not affect the correctness of the simplification operation. Additionally, when blocks are merged, duplicated triangles in the merged triangle list must be removed. Therefore, after the simplification, the triangle list of each leaf block needs to be updated.

Lastly, after the simplification and merging, the vertex list and triangle list of each block

Algorithm 2 UPDATE_TRIANGLE_LISTS(b, l_b)

Input:

b: the current block, can be either leaf block or internal block. b_t is its original triangle list

l_b: label of the current block with respect to its parent

Output:

h: a hash table stores triangles that need to be checked against other blocks

```
1:  if b is leaf block then // Leaf block case
2:      h_t ← { } // Create a hash set storing the triangles that should be kept in b_t
3:      for triangle t in b_t do
4:          if t ∈ h_t or INTERSECT(t, b) == False then // t is a duplicate or has no intersection with
            b
5:              continue
6:          else
7:              add t to h_t  // The current triangle should be kept
8:              n ← VERTEX_IN_BLOCK(t, b)  // Count the number of vertices of t in block b
9:              if n ≠ 3 then // If t is not fully contained by b
10:                 add (t, l_b) to h
11:             end if
12:         end if
13:     end for
14:     b_t ← h_t  // Set the triangle list of current block to h_t
15: else // Internal block case
16:     h_I ← { } // Create a hash table to store the triangles returned by its four sub-blocks
17:     for i ← GET_CHILDREN(b) do
18:         c ← GET_CHILD(b, i) // Get the i-th children of b
19:         add UPDATE_TRIANGLE_LISTS(c, i) to h_I // Merge the results to h_I
20:     end for
21:     for t_i in GET_KEYS(h_I) do
22:         n ← VERTEX_IN_BLOCK(t_i, b)
23:         if n ≠ 3 then
24:             add l_b to h[t_i]
25:         end if
26:         for i ← GET_CHILDREN(b) do
27:             c ← GET_CHILD(b, i)
                // If t_i has not been checked against the sub-block c and it intersects with c
28:             if i ∉ h_I[t_i] and INTERSECT(t_i, c) == True then
29:                 INSERT_TRIANGLE(t_i, c) // Insert t_i to a tree in which c serves as the root block
30:             end if
31:         end for
32:     end for
33: end if
34: return h
```

may not be accurate since they may contain vertices and triangles that have been removed. In the UPDATE_INDEX procedure, the removed vertices and triangles are flagged as deleted in the corresponding global vertex and triangle arrays. Although one can check the global arrays to determine if a vertex or a triangle in the local arrays of a block b is removed or not, additional time for performing these checks and additional memory for storing this information are required, ultimately affecting the efficiency of a Terrain tree.

Considering the above factors, a post-processing step is performed to update the Terrain tree Σ after the simplification of the encoded mesh Σ. This post-processing involves the following steps:

1. update the subdivision of *T* based on the remaining vertices;
2. reindex the vertices in Σ and encode the vertex list within each leaf block through SRE;
3. check intersections between triangles and blocks and update the triangle list of each block in the new subdivision;
4. reindex the triangles in Σ and encode the triangle list within each leaf block through SRE.

In *step (2)* and *step (4)*, the vertex and triangle lists of each block are encoded through SRE again to keep the Terrain tree compact. These steps are performed in the same way as the procedures used during the initial generation of the Terrain tree [72]. In the following, we describe the details of *step (1)* and *step (3)*.

Since the PR Terrain tree is used in our simplification, the nested subdivision of *T* depends solely on the vertices in Σ. Let us assume the capacity of *T* is k_V in *step (1)*. To update the subdivision of *T*, we visit *T* through a depth-first-traversal. During the traversal, if an internal

block b_I contains fewer than k_V vertices, all of its sub-blocks are merged and b_I becomes a leaf block. After this, the resulting subdivision matches the one obtained when we generate a new PR-Terrain tree based on the vertices of the simplified mesh. When merging blocks, both the triangle lists and the vertex lists of these leaves are merged. Since one vertex can only be indexed by one leaf block, there is need to remove duplicates from the merged vertex list. After this step, duplicates may exist in the merged triangle list, but they will be removed in *step (3)*.

In *step (3)*, function UPDATE_TRIANGLE_LISTS updates the triangle list of each leaf block. All blocks in *T* are visited again through a depth-first traversal. Given a block *b*, function UPDATE_TRIANGLE_LISTS takes *b* and its label with respect to its parent block as inputs and returns a hash table h storing triangles in *b* that intersect with other blocks. The value of l_b can be 0, 1, 2, or 3 indicating the position of *b* in the child list of its parent. The key in h is the index of a triangle that needs to be checked against other blocks and the corresponding value is the input l_b.

When *b* is a leaf block, each triangle t in the triangle list of *b* is checked to understand if t should be kept in the local list of *b* and if t is fully within *b*. If t is not entirely in *b*, it is passed to the parent block of *b* to be checked in the neighborhood of *b*. When *b* is an internal block, triangles passed from each of its children are checked to understand if they intersect with the other three children of *b* and if they are fully within *b*. Similar to the leaf block case, a triangle partially in *b* is passed to the parent block of *b* to be checked again. Algorithm 2 provides a pseudo-code description of the algorithm to update the triangle list of a block *b*.

When the current block *b* is a leaf block (row 1-14), a hash set h_t is used to store triangles to be kept in *b* and to avoid duplicates. For each triangle t in the original triangle list b_t of *b*, we first check if t is in h_t to understand if it is a duplicate and then we check if t has an intersection

with b (row 4, INTERSECT procedure). t is added to the new triangle list of b, i.e., h_t (row 7), only if it is not already in h_t and it intersects b. VERTEX_IN_BLOCK procedure counts the number of vertices of t within block b, which is denoted as n (row 8). Since the vertex index range of b is encoded through SRE [74], checking if a vertex v is in b can be done in constant time by comparing the index of v to the pair of numbers representing the vertex range of b. If n equals 3, t is fully contained by b. Otherwise, t is partially within b and it should be checked against other blocks. The pair (t, l_b) is then added to h (row 9-11). The triangle list of b is updated to h_t (row 14).

When the current block b is an internal block (row 15-32), for each child b_i of b, the triangles to be checked against other children are collected through function UPDATE_TRIANGLE_LISTS and added to a local hash table h_I (row 17-20). Given a triangle t in h_I, the value of h_I[t] is a list of sub-blocks of b which have t in their hash tables. The intersection relations between these sub-blocks and t are known, and thus, they are not checked against t again. For each triangle t in h_I, we check first if t partially overlaps b and if so, the pair (t, l_b) is added to h and passed to the parent of b (row 22-25). Besides, if t intersects with a sub-block c of b and c is not in $h_I[t]$, t is inserted into the sub-tree rooted in c (row 28-30, INSERT_TRIANGLE procedure).

4.7 Terrain mesh evaluation

In this section, we discuss how we evaluate the quality of meshes obtained from the simplification process. A simplified mesh is an approximation of the original mesh, and therefore it is usually evaluated based on the approximation error with respect to the original mesh. The *approximation error* can be measured by perceptually-based metrics [113], which evaluate the

visual appearance difference between two meshes, or by geometric-based metrics [160]. Visual metrics are usually used to evaluate meshes for visualization and rendering purposes instead of terrain analysis. Such measures are computationally complex since rendering methods and lighting environments should also be considered. Therefore, in most non-rendering applications, geometric measures are often preferred for evaluating approximation error as they are more computationally efficient [160].

One straightforward geometric measure of error is the distance between the simplified mesh and the original mesh. Many common geometric measures rely on the concept of the *Hausdorff distance* [9, 29], which evaluates the distance between two surfaces when no distance direction has been set. When assessing simplified TINs, measures based on vertical distances, i.e., the elevation difference, between meshes are also widely used [26, 196, 199]. Two commonly used measures are the maximum vertical distance and the Root Mean Squared Error (RMSE) of the vertical distance. Compared to the Hausdorff distance, the vertical distance is easier to calculate and more significant in evaluating terrain models. Since it is computationally intensive to obtain vertical distances of all points on the two meshes, we calculate the vertical distances of all vertices in the original mesh to the simplified mesh. The maximum distance and RMSE of such vertical distances are used to measure the approximation error of the simplified meshes.

In addition to the approximation error, another important metric to evaluate a simplified terrain mesh is the shape of triangles. Triangle meshes representing terrain surfaces are usually generated through *Delaunay triangulation* [87]. A triangle t in a mesh Σ satisfies the *circumcircle property* if the circumcircle of t does not contain other vertices of Σ in its interior. A triangulation where all triangles satisfy the circumcircle property is a Delaunay triangulation. Delaunay triangulation is favored when generating terrain meshes because it avoids creating triangles with

very small angles, which deteriorate the interpolation result on the terrain mesh. A terrain mesh is considered to have good triangle shape quality when its projection on the horizontal plane is a Delaunay triangulation. But evaluation metrics based on the Delaunay triangulation definition are very sensitive to mesh modification since contracting a single edge in a Delaunay triangulation may make it invalid. Moreover, these metrics evaluate the global triangle shape quality of a mesh, and do not provide a quantitative evaluation of a single triangle. Therefore, quantitative triangle shape measures are more widely used in mesh evaluation. In this study, we use the triangle shape measure proposed by Guéziec [86]:

$$\kappa = \frac{4\sqrt{3}A}{l_1^2 + l_2^2 + l_3^2} \tag{4.1}$$

where A is the area of the triangle, and l_1, l_2, l_3 are the lengths of three edges, respectively. The shape quality of a triangle t ranges from 1 (for an equilateral triangle) to 0 (when all three vertices of t are collinear).

Rather than solely focusing on geometric information, some applications, such as nautical charting [167] and tree segmentation [195], find it more beneficial to preserve high-level information about terrain structure. One example is topographic information such as a terrain's peaks, passes, and basins, which correspond to maxima, saddles, and minima. However, standard simplification algorithms can disrupt topological information in an uncontrolled manner. That is, a simplification algorithm can remove critical simplices, introduce new critical simplices or change the connectivity of critical simplices. Therefore, it is important to evaluate not only the geometric quality of a simplified mesh but also how much a simplification process affects the mesh topology.

To effectively and efficiently compare the topological information of two terrains, a topological descriptor and a distance function are required. As introduced in Section 2.1.2.2, one of the most popular topological descriptors is the *persistence diagram* [58]. Numerous algorithms have been defined to compute the persistence diagram by efficiently pairing points on an input scalar function [98].

For the sake of this work, it is important to notice that any change in the topology of a terrain corresponds to a change in the persistence diagram in the form of disappearing, appearing, or moving points. The major advantage provided by this descriptor is that it enables the measurement of differences between two persistence diagrams by means of distances such as the bottleneck distance [31], the Wasserstein distance [32], or the sliced Wasserstein distance [17].

In this work, we use the sliced Wasserstein distance to measure the distance between two persistence diagrams because faster to compute [17]. Formally, the q-th Wasserstein distance between two persistence diagrams X and Y [32] is defined as:

$$W_q(X,Y) = \left[\inf_{f:X\to Y} \sum_{x\in X} \|x - f(x)\|^q \right]^{1/q} \tag{4.2}$$

where f ranges over all bijections from X to Y. This distance measures the minimum total cost to match one persistence diagram X with another diagram Y. As computing all possible matchings in a 2D space is computationally expensive, the sliced Wasserstein distance approach [17] involves sampling this space by projecting all points onto a line passing through the origin. Subsequently, the Wasserstein distance is calculated in 1-dimension. In practice, the sliced Wasserstein distance is obtained by computing the 1-dimensional Wasserstein distances with respect to a group of lines and then getting the average of these distances.

We expect that using the topology-aware simplification method, the sliced Wasserstein distance between the simplified and the original mesh is zero. This is because the simplification process does not remove, add, or move critical simplices, thereby resulting in no change in the corresponding persistence diagram.

4.8 Experimental results

In this section, we evaluate the sequential and parallel topology-aware terrain simplification algorithms from three aspects: computing performance, compression rate, and the quality of simplified meshes. We evaluate both algorithms by comparing them to the topology-aware simplification on the most compact triangle-based data structure for meshes.

In subsection 4.8.1, we evaluate the computing performance of both algorithms. In subsection 4.8.2, we evaluate the compression rate, defined as the ratio between the number of removed vertices and the number of vertices in the original mesh. In subsection 4.8.3, we evaluate the quality of the simplified meshes by considering triangle shape quality, approximation error, and topological quality. An optimization strategy to improve output mesh quality is raised and evaluated in subsection 4.8.4. Additionally, we discuss how the selection of leaf block capacity for Terrain tree generation affects the output mesh quality in subsection 4.8.5. Note that the time measurements reported in the following experiments represent the elapsed time.

All experiments are performed on a dual Intel Xeon E5-2630 v4 @2.20Ghz CPU (20 cores in total), and 64GB of RAM. A total of six TINs, generated from raw point clouds using the *CGAL* library [24], are used in our comparisons. The number of vertices per TIN varies from 25 to 113 million (see Table 4.1). *Molokai* is a dataset consisting of both hydrographic and

Table 4.1: Overview of experimental datasets. For each terrain, the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$ are listed.

	Molokai	Great Smokey Mountains	Canyon Lake	Yosemite Rim Fire	Dragons Back Ridge	Moscow Mountain
$\|\Sigma_V\|$	25M	34M	49M	78M	91M	113M
$\|\Sigma_T\|$	50M	68M	98M	155M	182M	226M

topographic point cloud data provided by NOAA National Centers for Environmental Information [140]. *Great Smokey Mountains*, *Canyon Lake*, *Yosemite Rim Fire*, *Dragons Back Ridge*, and *Moscow Mountain*, are topographic LiDAR point clouds from the OpenTopography repository [143]. All six datasets are publicly available and can be accessed either from the NOAA data portal or the OpenTopography website. The source code of the Terrain trees-based simplification algorithm is available at [170].

4.8.1 Performance evaluation

In this subsection, we evaluate the performances of both the sequential and parallel topology-aware terrain mesh simplification algorithms on the Terrain tree. In subsection 4.8.1.1, we compare the performance of the sequential topology-aware simplification on the Terrain trees against our implementation on the IA data structure [145], which is the most widely-used compact triangle-based data structure for meshes. In subsection 4.8.1.2, we compare the performance of the sequential and parallel simplification strategies implemented on the Terrain trees.

The generation of the Terrain tree relies on a single parameter that defines the maximum number of vertices allowed in each leaf block of the decomposition, known as the *leaf block capacity*. In Section 3.7.1, we introduced a strategy to select the suitable block capacity for each dataset when generating a Terrain tree. With the same strategy, we evaluate the computing

performance of the simplification when different capacity values are used for each dataset. This process is elaborated upon in Appendix A.5. In the following, for each dataset, we use the capacity value showing the best trade-off between simplification time and memory requirements.

4.8.1.1 Topology-aware mesh simplification on the Terrain tree and IA data structure

Recall that the IA data structure encodes a vertex array, containing the coordinates of the vertices of the TIN plus the elevation, and a triangle array that encodes, for each triangle t, the indexes to its three vertices plus the indexes in the triangle array of the three triangles sharing an edge with t. In our implementation [71], we use an enhanced version that also encodes for each vertex v, the index of one triangle incident in v. Such an optimization allows extracting all vertex-based relations in optimal time, i.e, in time linear in the size of the output, thereby significantly enhancing the efficiency of the IA data structure during edge contractions.

Given a user-defined threshold ω, we simplify all contractible edges with a cost lower than ω. Based on the initial error quadrics, we compute the costs for all edges in Σ, and use the quartile values as three different thresholds for the simplification. Table 4.2 compares the time and memory cost of the two methods when the cost threshold is set to the first, the second, and the third quartile edge costs, referred to as Q1, Q2, and Q3, respectively. The result shows that the simplification on the Terrain tree is always faster. When using a larger value of ω, the Terrain tree is at least twice as fast as the IA data structure. Moreover, as ω increases, the memory requirements on the IA data structure also increase, whereas they remain stable on the Terrain tree. The difference in the timings and memory requirements is even more relevant when edges

Table 4.2: Time **T** (in minutes) and peak memory usage **M** (in Gigabytes) of the simplification on the IA data structure and the Terrain tree (TT) when using different cost threshold ω. **Q1**, **Q2**, and **Q3** represent the first, second, and third quartile edge costs of each dataset, respectively. The best performance value is denoted in bold face and blue color.

	Molokai						Great Smokey Mountain						Canyon Lake					
ω	**Q1**		**Q2**		**Q3**		**Q1**		**Q2**		**Q3**		**Q1**		**Q2**		**Q3**	
	IA	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**	**IA**	**TT**
T	13.6	**10.5**	21.6	**13.2**	31.8	**17.4**	**15.0**	**15.0**	26.7	**20.5**	44.3	**23**	39.4	**28.5**	79.2	**36.1**	96.8	**39.3**
M	17.5	**12.7**	18.7	**12.7**	20.0	**12.7**	23.7	**17.2**	25.5	**17.2**	27.2	**17.2**	34.2	**24.6**	36.9	**24.6**	39.2	**24.6**

Table 4.3: Time **T** (in minutes) and peak memory usage **M** (in Gigabytes) of topology-aware mesh simplification on the IA data structure, and on sequential (seq.) and parallel (para.) versions on Terrain trees. The best performance value is denoted in bold face and blue color.

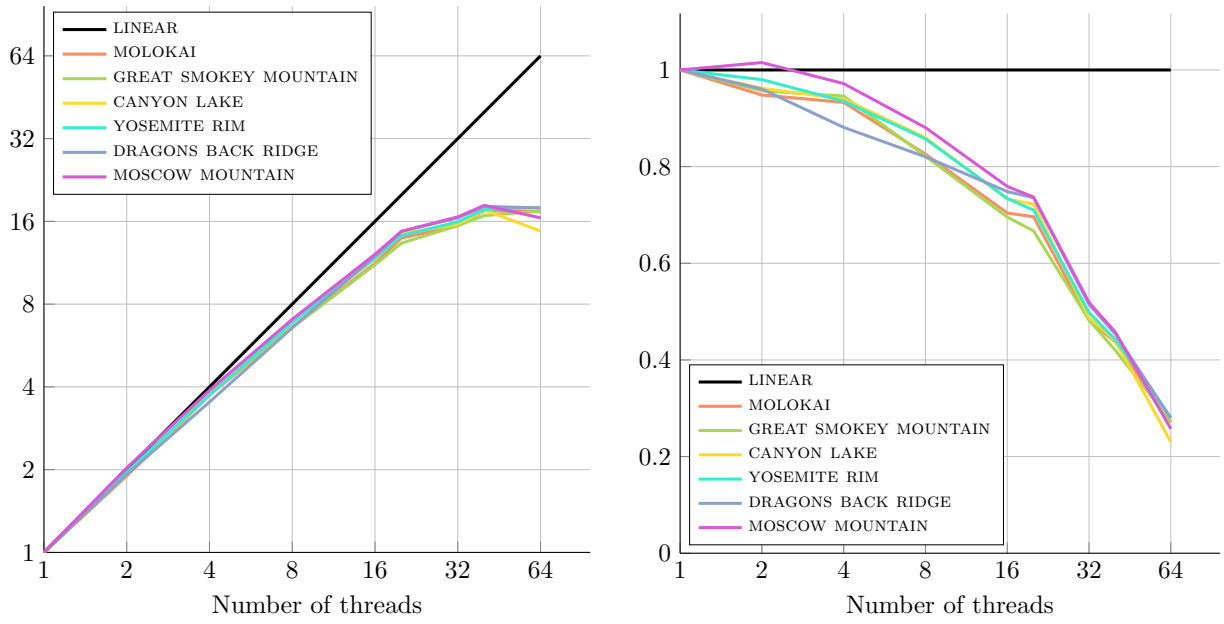
	Molokai			Great Smokey Mountain			Canyon Lake			Yosemite Rim			Dragons Back Ridge			Moscow Mountain		
	IA	**TT**		**IA**	**TT**		**IA**	**TT**		**IA**	**TT**		**IA**	**TT**		**IA**	**TT**	
		seq.	**para.**		**seq.**	**para.**		**seq.**	**para.**		**seq.**	**para.**		**seq.**	**para.**		**seq.**	**para.**
T	58.0	29.0	**2.38**	71.0	39.1	**3.31**	144.1	65.1	**5.34**	-	100.1	**8.12**	-	99.8	**7.85**	-	170.9	**13.7**
M	21.3	**12.7**	12.8	29.0	**17.2**	17.4	41.8	**24.6**	25.0	O.O.M.	**39.0**	39.6	O.O.M.	**45.9**	46.5	O.O.M.	**57.4**	58.0

are simplified in bulk without setting a specific threshold for the edge cost.

Table 4.3 summarizes the results obtained when simplifying all contractible edges. The results include the timings required for computing initial error quadrics and performing the topology-aware simplification, as well as the memory footprint required by the simplification. On average, Terrain trees use from 45% to 56% less time than the IA data structure. Additionally, the memory peak on Terrain trees is approximately 41% less than that of the IA data structure. Due to the higher memory requirements, only three of the test datasets can be simplified using the IA data structure.

4.8.1.2 Parallel topology-aware mesh simplification on the Terrain tree

We evaluate here the performance of the parallel topology-aware mesh simplification algorithm introduced in Section 4.5.1 when using from 1 to 64 threads.



(a) Speedup

(b) Efficiency

Figure 4.6: (a) Speedup and (b) efficiency achieved by the parallel simplification algorithm when different numbers of threads are used.

The speedup of a parallel algorithm is defined as $S = T_1/T_N$, where T_N is the time for the parallel algorithm using N threads, T_1 is the time for the parallel algorithm using a single thread. Fig. 4.6(a) shows the speedup achieved by the parallel simplification algorithm when the number of threads increases. The approach scales well as long as the number of threads is lower than the number of physical cores (20 in our setup). Although the speedup continues to slightly increase with more than 20 threads, it starts to decrease beyond 40.

The efficiency of the parallel algorithm is computed as $E = S/N$, where S is the speedup of the parallel algorithm using N threads. Fig. 4.6(b) shows the efficiency results, revealing a reduction in efficiency as the number of threads increases. This is common for parallel algorithms due to possible load imbalance and overheads during the computation. When using 20 threads, the efficiency of the parallel simplification is 67% on all experimental datasets. With more than 20 threads the efficiency decreases faster. Considering these results, we observe that the best

trade-off is achieved when the thread number is equal to the number of available cores.

Comparing the parallel and sequential mesh simplifications using the same Terrain tree and with 20 threads, the parallel simplification strategy provides a 12x speedup compared to the sequential strategy (see Table 4.3). Furthermore, despite processing multiple leaf blocks concurrently, the parallel strategy maintains a stable memory footprint. On average, it uses only 1% more memory than the sequential algorithm. These results underscore the scalability and efficiency of the Terrain tree representation, particularly when using shared-memory processing techniques.

4.8.1.3 Performance of Terrain trees update

In subsection 4.6, we introduced an algorithm to update the triangle lists of leaf blocks after simplification. An alternative solution is to clear all triangle lists and insert all triangles to the Terrain tree again. We call the former method as the *update* method and the latter one the *reinsert* method.

In this subsection, we evaluate the *update* method by comparing it with the *reinsert* method. Since the comparison shows the same result on all datasets, in this section, we show just the results from the *Molokai* dataset, the *Great Smokey Mountains* dataset, and the *Canyon Lake* dataset and results on the other three datasets can be found in Appendix A.6. Fig. 4.7 shows the time for updating the Terrain trees after simplifying the same mesh with different cost thresholds. As expected, the time for updating the Terrain tree decreases when more edges are contracted. This is because both methods need to iterate through all triangles in the simplified mesh, thus requiring more time to update the tree if fewer triangles are removed.

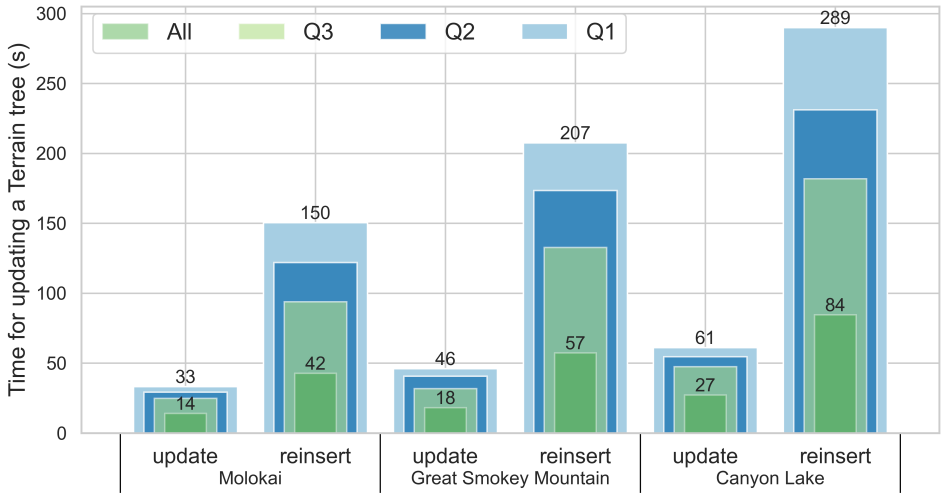


Figure 4.7: The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original *reinsert* method. **Q1**, **Q2**, and **Q3** represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. **All** represents the result when contracting all contractible edges without a cost threshold. The text labels on the boxes in figure indicate the time when **Q1** cost threshold is used and when **All** edges are contracted.

We can see that the *update* method is always faster than the *reinsert* method, requiring 68-77% less time. When the cost threshold is set to the first, second, or third quartile edge costs, the tree update time with the *reinsert* method is longer than the simplification time of the parallel simplification, requiring 113-291% of the simplification time. Conversely, the tree update time with the *update* method ranges from 27-63% of the simplification time. Compared to the *reinsert* method, the *update* method requires additional space to store triangles to be processed by other blocks. However, our experiments show that there is no significant difference in the memory costs between the two methods, and both require less memory than the simplification itself.

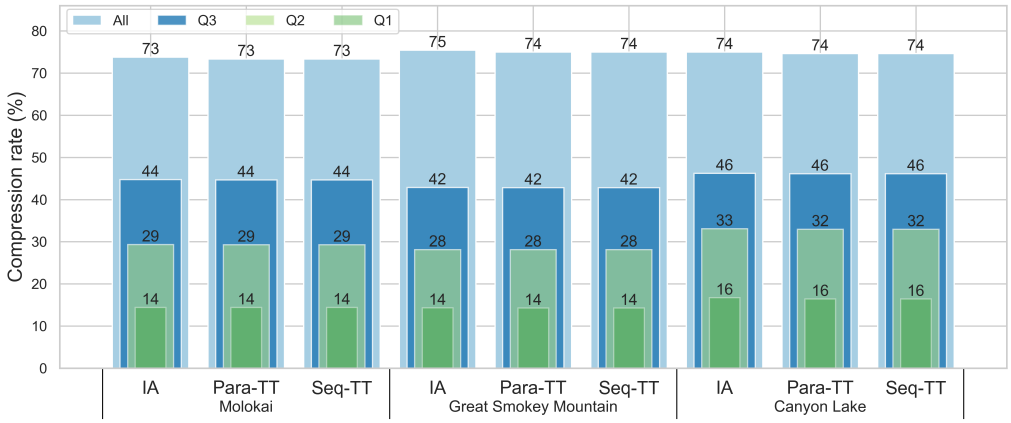


Figure 4.8: The compression rates of three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (**Q1**, **Q2**, and **Q3**) and when all contractible edges are contracted (**All**).

4.8.2 Compression rate evaluation

Both the Terrain tree and the IA data structure use a priority queue for sorting candidate edges, as described in Algorithm 1. The Terrain tree uses a local priority queue for candidate edges within each leaf block, while the IA data structure utilizes a global queue for storing all candidate edges of the TIN. The use of different priority queues leads to different orders in which edges are contracted. For clarity, we refer to the sequential topology-aware simplification on Terrain trees as the *Seq-TT* method, the parallel topology-aware simplification on Terrain trees as the *Para-TT* method, the topology-aware simplification on the IA as the *IA-based* method. To assess the impact of local queues on the simplification process, we compare the compression rates of the Seq-TT, Para-TT, and IA-based methods while varying the quality control parameter.

Similar to the experiments in subsection 4.8.1, we compute the compression rates of these three methods using different cost thresholds: the first quartile edge costs (Q1), the second quartile edge costs (Q2), and the third quartile edge costs (Q3) in the original mesh. Additionally, we compute the compression rate when we simplify all contractible edges, which we refer to as

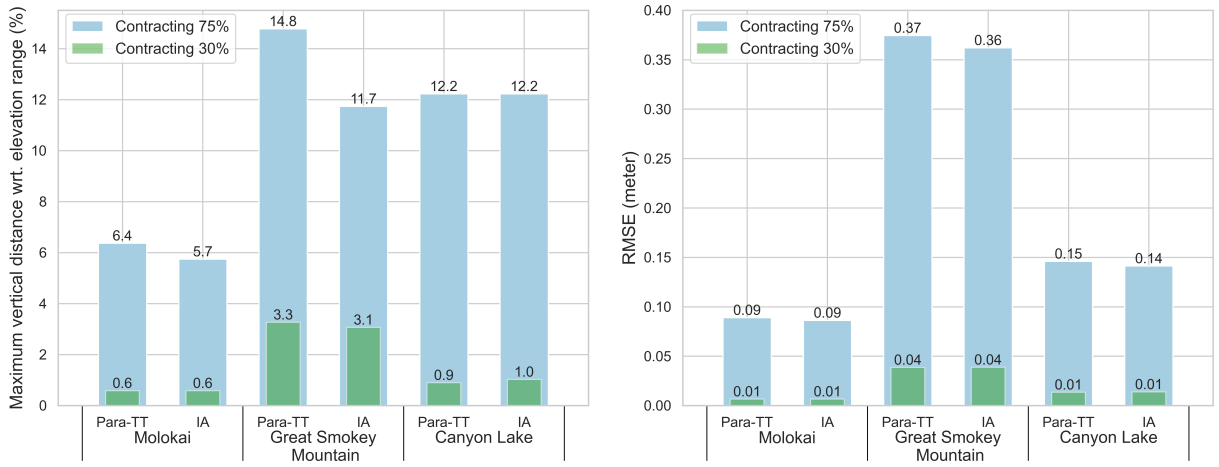
All. The results, illustrated in Fig. 4.8, reveal that the difference in the compression rate between the Seq-TT and the Para-TT methods is very small (less than 0.003%). When comparing the two methods to the IA-based method, the IA-based method always yields a slightly higher compression rate compared to the two methods on the Terrain trees, ranging from 0.02 to 1.7%. This difference is almost negligible when the cost thresholds are small, while it becomes more evident when all contractible edges are simplified. The reason behind it is that the simplification process on the IA data structure relies on a global queue to determine the contraction sequence, allowing it to always contract the edge that introduces the least error. On the other hand, the simplification process on the Terrain trees simplifies the mesh block by block, so the sequence of contractions is not the global optimum.

4.8.3 Mesh quality evaluation after simplification

In this subsection, we evaluate the quality of the simplified meshes by using the metrics introduced in Section 4.7. In subsection 4.8.3.1, we evaluate the triangle shape quality of the simplified mesh using the triangle shape measure. In subsection 4.8.3.2, we evaluate the approximation error of a simplified mesh by calculating the maximum and the RMSE of vertical distances. In subsection 4.8.3.3, we evaluate the topological quality of the simplified meshes by comparing the topology-aware method to simplification algorithm does not take into account the gradient, which we refer to as the *geometric* simplification method. The geometric simplification uses just the edge contraction operator introduced in Section 4.3. Specifically, we refer to the parallel geometric simplification on Terrain trees as the *Geometric-TT* method.

4.8.3.1 Triangle shape measure

When evaluating the average triangle shape measure, our experiment shows that the average triangle shape measures of simplified meshes obtained from different topology-aware simplification methods are quite similar. Therefore the result is not displayed here and we refer interested readers to find the result in Fig. A.14 in Appendix A.6. When all contractible edges are contracted, meshes simplified with the IA-based method have slightly better triangle shape quality than those from the other two methods (around 0.35% better on average). The Para-TT and the Seq-TT methods show comparable results. When the cost threshold is set to Q2, meshes from three different methods have very similar shape quality, with less than 0.1% difference. These findings suggest that the local queues used by Terrain trees can lead to a slightly worse triangle shape quality when the cost threshold is set to infinity or if it is very large. One potential explanation for this is that the contraction of longer edges tends to deteriorate the shape quality of a mesh since adjacent edges are stretched more after updating the mesh. When the cost threshold is small, all three methods do not contract very long edges, so the difference in shape quality is not evident. Conversely, when all contractible edges are contracted (i.e., the cost threshold is set to infinity), the simplification process on the Terrain trees may remove edges with very high costs in the early stage of the simplification if they are in the blocks that are visited early. On the contrary, in the IA-based method, those edges are always contracted in the late stage since a global edge queue is used.



(a) Maximum distance wrt. range (b) RMSE

Figure 4.9: (a) The maximum vertical distance with respect to the elevation range of the original mesh and (b) the RMSE of vertical distances between the original mesh and the simplified mesh obtained from Para-TT and IA-based simplification.

4.8.3.2 Approximation error evaluation

To evaluate the approximation error of the simplified mesh, we compute measures based on the vertical distances between the original mesh and the simplified mesh. We compare the Para-TT method and the IA-based method when the same number of edges are contracted. The results from the sequential simplification on the Terrain trees are very similar to the parallel one. Fig. 4.9(a) and Fig. 4.9(b) show the maximum and the RMSE of vertical distances between the original mesh and the simplified mesh when 30% edges and 75% edges of the original mesh are contracted. When 30% edges are contracted, the difference in the maximum vertical distance between two methods is not evident. For the *Great Smokey Mountains* dataset, the maximum distance on the Terrain trees is 6.5% more than that on the IA data structure. Conversely, for the *Canyon Lake* dataset, we get the opposite result, where the maximum distance of the Para-TT method is 12% less than that of the IA-based method. The RMSE of the Para-TT method is

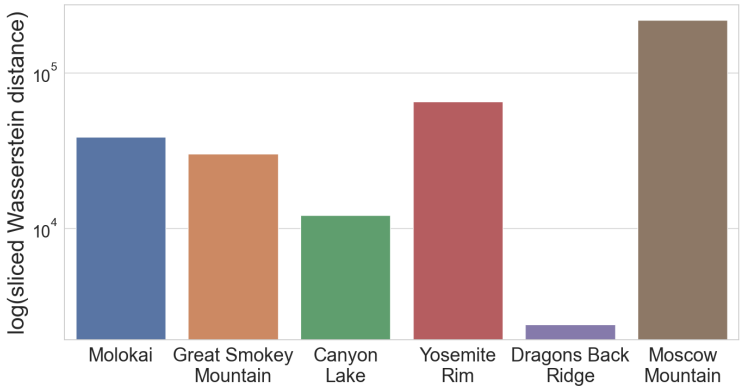


Figure 4.10: The sliced Wasserstein distance between the original mesh and the simplified mesh when the mesh is simplified with the Geometric-TT method.

around 1-2% less than that of the IA-based method, indicating a slightly better global approximation quality.

When 75% edges are contracted, the IA-based method produces meshes with 10-21% smaller maximum distance and 3% smaller RMSE compared to the Terrain trees. The usage of the local queues in the Terrain trees changes the sequence of contractions, which leads to the difference in both the maximum vertical distance and the RMSE. However, this change is dataset dependant and there is no significant quality difference between the two methods when fewer edges are contracted. Similar to the change in the triangle shape measure, when more edges are contracted, the Para-TT method is likely to contract high-cost edges in the earlier stage of the simplification process, so the error increases faster than when using the IA-based method.

4.8.3.3 Topological mesh quality evaluation

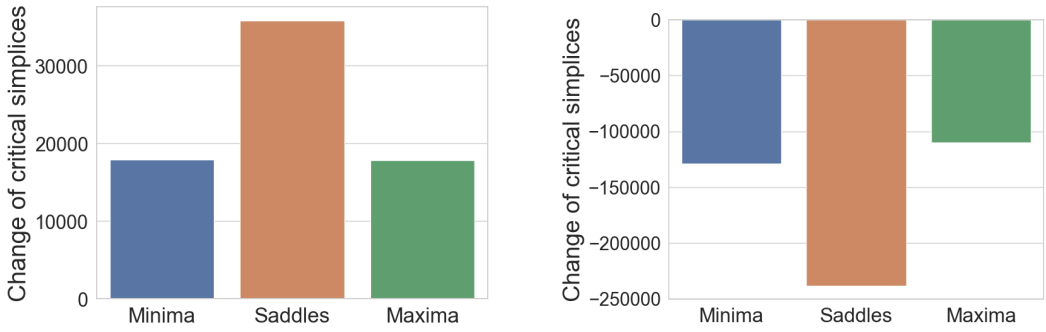
To evaluate how the topology of the mesh may change if the gradient condition is not considered, we remove all contractible edges with the Geometric-TT method. Then we evaluate the topological quality of the simplified meshes with two metrics: (1) the sliced Wasserstein distance

between the original mesh and the simplified mesh, and (2) the change of critical simplices.

We use an open-source program Persim [149] for the calculation of the sliced Wasserstein distance. For each dataset, we extract the persistence diagrams of the original mesh and of the simplified mesh when all contractible edges are removed. Fig. 4.10 shows the sliced Wasserstein distance between those two diagrams when the mesh is simplified with the geometric simplification. The sliced Wasserstein distance remains zero when meshes are simplified with topology-aware simplification and is therefore not shown in the figure. This result aligns with our expectation, as stated in Section 4.7, that the critical simplices are preserved and the persistence diagram does not change during this process. As it can be noted with the sliced Wasserstein distance, the persistence diagrams of different datasets have variable degrees of change when they are simplified by the geometric method. This is because the geometric method does not consider the topology of the terrain during the simplification, and such topology can drastically change during the simplification process.

Among six datasets, it is noteworthy that the sliced Wasserstein distance of the *Dragons Back Ridge* dataset is much smaller compared to the others. This happens since the percentage of critical simplices in this dataset is significantly lower, with only 0.05% of simplices being critical, compared to other datasets, where 0.3-0.6% of simplices are critical. As a result, the change in the topology of the *Dragons Back Ridge* dataset during geometric simplification is less evident.

We also evaluate how the number of critical simplices changes when the mesh is simplified by the Geometric-TT method. In Fig. 4.11, we show the changes in the two larger datasets, i.e., the *Dragons Back Ridge* dataset and the *Moscow Mountain* dataset. The numbers of all types of critical simplices change significantly in both datasets. For the *Dragons Back Ridge* dataset, the numbers of minima, saddles, and maxima increase around 33%. Conversely, for



(a) Dragons Back Ridge (b) Moscow Mountain

Figure 4.11: The change in the number of critical simplices in two tested dataset when they are simplified with the Geometric-TT method.

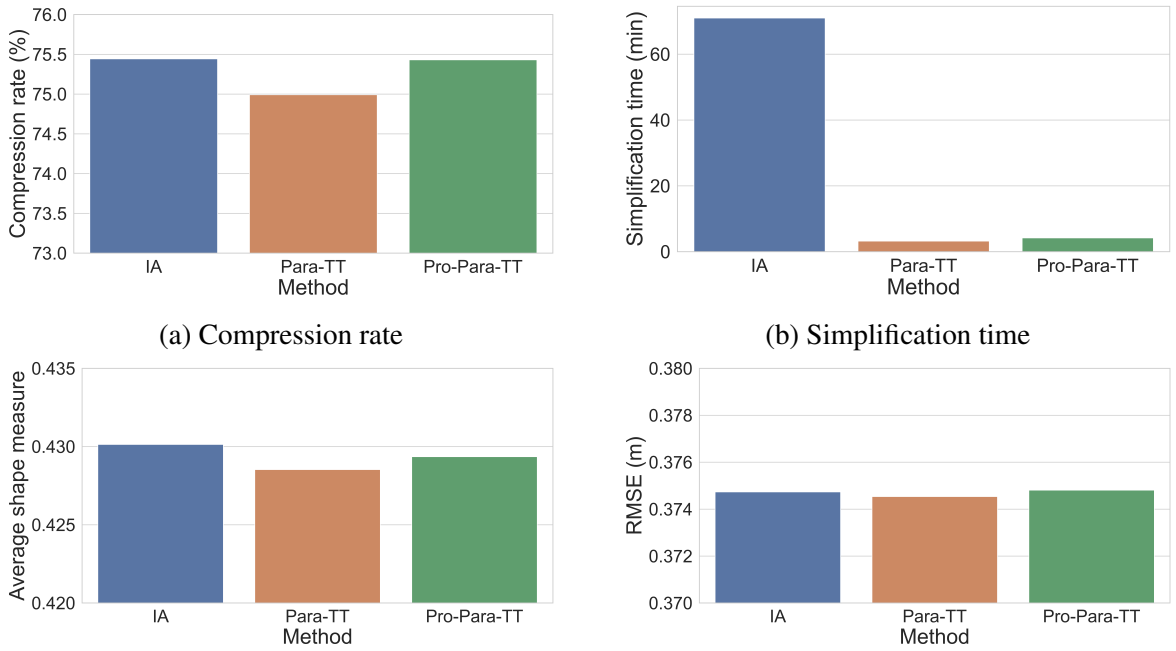
the *Moscow Mountain* dataset, the numbers of minima, saddles, and maxima are reduced by 22%. The changes in other datasets (see Fig. A.15 in Appendix A.6) show a similar pattern as the *Moscow Mountain* dataset. Notably, the *Dragons Back Ridge* dataset presents a unique case among the tested datasets, as it exhibits an increase in the number of critical simplices after a Geometric-TT simplification. This deviation is likely caused by the same factor leading to its low sliced Wasserstein distance. The percentage of critical simplices in the original *Dragons Back Ridge* dataset was significantly lower compared to other datasets, indicating a relatively simple terrain topology. Consequently, the geometric simplification introduces artificial irregularities or distortions to the terrain, resulting in an increase of the number of critical simplices. In contrast, the geometric simplification removes significant features from the terrain in other datasets.

These observations suggest that, when a mesh is simplified with the geometric method, the number of critical simplices in the terrain may either increase or decrease, depending on the terrain topology. In contrast, the number of critical simplices remains constant when a mesh is simplified by a topology-aware simplification.

4.8.4 Progressive strategy for improving output mesh quality

As shown in the above experiments, the simplification on the Terrain trees sometimes produces meshes with higher approximation errors and a lower compression rates compared to the IA-based method, especially when the cost threshold is huge. This difference is mainly caused by the local queues used by the Terrain trees. To improve the geometric quality when the cost threshold is set to a large value, we can use a *progressive* strategy to optimize the simplification on the Terrain trees. Instead of simplifying the mesh with the cost threshold in one round, we can simplify it with several increasing cost thresholds. This optimization helps us to avoid contracting edges with high costs in the early stages of the simplification, which is the major cause of the quality difference between Terrain trees and IA-based methods. We employ a progressive strategy in both the sequential (named *Pro-TT*) and parallel (named *Pro-Para-TT*) simplification methods. In this section, we use the *Great Smokey Mountain* dataset as an example to evaluate the *Pro-Para-TT* method by comparing it with the parallel simplification on the Terrain trees (i.e., Para-TT method) and the IA-based method. The experiments on other datasets show similar results and are shown in Appendix A.6.

In the *Pro-Para-TT* method, we first simplify the mesh in four rounds using cost thresholds set to the first quartile (Q1), second quartile (Q2), third quartile (Q3) of edges costs, and a very large value. In the fifth round, we remove all contractible edges remained in the mesh. Based on the distribution of edge costs, we consider 1000 to be sufficiently large to contract most normal edges while avoiding edges with abnormally high costs which could significantly deteriorate the mesh quality. For the IA-based method and the Para-TT method, all contractible edges are contracted. Fig. 4.12 shows the simplification time and quality of the simplified meshes using



(a) Compression rate

(b) Simplification time

(c) Average shape measure

(d) RMSE of vertical distances

Figure 4.12: The performance and quality evaluation of the parallel simplification on Terrain trees when a progressive optimization is applied to the *Great Smokey Mountains* dataset.

these three different methods.

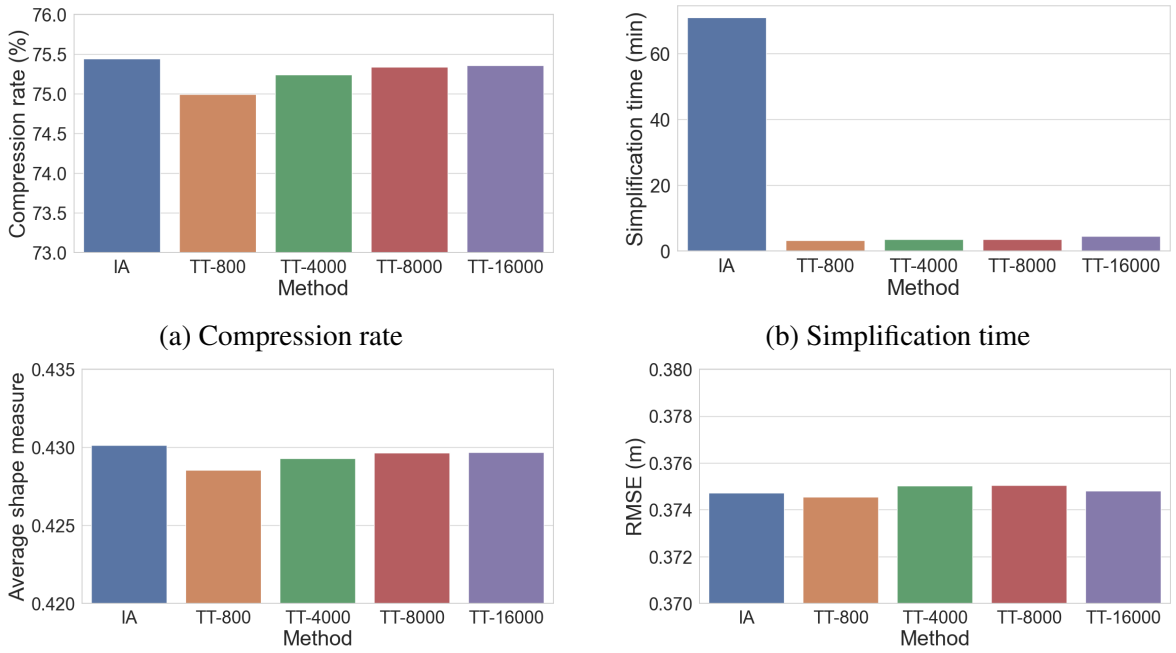
The results show that the difference in compression rate between the Para-TT and the IA-based methods is reduced from 0.6% to 0.01% and the difference in average shape measure is reduced from 0.37% to 0.18% when a *progressive* optimization is used. These results confirm our assumptions that the differences in the compression rate and average shape measure between the two methods are caused by the use of local queues. One interesting finding is that the RMSE of the Para-TT method is smaller than the one of the IA-based method, which indicates that the mesh simplified by the Para-TT method has a better global approximation quality. A possible explanation is that the compression rate of Para-TT method is slightly lower than that of the IA-based method and it is somehow expected to have a better mesh quality when fewer edges are removed. Therefore, it is also expected that the Pro-Para-TT method has a higher RMSE

compared to the Para-TT method as it has a higher compression rate. The RMSE of the Pro-Para-TT method is 0.02% larger than the IA-based method, which is a very small difference. Comparing the timings, we notice that the Pro-Para-TT method requires only 5.9% of the time used by the IA-based method, yet it is 30% slower than the original Para-TT method. This happens since the Pro-Para-TT method has to traverse more times the tree compared to the Para-TT method. Consequently, this method extracts local edge queues and local topological relations more frequently, resulting in slower processing times.

These experiments show that by performing a progressive simplification users can define a trade-off between simplification quality and time requirements. The progressive strategy has proven to effectively narrow down the quality difference between the IA-based method and Para-TT method while requiring slightly more time than the Para-TT method.

4.8.5 Selection of leaf capacity on mesh quality

In Appendix A.5, we evaluate how the leaf capacity affects the computing performance of the simplification on Terrain trees. In this subsection, we evaluate the influence of the leaf capacity on the output mesh quality by using the *Great Smokey Mountains* dataset as an example. The experiments on other datasets show similar results and are displayed in Appendix A.6. We compare only the Para-TT method with the IA-based method as the quality of meshes obtained from Seq-TT and Para-TT methods are similar. Our initial experiment in Appendix A.5 shows that the mesh quality does not change significantly when the leaf capacity is changed within a small range. Therefore, in this experiment, we use several capacity values that are much larger than the optimal range for computing performance. We evaluate the simplification time and mesh



(a) Compression rate

(b) Simplification time

(c) Average shape

(d) RMSE of vertical distances

Figure 4.13: The performance and quality evaluation of the simplification of the *Great Smokey* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds to leaf capacity value used for the Terrain tree generation.

quality measures when all contractible edges are contracted.

Fig. 4.13 shows the simplification time and several quality measures when different leaf capacity values are used for generating the Terrain trees. The numbers on X-axis denote leaf capacity values. The experiments show that when the capacity is 800, the compression rate of the Para-TT method is 0.6% lower than the IA-based method, and this difference is reduced to 0.12% when the capacity value is 16000. Similarly, the difference in the average shape measure changes from 0.37% to 0.1% when the capacity increases from 800 to 16000. When comparing the RMSE based on the vertical distances, the Para-TT method has a smaller error than the IA-based method when the capacity value is 800. However, such a result is obtained with fewer edges contracted by the Para-TT method. Therefore, when the capacity value increases to 4000, the RMSE of the

Para-TT method surpasses that of the IA-based method. The difference between the two methods is reduced from 0.08% to 0.02% when the capacity value continues to increase.

These findings suggest that using a larger leaf capacity enhances both the compression rate and the quality of the output meshes. This happens since a coarser Terrain tree is generated, resulting in simplification results closer to those obtained when using a global queue, such as in the IA-based method. Additionally, larger leaf capacities lead to increased sizes of local queues, thereby reducing the likelihood of contracting higher-cost edges during the early simplification stages. It is noteworthy that very large leaf capacities lead to increased simplification times. For instance, when the capacity value is set to 16000, the simplification requires 39% more time compared to a capacity of 800. This increase is due to the higher costs of extracting auxiliary data structures of cross edges in a leaf block. Nevertheless, even with larger leaf capacity values, the Para-TT method remains significantly faster, consuming only 6.3% of the time required by the IA-based method.

4.9 Concluding remarks

We introduced a new method for simplifying very large triangle meshes representing terrains on a compact data structure, the Terrain tree. Our method extends the strategy defined in [99], which is based on the gradient-aware edge contraction operator, to a global data structure. The proposed method is capable of reducing the resolution of a TIN while preserving the topology of the underlying terrain.

We have experimentally demonstrated how the method based on the Terrain trees can effectively reduce the time and memory requirements of a simplification procedure. Compared to the

IA data structure, which is the most widely used data structure for triangle meshes, Terrain trees achieve similar simplification level in half the time and with only 40% of the memory. Our evaluation also shows that the gradient-aware simplification on the Terrain trees produces meshes with comparable geometric quality as the simplification on the IA data structure. These results prove the scalability and efficiency of our method for processing large-scale triangle meshes. Thanks to the distributed nature of Terrain trees, we also defined a parallel version of the simplification method and implemented it with OpenMP [37]. Comparing the sequential and parallel strategies based on Terrain trees, we observed a further performance increase. The parallel strategy achieved a 12x speedup when using 20 threads while maintaining similar memory requirements.

We discussed how the selection of the leaf capacity values for generating Terrain trees influences the simplification time and quality. Our experiments in Appendix A.5 show that the simplification on Terrain trees has the best time and memory performance when smaller leaf capacities are used for generating the tree, while the simplified meshes have slightly lower quality compared to the IA-based method. In Section 4.8.5, we show that larger leaf capacities lead to better mesh quality but worse computing performances compared to smaller leaf capacities. One optimization strategy to reduce the difference in quality between Terrain trees and the IA-based method is to avoid using a single large cost threshold for the simplification process. Instead, by using a progressive simplification strategy, i.e., a set of increasing cost thresholds, it is possible to improve the final mesh quality at the expense of slightly slower simplification times.

We also designed a new algorithm for updating the Terrain tree after the simplification. This bottom-up method uses 68-77% less time compared to a naive solution that simply rebuilds the tree from scratch. The proposed update algorithm is independent of the gradient-aware simplification algorithm and can be applied to any case in which the mesh encoded in a Terrain tree

is modified.

Topology-aware simplification keeps all critical simplices in a terrain. However, original TIN datasets may be noisy or may contain too many insignificant terrain features. To address this issue, one can remove critical simplices originated from noise and insignificant features in a pre-processing step using the topology simplification method [99] as described in Section 4.2.

The parallel strategy developed here can be easily extended to other topology-aware edge contraction operators, such as the one introduced in [50], since the range of simplices involved in those topology-preserving conditions is the same as for our gradient-aware simplification operator.

Chapter 5: TIN-based sea ice surface topography mapping

5.1 Introduction

This chapter discusses our research work concerning the use of TIN-based surface models to represent sea ice surfaces in polar regions and extract topographic features.

Recent research demonstrates the ability of ICESat-2 data for extracting topography features including surface roughness, pressure ridges, melt ponds, floes, and leads [67]. However, these methods for extracting topographic features are based on profiles along scanning tracks. They cannot be simply extended to two-dimensional feature extraction from ICESat-2 data, given the larger distance between adjacent tracks (3km) compared to the point spacing along the track (0.13m).

In this work, we focus on two-dimensional sea ice topography features, and thus, we consider another data source for sea ice elevation, the airborne laser altimeters. NASA Operation IceBridge (OIB) campaign collects a large volume of sea ice surface measurements by using the Airborne Topographic Mapper (ATM) [104]. The ATM swath, which can reach a width of 200-300 meters across the flight track, offers us the opportunity to extract two-dimensional features from sea ice surfaces that are challenging to obtain from linear profiles [151]. Existing methods for analyzing sea ice topography from LiDAR point clouds usually require converting point clouds into raster grids [151, 197] first. As discussed in Section 1.1, the generation of a TIN

does not require a resampling of the surface and is able to maintain all information derived from the elevation measurements. Since point measurements are distributed irregularly over the study area, using grid-based analysis methods tends to average out some topographic details over the surface. Besides, TINs can include accurate line features and boundaries of regional features, which are beneficial in topographic feature extraction. In the sea ice field, triangle meshes have been used in simulation models [82, 111] and mass change evaluation [181], but they have not been employed in topographic feature study.

In the work presented in this chapter, we introduce methods for topographic analysis on TINs generated from LiDAR points collected from sea ice surfaces. Specifically, we primarily address three key objectives: (1) creating a TIN for an accurate representation of the sea ice surface; (2) developing statistical methods for assessing ATM point density, extracting data missing regions, and calculating region roughness; and (3) extracting important topographic features from sea ice surface, specifically ridges. A draft of the manuscript covering this work is complete and we plan to submit it to IEEE Transactions on Geoscience and Remote Sensing [169]. We also plan to publish the code for all Terrain trees-based topographic analysis methods introduced in this study as an open-source library together with the paper.

The remainder of this chapter is organized as follow. Section 5.2 introduces the information of datasets used in this study. Section 5.3 introduces a strategy to generate TIN-based surface models from OIB ATM point cloud. Section 5.4 introduces how to calculate point density and roughness on TIN-based surface models and how to extracting data missing regions. Section 5.5 introduces an algorithm to extract pressure ridge sails from TIN-based models. Section 5.6 contains the topographic analysis results of the study region and a discussion of how to select the optimal parameters for ridge extraction. Finally, in section 5.7, we summarize our work and

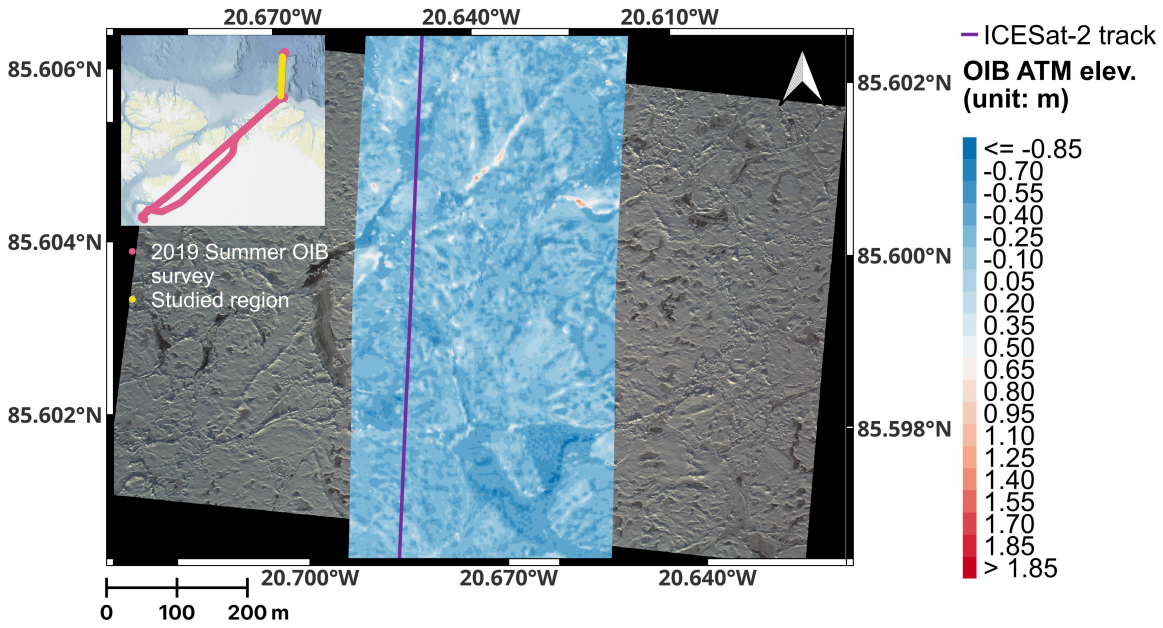


Figure 5.1: Example of the datasets used in this study. The basemap is an image collected by Operation IceBridge (OIB) Continuous Airborne Mapping By Optical Translator (CAMBOT), the mesh layer colored from blue to red is the TIN surface model generated from OIB Airborne Topographic Mapper (ATM) dataset, and the purple path overlaying the mesh layer is the track of ICESat-2. All datasets were collected during 2019 summer. The inset map displays the flight path of 2019 Summer OIB survey on September 9, 2019. The basemap of the inset map is the topographic basemap offered by Arctic SDI [2].

discuss our findings from conducted analysis.

5.2 Data description

From 2009 to 2019, the NASA OIB campaign performed annual surveys of the Arctic region with a set of instruments including laser altimeters, a snow radar, a radar altimeter, and an optical camera. OIB has collected a large volume of sea ice surface measurements. In this work, we use the LiDAR points collected from the OIB Airborne Topographic Mapper (ATM) [104] to generate the triangulated surface models and the OIB Digital Mapping System (DMS) optical imagery for validation of the LiDAR surface returns. The ICESat-2 ATL03 product is also used

for the validation of generated elevation models and extracted surface features. Specifically, the data collected during fall freeze-up (September 2019) are used. In this work, we study a 234 km segment of September 2019 OIB ATM data. Fig. 5.1 shows a subregion of area we study in this study and how different datasets match up. Elevations from all datasets (OIB ATM and ICESat-2 profile elevations) are converted to relative elevations with respect to the Earth Gravitational Model 2008 (EGM2008) geoid [147] before further analysis. In the rest of this section, we describe the datasets we used in greater details.

5.2.1 Operation IceBridge Datasets

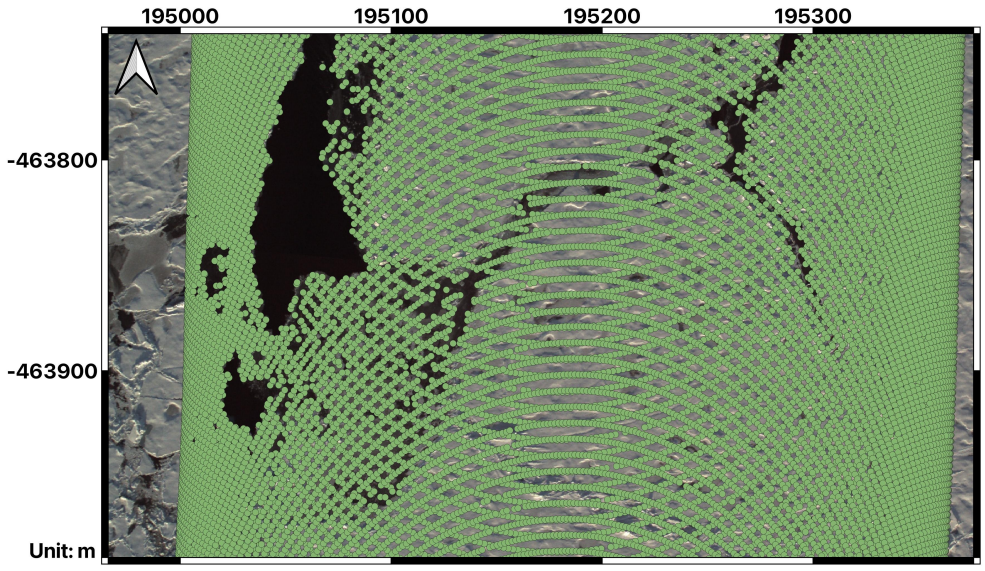
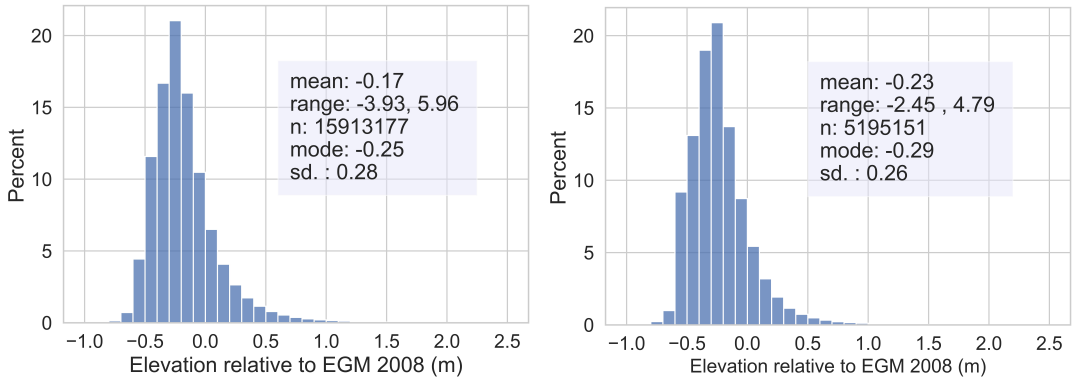


Figure 5.2: OIB ATM point cloud map of a region with open water area. ATM points are displayed as green points in the figure. The basemap layer is the OIB CAMBOT image coincident with the displayed section of ATM point cloud.

The ATM is a conical-scanning laser altimeter that collects a swath of point measurements along the path of the aircraft [104]. Two types of LiDAR data, the wide-scan data [176] and the narrow-scan data [177], are collected during OIB with different ATM instruments. The wide-scan swath is around 400 m wide with 0.1 $pt./m^2$ density [176] and the narrow-scan swath is

around 45 m wide with 0.5 $pt./m^2$ [177] when flown at the nominal altitude of ∼500 m. In this work, the wide-scan data is used due to its larger coverage while the same method can be applied to the narrow-scan data. The wide-scan ATM LiDAR has a wavelength of 532 nm, a laser pulse rate of 5kHz, and a scan width of 22.5° off-nadir [176]. Previous studies show that the low reflectivity over water surface and thin ice leads to laser dropouts in the OIB wide-scan ATM point dataset [106, 179]. As shown in Fig. 5.2, the distribution of OIB wide-scan points in the study area matches with this finding. This phenomenon is less common in narrow-scan ATM data due to the reduced scan angle. The data dropouts lead to regions without enough measurements in the study area and including these regions in the surface model can lead to imprecise extraction of topographic feature ranges. In Section 5.3, we discuss how to address this issue during the generation of triangulated surface models for representing the sea ice surface.



(a) Study region (b) Segment for ridge extraction experiments

Figure 5.3: The elevation distribution of OIB ATM points (bin width = 0.1 m) (a) within the complete study region and (b) within the segment for ridge extraction experiments. All elevations are relative to he Earth Gravitational Model 2008 (EGM2008) geoid [147].

The elevations of ATM points within the study region range from -3.93 m to 5.96 m. The detailed elevation distribution is shown in Fig. 5.3(a).

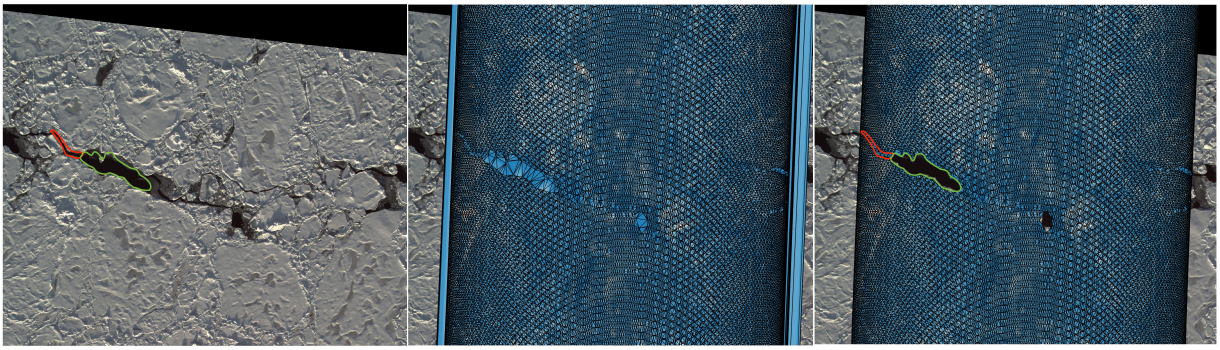
High-resolution images, coincident with the ATM LiDAR data, are collected by the Contin-

uous Airborne Mapping By Optical Translator (CAMBOT) system during the OIB campaign [178]. The equipped camera is designed to provide images marginally wider than the ATM wide-scan swath. Each CAMBOT image covers a ground range of ∼1000 m x ∼650 m, has a spatial resolution of ∼0.2 m, and is both orthorectified and geolocated [178].

5.2.2 ICESat-2 datasets

Since its launch in 2018, ICESat-2 has collected continual measurements over both polar regions with the equipped Advanced Topographic LiDAR Altimeter System (ATLAS) [125]. ATLAS produces 3 beam pairs (each pair has a strong and a weak beam) with a 3 km gap between pairs and a 90 m gap within pairs.

One objective of the OIB 2019 summer campaign is to include ICESat-2 low-latency "race-track" flights, maximizing the chance of coincident ATM and ICESat-2 coverage when the sea ice drift is considered [174]. For the purpose of validating ridge extraction results, we intentionally selected a 72 km segment of the OIB flight which had better coincidence with the ICESat-2 track after a sea ice drift correction was applied. The difference in the collected time between two datasets within the 72 km segment ranges from 47 to 57 minutes. The ATM elevation distribution for the 72 km ridge validation segment is shown in Fig. 5.3(b). We use the elevation profile extracted from ICESat-2 ATL03 product [137] with the University of Maryland-Ridge Detection Algorithm (UMD-RDA) [67] to validate our results on surface modeling and ridge extraction. The extracted ICESat-2 profile has a sampling interval of ∼0.7 m and a vertical precision of ∼0.01 m over flat surfaces. It has been proved that the UMD-RDA method can resolve features as narrow as 5.6 m [53]. The ICESat-2 UMD-RDA elevations of the track segment within the



(a) (b) (c)

Figure 5.4: (a) CAMBOT image of a region with open water areas. (b) Corresponding TIN generated from the OIB ATM data in the same region and (c) the modified TIN after the underlying mesh is processed with α-shape method (when $\alpha = 20$ m). The triangles in both (b) and (c) are from the underlying triangle meshes of generated TINs.

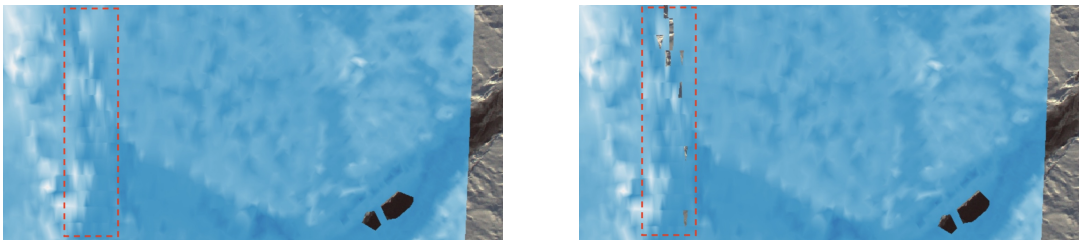
study region vary from -0.82 m to 2.52 m.

5.3 Triangulated surface model

In this section, we will describe the generation of triangulated surface models, i.e., TINs, from ATM point clouds.

As explained in Section 5.2, the elevations of ATM point clouds are subtracted by the EGM 2008 before TIN generation. Besides, all points are projected to NSIDC Sea Ice Polar Stereographic North coordinate system (EPSG: 3411).

The Delaunay triangulation [42] is the most widely-used method to generate a TIN since it ensures the exclusion of elongated triangles in the resulting mesh. Avoiding such triangles is crucial for the accurate interpolation of elevation on the TIN. The Delaunay triangulation of a point set P defines a triangle mesh Σ on the plane, of which the vertices are the points in P and the domain is the convex hull of P [42]. A TIN consists of a mesh Σ generated with Delaunay triangulation, elevations defined on the vertices of Σ, and an interpolating piecewise



(a) $\alpha = 20$ m (b) $\alpha = 15$ m

Figure 5.5: TIN after the underlying mesh is processed through α-shape method with two different α-values: (a) α-shape when $\alpha = 20$ m; and (b) α-shape when $\alpha = 15$ m. The triangle edges are hidden in both figures for a clearer visualization. The red dashed box highlights the region where triangles overlapping sea ice surface are removed from the TIN unintentionally in (b).

linear function [183].

However, since the domain of Σ is always a convex polygon, it may not match the actual domain of the point dataset. This leads to elongated triangles on the boundary of the TIN, potentially causing inaccurate analysis results. We consider such triangles as artifacts produced by the Delaunay triangulation and should be removed from Σ. Additionally, as mentioned in Section 5.2.1, the possible data dropouts over water surface lead to regions without measurements. To address these two issues, the α-shape method [59] is applied to the meshes of generated TINs as a post-processing step.

The α-shape of a set of points P on the plane depends on the value of parameter α. Intuitively, the α-shape is obtained from the Delaunay triangulation Σ of P by using a circular eraser with radius α to remove triangles from Σ while ensuring that no points in P are affected. Smaller α values result in the removal of more triangles from the TIN. Specifically, when $\alpha = \infty$, the α-shape is the convex hull of P, and when $\alpha \to 0+$, the α-shape is the set of points P itself [59].

Using this tool, we can effectively remove triangles formed by widely spaced vertices in a triangle mesh. For simplicity, we denote the mesh after α-shape processing with $\alpha = i$ (unit: meter) as Σ_i. An example of Σ_{20} is presented in Fig. 5.4(c). In comparison with the mesh of

original TIN in Fig. 5.4(b), Fig. 5.4(c) shows that the long triangles on the boundary and most triangles overlapping the open water regions are removed in Σ_{20} (the region highlighted with a green boundary in Fig. 5.4(c)). However, it is worth noting that some triangles overlapping the water surface are not removed in the processed mesh (the region highlighted with a red boundary in Fig. 5.4(c)). This is because the distances among vertices of these triangles are too small for them to be removed with the current α value. When the α value decreases, more such triangles can be removed, but it is possible to mistakenly remove triangles representing the actual sea ice surface (e.g., the holes in the red dashed box in Fig. 5.5(b)). The reason is that the sampling density varies across different areas, and some sea ice regions have sparser points that lead to larger triangles.

To obtain the range of sea ice surface, we remove as many invalid triangles as possible while ensuring that no triangle covering the actual surface is removed. Thus, we conduct experiments with different α values, starting from 15 m and increasing the value until no triangles overlapping the sea ice surface are removed from the mesh. Our experiments show that the smallest α value to avoid removing valid triangles for this dataset is 20 m.

Using the method described above, we generate a TIN from the OIB ATM dataset and process its underlying mesh with the α-shape method using $\alpha = 20$ m. A new TIN Θ, which is the final surface model, is defined on the resulting mesh with the same elevations at vertices. For the sake of clarity, we refer to this method to generate a triangulated surface model as the *OIB ATM-TIN* method, and the generated surface model as the *ATM-TIN model*. We utilize the Computational Geometry Algorithms (CGAL) library [24] for both Delaunay triangulation and α-shape calculations.

The surface model Θ is then compared to the ICESat-2 profile calculated from ICESat-

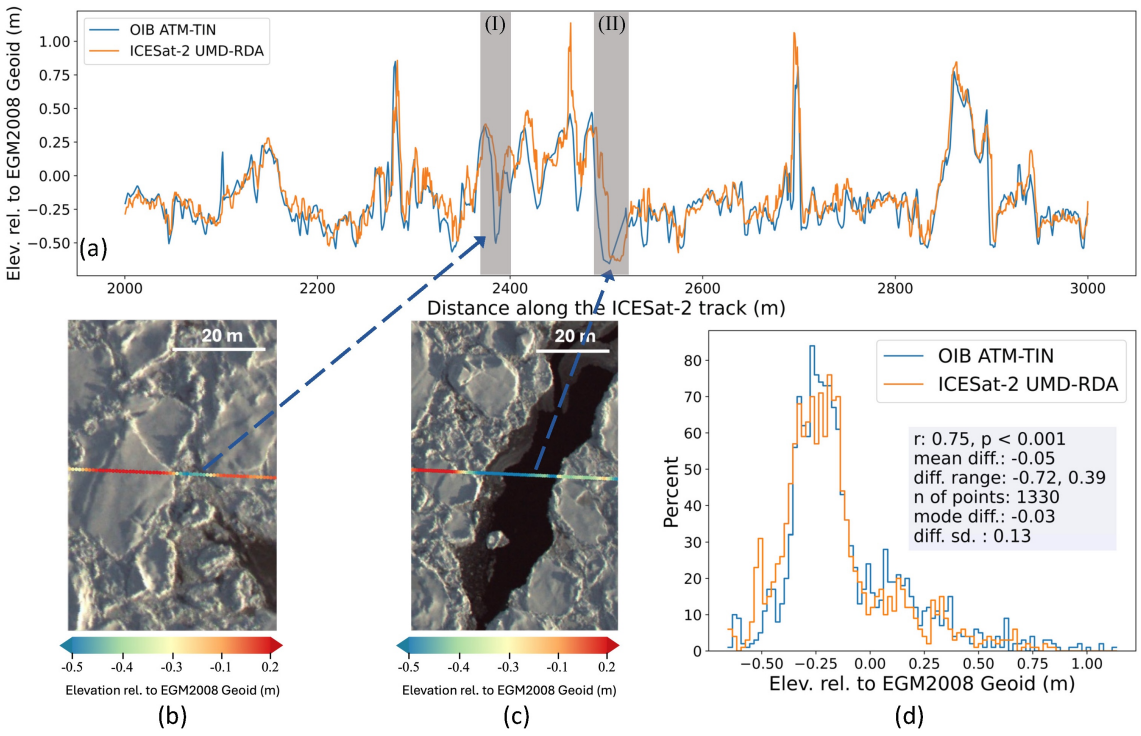


Figure 5.6: Comparison between OIB ATM-TIN model and ICESat-2 University of Maryland-Ridge Detection Algorithm (UMD-RDA) profiles. (a) The profiles from interpolation on OIB ATM-TIN model along ICESat-2 track (blue line) and the ICESat-2 ATL03 UMD-RDA (orange line) along a 1km segment. (b) Example of a small surface feature that cannot be detected from ICESat-2 UMD-RDA result, but can be detected in OIB ATM-TIN profile. (c) Example of an open water region, which leads to data dropouts in OIB ATM. (d) Distributions of elevations from two profiles in the selected segment (bin width = 0.02 m). Points whose elevation measurement is out of the range ($mean-2\times sd.$, $mean+2\times sd.$) in either distribution are excluded as outliers.

2 ATL03 using the UMD-RDA method [67], which we refer to as the *ICESat-2 UMD-RDA* method. In order to compare them, we calculate a profile by interpolating Θ at ICESat-2 UMD-RDA sampling points along the track. Fig. 5.6 displays the elevation profiles of a segment of the ICESat-2 track from both the OIB ATM-TIN and the ICESat-2 UMD-RDA results. The two profiles closely match with only minor elevation differences in some areas. For instance, in segment (I) in Fig. 5.6(a), the OIB ATM-TIN profile exhibits a short, low segment, evidently lower than the corresponding segment in the ICESat-2 UMD-RDA profile. This discrepancy corresponds to a small refrozen melt pond ($\sim$5 m in length) on the sea ice surface. This difference

arises from the higher resolution of OIB ATM data with respect to the ICESat-2 UMD-RDA result. It allows for the detection of small features like old melt ponds and small cracks (even in mélange areas) with OIB ATM-TIN method where ICESat-2 struggles to resolve due to the larger footprint.

Another example of the difference can be observed in segment (II) in Fig. 5.6(a). In the ICESat-2 profile, segment (II) consistently maintains a low elevation, while in the OIB ATM-TIN profile, there is a sharp elevation change in the middle of this segment. This segment corresponds to a lead in the region. The difference between the two profiles is caused by the errors in the OIB ATM result as there is no or very few measurements over open water. In contrast, the ICESat-2 dataset performs well over leads and can even distinguish the elevation differences between thin ice and open water. We further compare the elevation distributions from two methods in Fig. 5.6(d). We focus on sampling points which have elevation observations falling within the range defined by the mean elevation plus or minus two times the standard deviation in both OIB ATM-TIN and ICESat-2 UMD-RDA results. The results from two methods are highly correlated (with a correlation coefficient $r = 0.75$). There are still discrepancies between two profiles, which is potentially caused by the two factors discussed above. Since the profile from the OIB ATM-TIN model is obtained through interpolation while the profile from ICESat-2 UMD-RDA is obtained from direct measurements, a difference between the two profiles is expected.

The primary challenge in using TINs to represent surfaces is the high time and storage requirements during processing. Therefore, we employ Terrain trees [72], which is introduced in Chapter 3, to store the generated TINs as it is the most efficient data structures for TINs and can significantly reduce the storage and time costs for handling TINs. It is worth noting that, while Terrain trees are designed for TINs, their spatial subdivision also facilitates operations on

points, such as point query and range query, thanks to the inclusion of a spatial index for efficient coordinate searching.

The *Terrain Trees library (TTL)* [73], which is also introduced in Chapter 3, contains a variety of terrain analysis methods. However, the existing functionalities of TTL were not originally tailored for sea ice studies. In this work, we extend TTL's capabilities by modifying existing functions and implementing new algorithms, based on the operators supported within TTL.

5.4 Topographic statistics calculation

In this section, we introduce a few topographic statistical measures we study in this work and their computation from TINs. In subsection 5.4.1, we introduce measures describing the OIB ATM data coverage and point density. Subsection 5.4.2 presents the calculation of sea ice surface roughness at different scales from a TIN.

5.4.1 Regional data statistics

The distribution of ATM point clouds is not uniform across all locations. This variation arises from both instrument limitations and surface characteristics in the region. As discussed in Section 5.2.1, there are dropouts over water surface and the point density in the middle of the swath is lower than the one at both sides of the swath. To better quantify and visualize the ATM point density difference, we calculate the point density within a 10 m search radius for each point in the dataset.

In the following, we denote the study ATM dataset as P. The *expected coverage range* (CR) is defined as the designed scan range of P. Given a point p in P and a 10 m circle C_p centered at

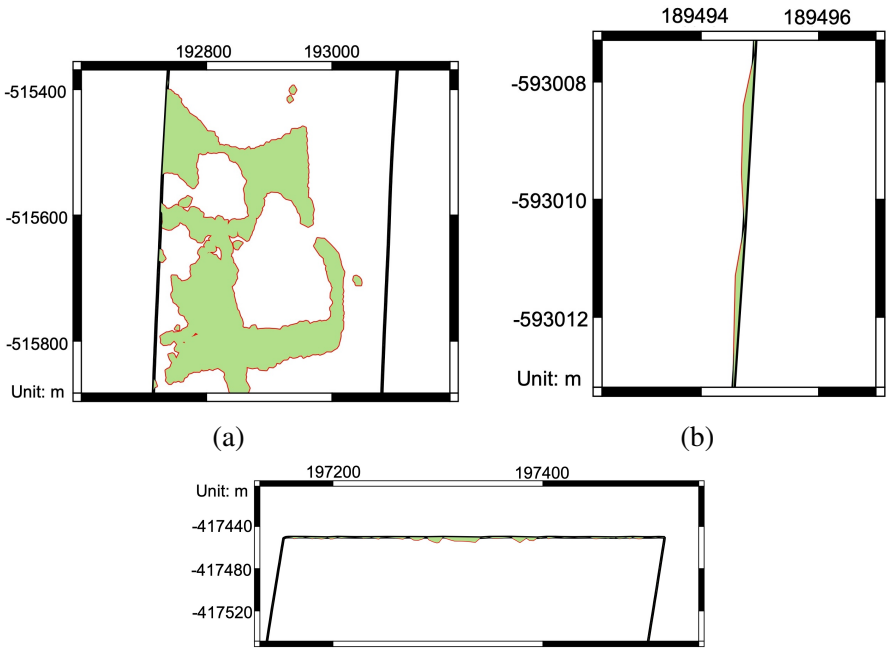


Figure 5.7: Possible cases of extracted invalid regions by comparing Σ_{20} and Σ_{25000}. The difference between them are displayed as green regions. The black lines in the map are the boundary of Σ_{25000}. The red lines in the map are the boundary of Σ_{20}. As discussed in Section 5.4.1, (a) shows a case in which the regions of difference indicate regions without measurements, while (b - c) show cases where the regions of difference are caused by the artifacts introduced during the Delaunay triangulation. Σ_i: the mesh after α-shape processing with $\alpha = i$ (unit: meter).

p, the density d_p at p is calculated as the ratio between the number of points N_p in circle C_p and the area of the intersection between CR and C_p:

$$d_p = N_p / A_{CR \cap C_p} \tag{5.1}$$

Similarly to the generation of surface model, the estimation of the expected coverage of P involves generating a mesh Σ through a Delaunay triangulation from P and processing the mesh by using the α-shape method. However, it is not suitable to use the same α value used in Section 5.3, which we denote as α_S in the following. When estimating the expected coverage, we should include also regions with data dropouts and we should only remove long, spurious

triangles along the boundary of Σ, which are artifacts produced during the Delaunay triangulation process. Using α_S in the α-shape method produces a more refined α-shape that closely follows the density and distribution of the input points, removing regions with data dropouts. From the density calculation result (shown in Section 5.6.1), we can observe that most of the study area has comparable point density, and, thus, most triangles in the generated mesh have similar sizes. Data dropouts, on the other hand, lead to lower density in some regions, and consequently, larger mesh triangles. Such triangles are usually smaller than the boundary triangles caused by artifacts of the Delaunay triangulation process. Therefore, we use another α value α_L, which is greater than α_S, in the α-shape method so that the processed mesh is a coarser approximation of OIB ATM point coverage that excludes boundary artifacts but retains large triangles caused by data dropouts.

The selection of the value of α_L depends on the extent of study area and the size of largest water surface region within it. If selected value of α_L is too small, it may result in unintentional removal of water surface regions. Conversely, if selected value of α_L is too large, it is likely to overestimate the coverage and retain some boundary artifacts.

To find the suitable value, we conduct experiments using α values starting from 10,000 m and increasing the value by 1,000 m each time. The resulting α-shape is compared to corresponding OIB CAMBOT images to ensure that no triangle covering water surface is removed. Ultimately, we determine to use 25,000 m as the value of α_L, and thus, the range of expected ATM coverage CR is the domain of Σ_{25000}.

Another challenge in the density computation is that the operation to find all points within C_p from P is time-consuming, especially when this operation is repeated for all points in the point cloud. By using a Terrain tree T, the problem transforms into finding all blocks in the

spatial decomposition of underlying T that intersect C_p and checking points indexed within those blocks. When the radius of C_p is much smaller than the range of P, the computation time is significantly reduced. Additionally, computing the densities at different points in parallel helps accelerate the overall computation process.

In addition to the density map, we are also interested in the range and distribution of regions without measurements in the OIB ATM dataset. Extracting those regions not only helps us to understand the data quality of OIB ATM, but also helps to validate the assumption that dropouts are caused by the water surface. To extract such regions, we compare the two α-shapes generated with two different α values, α_L and α_S, i.e., Σ_{25000} and Σ_{20}. As discussed earlier, Σ_{25000} contains internal triangles caused by data dropouts, while Σ_{20} does not. Regions that are within Σ_{25000} but are not within Σ_{20} are considered as *invalid regions*. Thus, we expect these invalid regions are regions caused by data dropouts, or regions without measurements.

When comparing the extracted invalid regions with the CAMBOT images, we find that most of them are regions without measurements, with only a few exceptions. Fig. 5.7(a) shows the most common case, in which the invalid regions are formed by large triangles caused by distant points. Such regions are extracted because they are regions without measurements. The exceptions happen at the boundary of Σ_{25000}. During the selection of the value of α_L, we try to remove most large triangles caused by the artifacts, but we have to avoid removing internal triangles caused by data dropouts. Therefore, some small triangles caused by artifacts on the boundary remain in Σ_{25000} if they are smaller than triangles caused by data dropouts. Such triangles are removed together with triangles caused by data dropouts when $\alpha_S = 20$ is used to generate the α-shape. Figs. 5.7(b - c) depict such cases where the extracted invalid regions do not correspond to regions without measurements, but are boundary regions caused by artifacts.

As we can see, they are either at the sides of the swath (as shown in Fig. 5.7(b)) or at the two ends of the study area (as shown in Fig. 5.7(c)). Such cases are even more common when the flight path is not straight since the difference between the convex polygon of *P* and the expected data coverage is more significant and more triangles on the boundary are caused by artifacts.

In this work, we are only interested in regions caused by unexpected data dropouts, i.e., cases like Fig. 5.7(a). We refer to a region as an *internal* region if it is not on the boundary of Σ_{25000}. Otherwise, it is a *boundary* region. Since the artifacts only happen on the boundary of Σ_{25000} and are usually small regions, we calculate the minimum area A_{min} of internal regions and remove all boundary regions smaller than A_{min}. While triangles at the sides of swath (case in Fig. 5.7(b)) can be completely excluded in this way, triangles at the ends of dataset (case in Fig. 5.7(c)) are usually longer and larger. We visually check a boundary region when all of its vertices have a point density higher than the modal point density in the area, as it suggests that the region is not located in areas with sparse measurements and is likely caused by artifacts. Following this strategy, 12 regions at the ends of study region are checked and removed, totaling an area of 529.46 m^2.

All the operations on TINs are implemented in the Terrain trees library [73] and geometry operations like computing the intersection between a circle and CR and computing the difference between two α-shapes are implemented with Shapely [164] and GeoPandas [84] libraries.

5.4.2 Surface roughness

Roughness is an important sea ice topography feature extensively studied in previous research [14, 67]. The knowledge of sea ice roughness is crucial for understanding snow distribu-

tion [97], parameterizing momentum transfer to the ocean [126], processing of remote sensing data collected from sea ice surfaces [109, 110], and evaluating sea ice surface trafficability [39]. Generally, higher sea ice roughness indicates more complex topographic features, and thus, lower trafficability. Sea ice roughness is influenced by ice deformation and, in return, has been used to identify topographic features such as ridges and leads [139] or to determine level sea ice [54].

There are different definitions of roughness depending on the scale studied and on the data format. In most sea ice studies, roughness is defined as the standard deviation of elevations, but the scales and ranges of elevations vary. For instance, in [67], roughness is defined as the standard deviation of sea ice heights within 25-km-long segments. In [14], roughness is the standard deviation of elevations for each scan line measured by the laser scanner, or the standard deviation of a five-data-point moving window for the single-beam altimeter. In [101], roughness is calculated as the standard deviation of elevation measurements within a 1.1 km resolution pixel.

In this work, we define the roughness at a point p as the standard deviation of elevations of points within a certain distance from p, i.e., within a circle C_p centered at p. To understand how different scales affect the interpretation of roughness results, we compute roughness with different circle radii γ. The circle is denoted as $C_{p,\gamma}$. Similar to the computation of density, the main challenge in roughness computation is to efficiently find points within $C_{p,\gamma}$. We employ both the spatial subdivision of Terrain trees and a parallel computing strategy to expedite the computation.

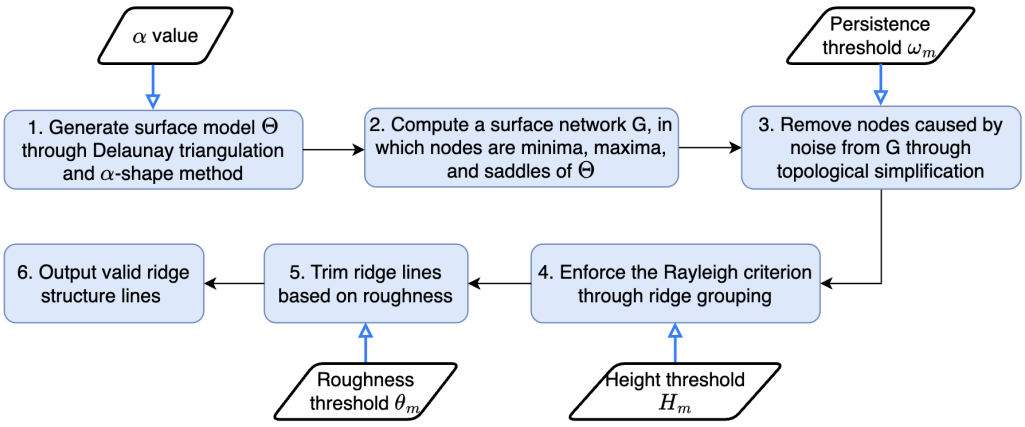


Figure 5.8: The flowchart of steps to extract ridge structures from preprocessed OIB ATM points by using TIN-based surface models.

5.5 Pressure ridges extraction

The collisions between ice floes create pressure ridges. In this work, we focus on the elevated part of a pressure ridge, i.e., the *ridge sail*. The part of a ridge below the water level, named *ridge keel*, cannot directly be extracted from a surface model, and thus, is not discussed in this work. Ridge sails, acting as the major contributor to the sea ice roughness, are important factors to be considered in trafficability analysis [39] and in atmosphere-ice-ocean momentum transfer models [126]. Several ridge features are studied and utilized in other applications. Airborne electromagnetic thickness measurements have a high uncertainty over ridged ice, which makes the mapping of ridges an important prerequisites for applications using electromagnetic data [111]. Ridge sail density is proved to be a good indicator of degree of deformation [154]. The geometry of ridge sails (heights and widths), as well as the ridge density, are necessary for atmospheric drag parameterizations [184].

Inspired by previous studies, we propose here a new method to extract the structure lines of a ridge, including the line along the raised part of a ridge and the steepest lines going down

the ridge peak, from TIN models generated from ATM points. This approach allows for the extraction of ridge structures in a three-dimensional context. Unlike previous methods, which relied on ridge detection from laser profiles or image-based analysis, our approach focuses on the extraction of ridge structures in a three-dimensional context. Different from the method proposed by Petty et al. [151], our method does not require converting the ATM points to grids, which allows us to better trace the extents of ridges. Besides, the input parameters of our method allow for adjusting the range of ridge structures to be extracted, which provides more flexibility in different sea ice applications, such as momentum transfer model, environmental monitoring, and trafficability analysis.

The method relies on the surface topology, which is defined by using discrete Morse theory [76], which is reviewed in Section 2.1.2.2. As described in Fig. 5.8, the process consists of six steps. The first step, the generation of surface model, has been described in Section 5.3. The other steps are implemented on top of the Terrain trees library. Among them, surface topology computation and topological simplification (step 2 and 3 in Fig. 5.8) are introduced in Section 3.5 and Section 4.2, respectively. Both operators are already available in the Terrain trees library. The rest, step 4 - 6, are implemented based on the available operations in the Terrain trees library.

In subsection 5.5.1, we discuss how topological features are related to sea ice ridges. Due to the noise in point cloud data and the variability in sea ice surface, it is necessary to simplify the surface topology so that only significant features are kept. In subsection 5.5.2, we briefly review the topological simplification method and explain how it is applied to the case of a sea ice surface (i.e., step 3 in Fig. 5.8). With the topological method defined in discrete Morse theory, we are able to extract potential ridge structures. Such potential ridge structures are further processed within steps 4 and 5 in order to better describe the range of a ridge. These two steps are described

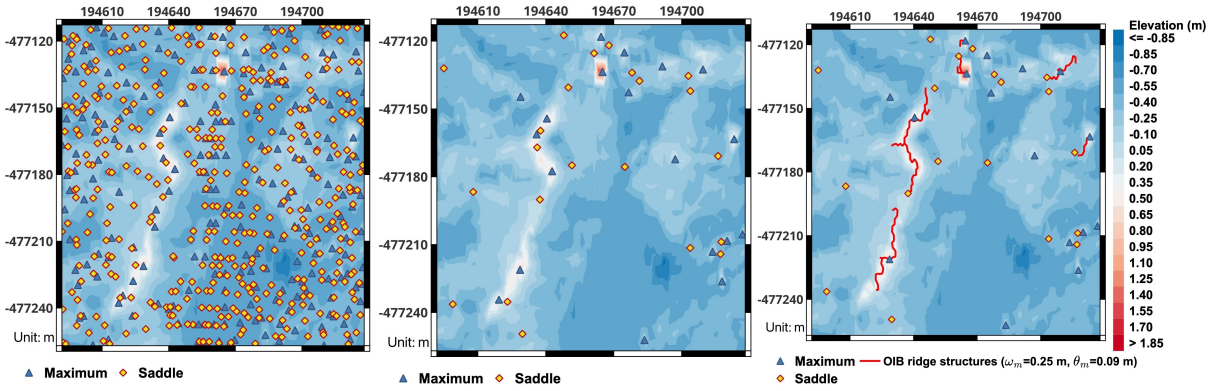
in subsection 5.5.3.

In this work, we focus on a total of 72 km long segment from the study area which matches the best with the ICESat-2 data after drift correction. The modal elevation within each ∼24 km segment is considered as the local level sea ice elevation. Similar to previous studies [53,54], we define the height above the local level ice surface as the *surface elevation anomaly* and denote it as H_A. All LiDAR measurements are subtracted by the local level sea ice elevation to obtain H_A.

5.5.1 Surface topology and ridge structure

We define a *ridge structure* as the subset of geometries of the arcs from a maximum to connected saddles such that these parts are not on the flat level sea ice. A *ridge peak* is a local maximum on the surface whose elevation anomaly H_A is higher than a height threshold H_m. Following previous studies on ridges [54, 67, 182], H_m is set to 0.6 m in this work. Recall that a maximum, as discussed in Section 2.1.2.2, is a critical triangle in a TIN Θ and is a node in the surface network *G* computed from the topology of Θ. An arc connects a maximum with a saddle and one saddle must be connected to exactly two maxima by definition.

Given a maximum *M*, there can be multiple arcs starting from *M*. The geometries of those arcs are potential ridge structure lines. But the existence of noise and small features give rise to spurious topological features. One example of extracted maxima and saddles in *G* is shown in Fig. 5.9(a). The original surface network calculated from the TIN includes an overly abundant nodes even in a small region (206 maxima and 423 saddles in an $140m \times 140m$ region). Thus, a preprocessing step to remove noise and critical points caused by natural elevation fluctuations is a must.

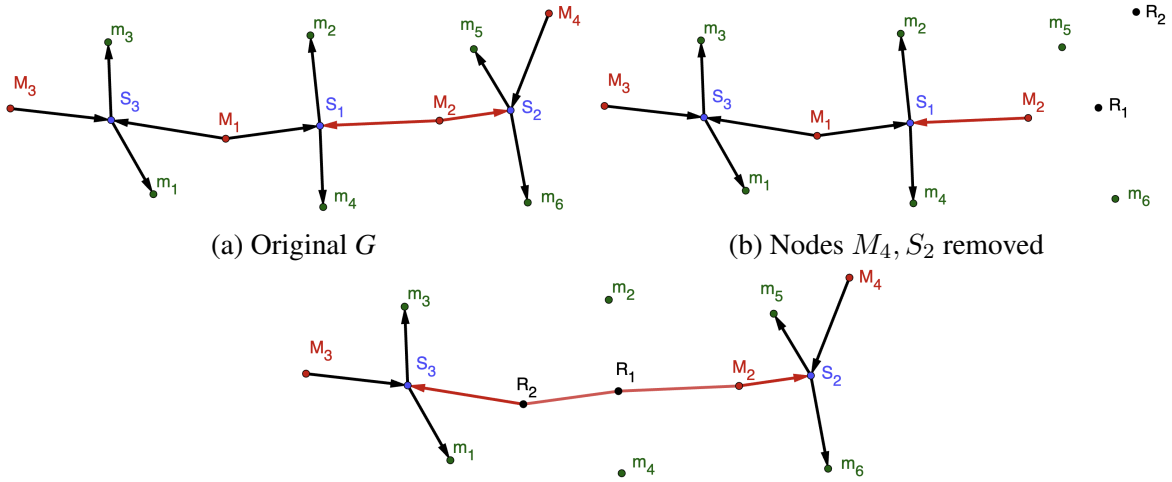


(a) (b) (c)

Figure 5.9: Maps of maxima and saddles (a) in the original Forman gradient field calculated from the TIN based on discrete Morse theory [76]; (b) after the topological simplification (ω_m = 0.25 m); and (c) after the ridge grouping and roughness filtering (θ_m = 0.09 m) steps. The red lines in (c) are final ridge structures extracted with the proposed method.

5.5.2 Topological simplification

To remove noise and insignificant features from sea ice surface, a topological simplification, as introduced in Section 4.2, is applied to the surface network G. Recall that there are boundary and internal critical maxima/minima, based on whether they have more than one adjacent saddles in G. The removals of boundary and internal critical minima and maxima have different effects on the ridge extraction result. Assuming maximum M_2 is a valid ridge peak, the arcs starting from M_2, which are the initial structure of ridge peaked at M_2, are highlighted in red in Figs. 5.10 (a - c). In both Fig. 5.10(b) and (c), a saddle connected to M_2 is removed, together with the other maximum connected to this saddle. When the removed maximum is a boundary maximum (M_4 in Fig. 5.10(a)), the structure is reduced after the cancellation since arc $M_2 \rightarrow S_2$ was removed. On the contrary, when the removed maximum is an internal maximum (M_1 in Fig. 5.10(a)), the structure grows larger since the geometry of arc $M_2 \rightarrow S_3$ consists of the geometries of three arcs.



(a) Original G

(b) Nodes M_4, S_2 removed

(c) Nodes M_1, S_1 removed

Figure 5.10: (Same figure as Fig. 4.1 in Section 4.2.) Example of a maximum-saddle-maximum cancellation operation on a surface network G. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines highlight arcs starting from maximum M_2. Note that the regular points are not a part of the surface network and are just displayed for illustration purpose. (a) Original surface network; (b) surface network when a boundary maximum M_4 and a saddle S_2 adjacent to it are removed; (c) surface network when an internal maximum M_1 and a saddle S_1 adjacent to it are removed.

As introduced in Section 4.2, whether an arc in a surface network G is removed or not depends on the *persistence* ω of the pair of nodes connected by this arc. A topological simplification process removes all arcs with ω lower than a threshold ω_m. Fig. 5.9(b) shows the result of topological simplification when $\omega_m = 0.25$ m. We observe that most maxima and saddles in flat regions are removed while maxima with evident elevation difference from the neighborhood and saddles separating those maxima are kept. In Section 5.6.3, we further discuss how the selection of persistence threshold ω_m affects the ridge extraction result.

5.5.3 Ridge structure extraction

After topological simplification, the potential ridge structure of a ridge can be extracted as the collections of geometries of arcs connected to a maximum M. A ridge can contain several maxima (peaks). A common rule to decide if they should be considered as the same ridge is the Rayleigh criterion [121]: the local maximum is at least twice as high as the neighboring minima on both sides to separate it from adjacent maxima. While the original criterion is defined for profile, it can be extended to ridge extraction from a surface by considering two maxima connected to the same saddle in the surface network G as adjacent maxima. Therefore, for a ridge to be considered as a separate ridge, we require that its maximum is at least twice as high as all the saddles connected to it. If both maxima connected the same saddle do not satisfy this criterion, the ridge structure of the lower maximum is merged to that of the higher maximum. Similarly to topological simplification, the merging can be handled through cancellation operator. We call the step of enforcing the Rayleign criterion *ridge grouping*.

Given a saddle S, and two maxima M_1 and M_2 adjacent to S in G, M_1 and S can be removed through cancellation if and only if:

1. the elevation anomaly H_A of M_2 is higher than that of M_1;
2. H_A of M_2 is higher than a surface elevation anomaly threshold H_m (0.6 m is used as mentioned in subsection 5.5.1);
3. H_A difference between M_2 and S is less than half of the H_A of M_2;
4. M_1 is not a boundary maximum.

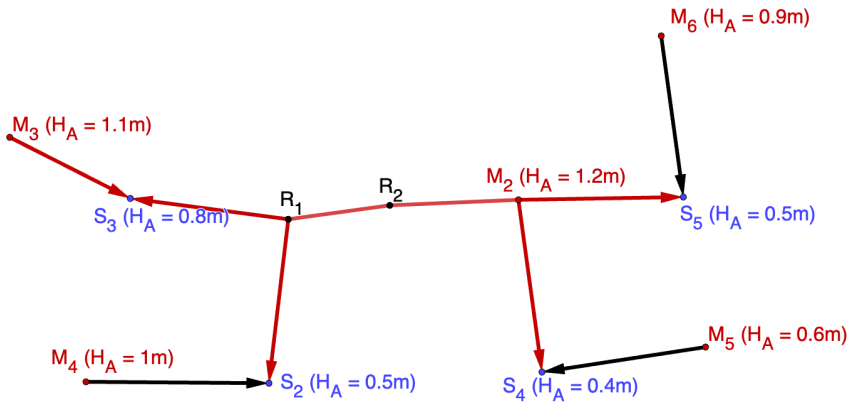


Figure 5.11: Example of ridge grouping. Red points with M- prefix are maxima, blue points with S- prefix are saddles, green points with m- prefix are minima, and black points with R- prefix are regular vertices, edges, and triangles. The lines are the arcs connecting those nodes in G, which indicate the gradient paths in the Forman gradient field. The arrows of lines indicate the directions of arrows along the paths. The red lines are considered the ridge structure of maximum M2. Note that other regular points along the gradient paths are not shown in this figure. The elevation anomaly H_A of a node is the anomaly of the highest vertex of the corresponding geometry. With the given anomalies in the graph, the ridge structures of M_3 should be merged to that of M_2 according to the discussion in Section 5.5.3.

All triplets $\{M_1, S, M_2\}$ are sorted in descending order with respect to the elevation anomaly H_A of the highest maximum in the triplet. In another words, maxima adjacent to the highest maximum in the study region are checked first to see if they should be removed. Condition (4) above is necessary to avoid a loss of ridge structures. For instance, in the case in Fig. 5.11, when M_3 and S_3 are removed, the ridge structure of M_3 ($M_3 \rightarrow S_3$) is discarded instead of being merged to the structure of M_2. In the ridge grouping step, we avoid removing a boundary maximum to maintain the completeness of ridge structure. Instead, we add the ridge structure of a boundary maximum like M_3 to the structure of its adjacent maximum M_2 when M_3 only satisfies conditions (1-3).

After the grouping step, for each ridge peak with $H_A \geq 0.6$ m, we can extract its ridge structure. However, such a structure tends to be larger than the real range of a ridge. This is because the geometry of an arc extends as long as the elevation decreases or stays the same, and,

thus, parts of such geometry can be in flat regions. Such parts do not belong to ridges since the elevation is expected to change sharply near ridge structure lines. Thus, we use a roughness filter to eliminate such parts of arcs as flat regions have low roughness. From the experiments on sea ice roughness of different scales (see Section 5.6.2), we find that the ridged regions match the best with higher 5 m roughness areas. Therefore, we check the geometry of each arc starting from the maximum, and terminate the chain once all vertices of a triangle have a 5 m roughness lower than a predefined threshold θ_m. In Section 5.6.3, we discuss how the selection of θ_m affects the ridge extraction results.

Fig. 5.9(c) shows the result when the ridge grouping step and the roughness filter have been applied to the result in Fig. 5.9(b) when $\theta_m = 0.09$ m. We can see that a few peaks along the higher elevation regions are removed in Fig. 5.9(c), and, thus, the final ridge structure consists of longer paths and better describes the ridge range.

5.6 Results

In this section, we show the topographic features we have extracted with the proposed methods and the statistical analysis of the extracted features. In subsection 5.6.1, we show the density statistics of study region. The surface roughness calculated with different measure scales is shown in subsection 5.6.2. In subsection 5.6.3, we discuss the parameter selection for the proposed ridge extraction method. The result obtained with optimal parameters is evaluated and discussed in subsection 5.6.4.

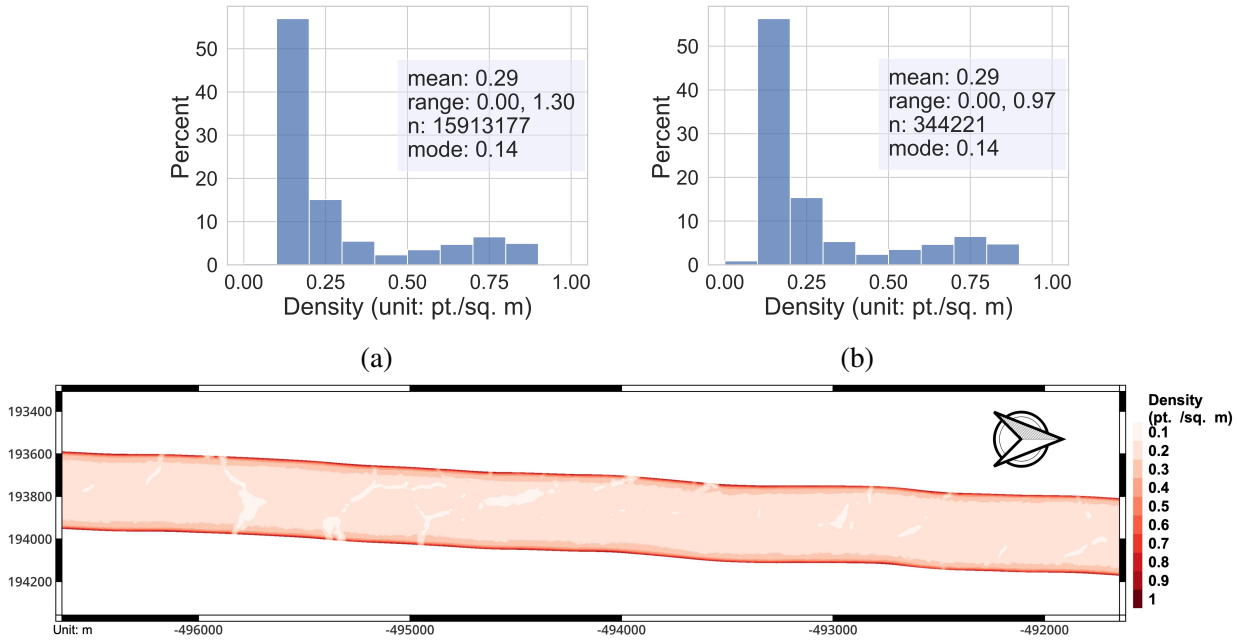


Figure 5.12: (a) Point density distribution within the study area (bin width = 0.05 pt/m^2). (b) Point density distribution (bin width = 0.1 pt/m^2) and (c) density map of a 5 km segment in the study area. The point density is calculated within a 10 m search radius for each point in the dataset.

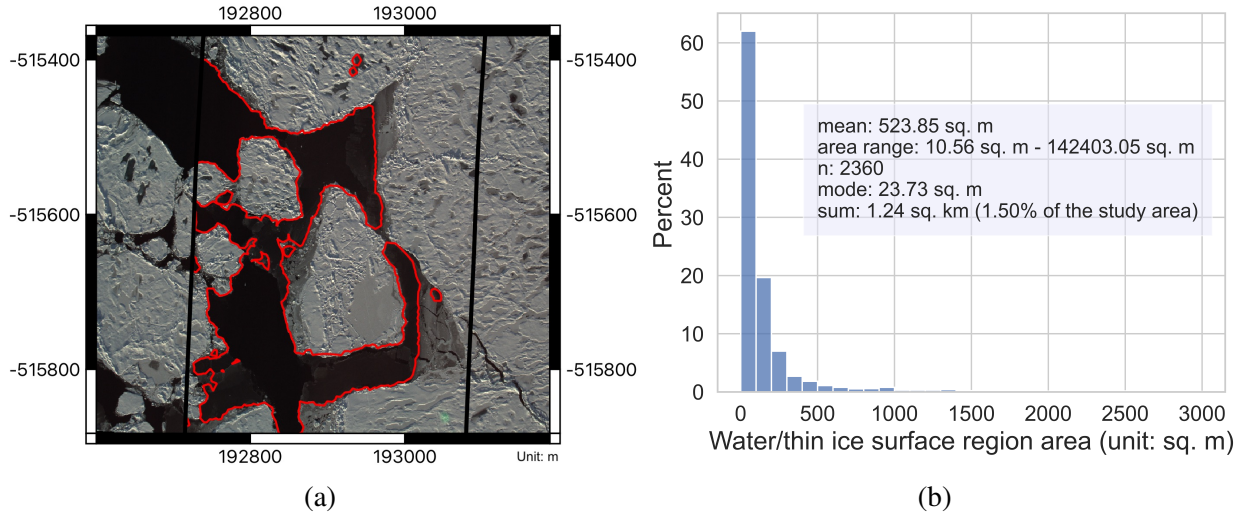
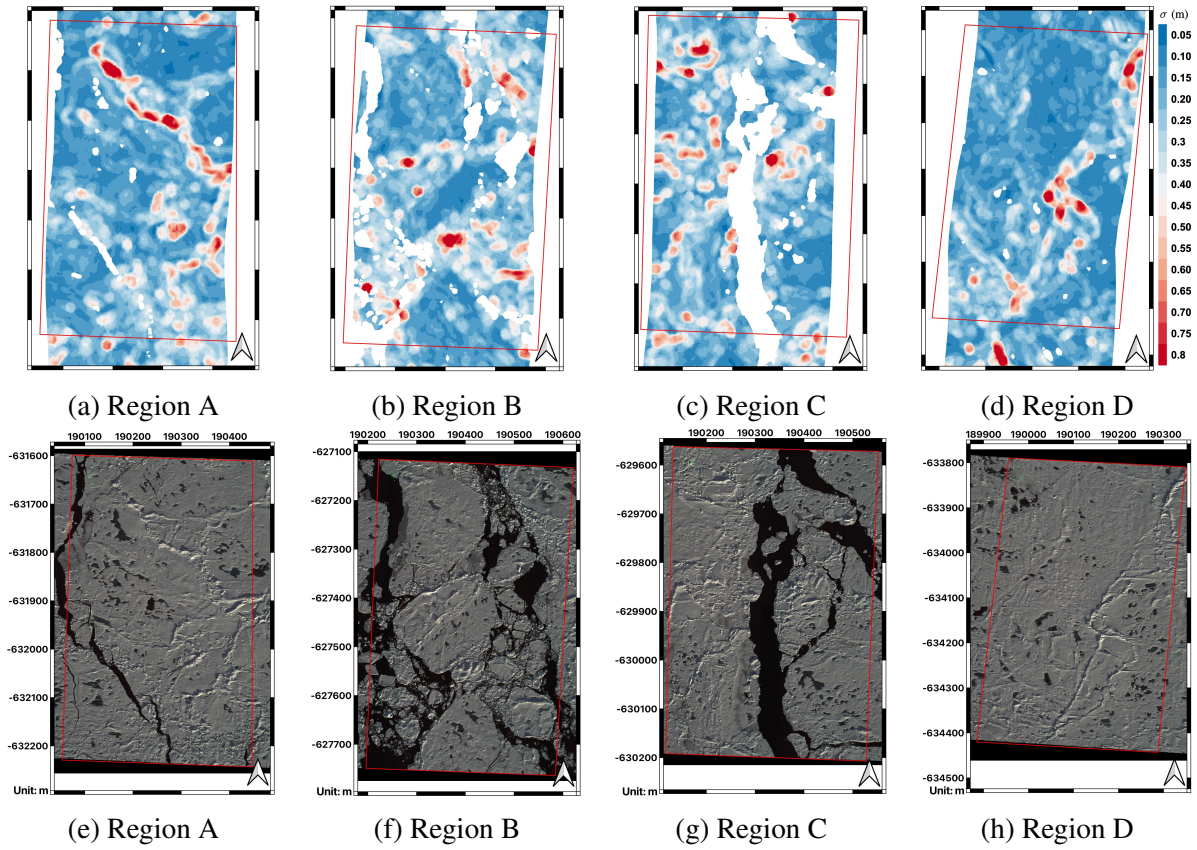


Figure 5.13: (a) An example of the extracted regions without measurements, the black boundary indicates the range of the study OIB ATM data and the red polygons indicate the regions without measurements, which is extracted by comparing Σ_{20} and Σ_{25000}. (b) The distribution of extracted water/thin ice surface regions within the study region. Σ_i: the mesh after α-shape processing with $\alpha = i$ (unit: meter).



(a) Region A (b) Region B (c) Region C (d) Region D

(e) Region A (f) Region B (g) Region C (h) Region D

Figure 5.14: (a - d) The 10 m roughness maps of four selected regions. (e - h) The corresponding images of the four regions. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image.

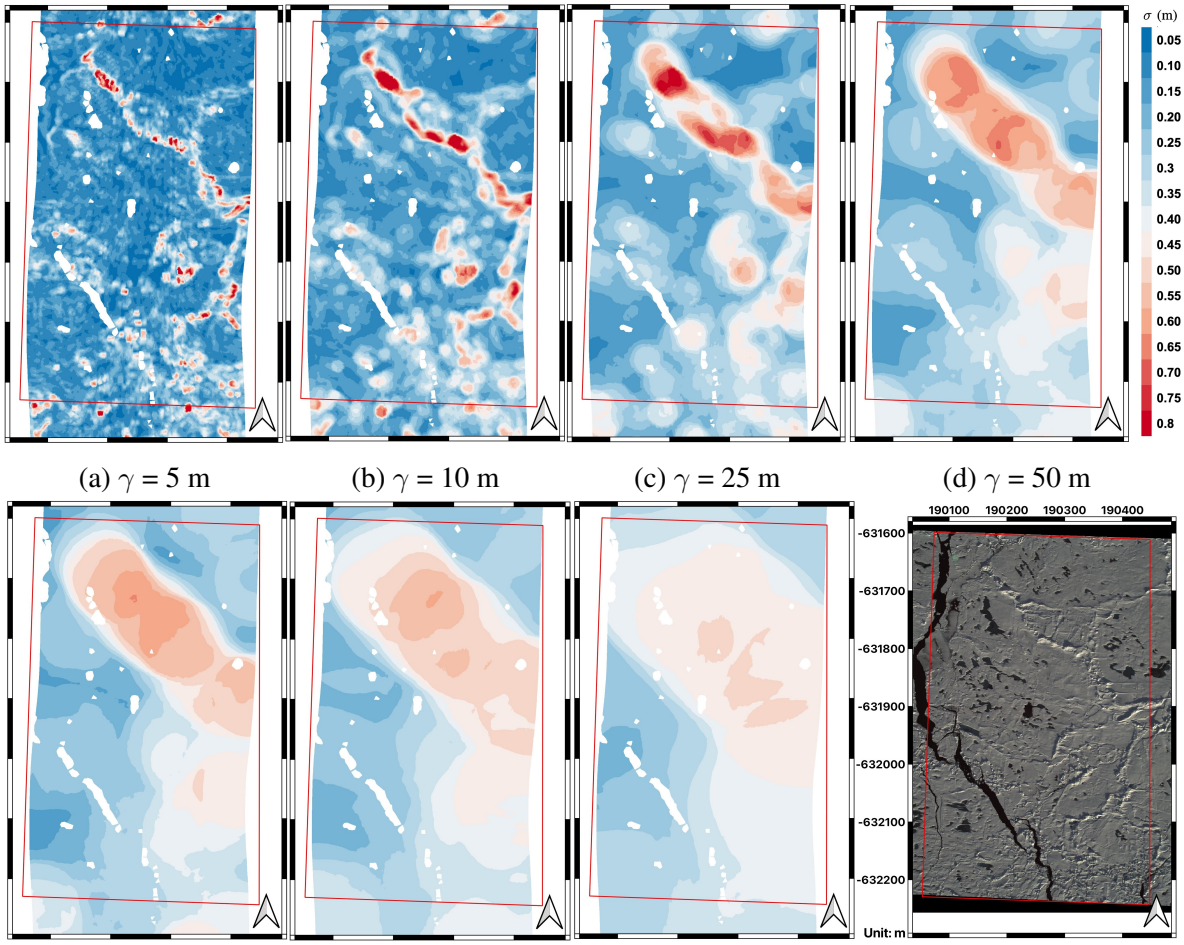
5.6.1 OIB ATM point density

The *average point density* within the study region is calculated as the number of points divided by the area of the expected ATM coverage, i.e., Σ_{25000}. The average point density within the study area is 0.19 $pt./m^2$, nearly twice the planned OIB ATM wide-scan point density of 0.1 $pt./m^2$. As discussed in Section 5.4.1, we also calculate the point density at sampling points within a 10 m search radius. Fig. 5.12(a) shows the distribution of point density at all points in ATM OIB point cloud. It is noteworthy that the average point density mentioned earlier is different from the mean point density in Fig. 5.12(a), which is the average point density of densities

at all sampling points. There are less than 0.003% points with a density lower than 0.1 $pt./m^2$ within the study region. Most points have a density between 0.1-0.15 $pt./m^2$. The distribution is notably right-skewed, with a concentration of lower values with a long tail extending towards higher densities. There is also a noticeable secondary peak around 0.75-0.85 $pt./m^2$.

To better understand the rationale behind this distribution, we visualize the density by assigning density values to the vertices of the TIN and coloring the triangles based on a linear interpolation of the density values at vertices. A section of the generated map is shown in Fig. 5.12(c) and the density distribution within the section is shown in Fig. 5.12(b). Overall, the point density within the study region follows a specific pattern. Point density increases towards the two sides of the swath, corresponding to the second peak around 0.75-0.85 $pt./m^2$. Meanwhile in the middle of the swath, densities typically range from 0.1 to 0.2 $pt./m^2$. This explains the first peak in the distribution, where most points have a density between 0.1-0.15 $pt./m^2$. We also notice a few regions with evidently lower point density compared to the surroundings. As discussed in Section 5.2.1, this is likely due to the data dropout caused by low reflectivity over water surface regions.

To validate this assumption, we identify regions without measurements within the study area by comparing the underlying mesh of extracted sea ice surface (i.e., Σ_{20}) with the expected ATM data coverage (i.e., Σ_{25000}), as described in Section 5.4.1. We then compare the regions without measurements to the CAMBOT images (e.g., Fig. 5.13(a)). The results reveal that all regions without measurements correspond to water surface regions or regions covered with thin ice. However, it is important to note that not all water surface regions are identified as regions without measurements by this method. Σ_{20} still includes small melt ponds or narrow leads, and this is due to the small distance between points in these cases. The statistical summary of



(a) γ = 5 m (b) γ = 10 m (c) γ = 25 m (d) γ = 50 m

(e) γ = 70 m (f) γ = 100 m (g) γ = 150 m (h) image

Figure 5.15: Region A: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image.

all extracted water/thin ice surface regions is presented in Fig. 5.13(b). A total of 1.24 km^2 water/thin ice surface regions are extracted with this method, accounting for 1.5% of the study area (82.5 km^2). Considering the potential omission of small water surface regions, this number is likely to be slightly smaller than the actual water surface region area in the study area.

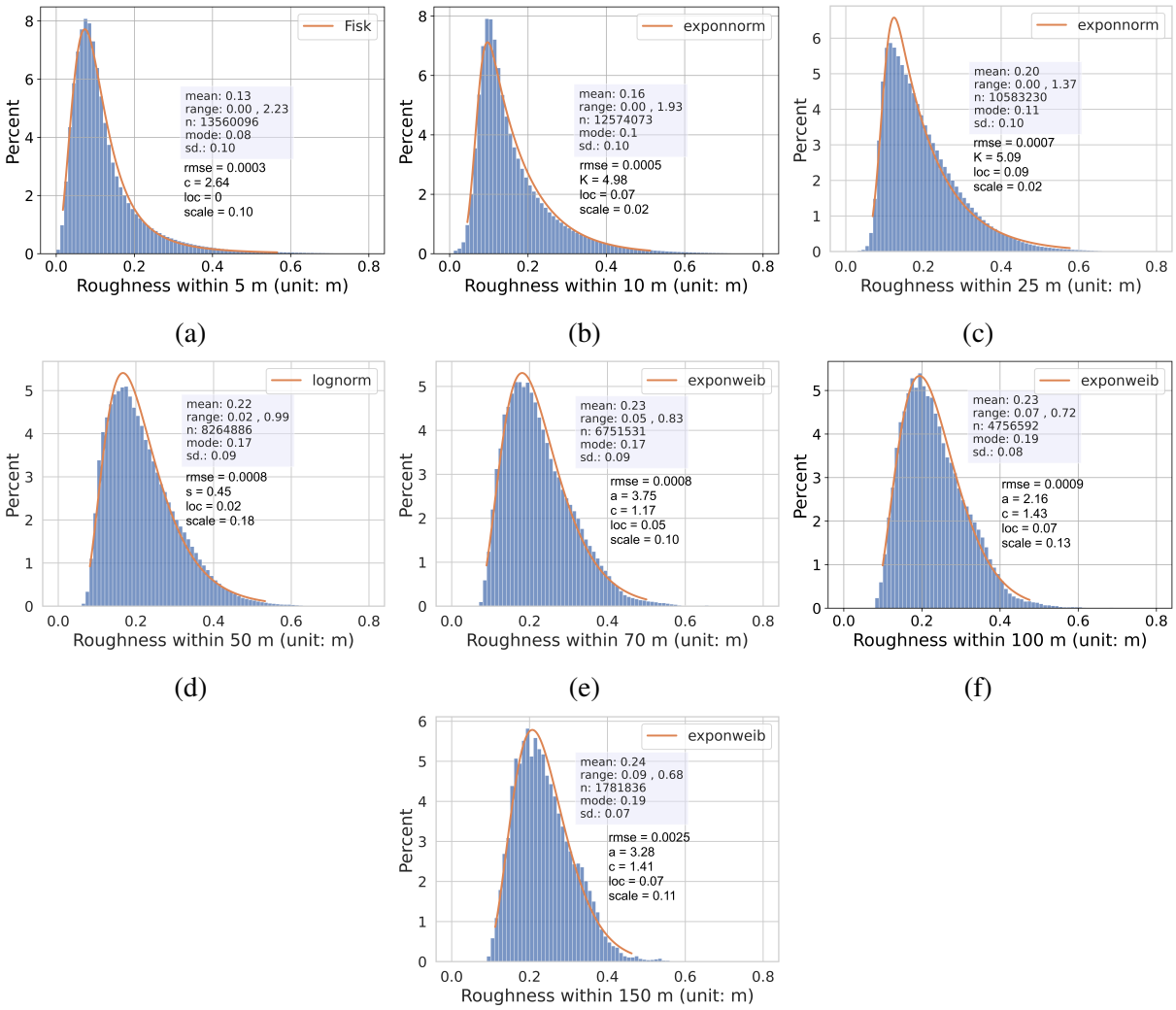


Figure 5.16: The distribution histograms of roughness values at vertices in the study region with different measure range radii (bin width = 0.01 m). The curve line represents the best-fitting distribution model, starting at the 1% percentile point function (PPF) and concluding at the 99% PPF of the distribution. The root mean square error (rmse) and parameters for this best-fitting model are listed. The name of the best-fitting model is labeled in the top-right corner of each chart.

5.6.2 Sea ice surface roughness

In this subsection, we compare the results when different radii are used in the sea ice surface roughness calculation on the TIN as discussed in Section 5.4.2. The selected radii include 5 m, 10 m, 25 m, 50 m, 70 m, 100 m, and 150 m, which closely align with the spatial resolutions

of common altimeters. We take four regions as examples to examine how different radii affect the significance of computed roughness. Roughness is computed on Σ_{20}, and, thus, the regions without measurements have been removed from the TIN leading to white regions in the maps.

Region A (Figs. 5.14 (a, e)) is within a single large ice floe with a narrow lead. The ice floe is likely to be a thick multi-year ice floe with older weathered ridges and a few very young ridges. In region B (Figs. 5.14 (b, f)), there are unconsolidated ice floes with a mixture of floe sizes. Region C (Figs. 5.14 (c, g)) consists of two large ice floes separated by a wide lead. Region D (Figs. 5.14 (d, h)) is located within a large ridged ice floe with no lead. This region features weathered first-year ridges, multi-year ridges, and noticeable sastrugi (snow deposition features). Each of these regions also contains a few melt ponds that melted through to sea level.

Fig. 5.15 shows the roughness maps of region A with different radii. The results of the other regions show a similar pattern and are included in the supplementary material. In the roughness maps with radii γ = 5 m and γ = 10 m, a correlation emerges between high roughness values (indicated by the red areas) and topographic features, such as ice hummocks and ridges in the image. Lower roughness areas are depicted in blue. The visual analysis suggests strong alignment between the areas of higher (5 m/10 m) roughness and the topographical ridge ranges. These maps are crucial for identifying important topographic features on the sea ice surface. On the other hand, when the radius increases (e.g., to γ = 100 m or 150 m), small isolated elevation changes on the surface are averaged out. Higher roughness value is only evident in areas with clustered topographic features (as seen in region B and region C) or significant ridges (as in region A and region D). This phenomenon is expected, as a larger search radius includes more measurements, thereby reducing the impact of extreme values on roughness. Consequently, roughness maps with larger radii offer a broader view of the regional surface's variability, making

them useful for identifying areas with varied surface characteristics in general, rather than specific surface features.

To further investigate the difference among roughness at different scales, we calculate their distribution histograms within the study region. A roughness value based on a radius of γ at a point p is excluded from the distribution model fitting when the circle C_γ, centered at p, is not contained by the expected ATM coverage Σ_{25000}. This is because when C_γ at p is partially contained by Σ_{25000}, the roughness calculation only considers elevation changes along certain directions from p, where the measurements are available, rather than all possible directions. Thus, if the height variation is not isotropic, errors can be induced in the distribution. We compare the roughness distributions against several models. For each radius, we pick the model that fits the distribution best, i.e., the one that has the smallest Root Mean Square Error (RMSE). The results are summarized in Table 5.1.

The distribution of 5 m roughness adheres closely to a *Fisk* distribution, also known as the *log-logistic* distribution. It has an evident long right tail, indicating a tendency of occasional extreme values in the roughness measurements, which is potentially due to the existence of topographic features. It matches our observations on the maps that the 5 m roughness is more likely to be affected by these features. When the radius increases to 10 m and 25 m, the roughness distribution is best described by an exponentially modified normal distribution (denoted as *exponnorm*). Although visually similar to the *Fisk* distribution with a right-skewed tail, the exponnorm distribution is composed of a normal distribution that is skewed due to an exponential component. This might suggest that roughness measurements at these scales are influenced by a Gaussian process with an exponential process, potentially indicative of a directional process such as wind or ocean currents.

Table 5.1: The sum of square error (unit: m^2) and root mean square error (values in the brackets, unit: m) of roughness distributions at different radii and tested models. The tested models include the exponentially modified normal distribution (denoted as *exponnorm*), exponentiated Weibull distribution (denoted as *exponweib*), exponential power distribution (denoted as *exponpow*), Fisk distribution, *log-normal* distribution (denoted as *lognorm*), and normal distribution (denoted asd *norm*).

radius	**exponnorm**	**exponweib**	**exponpow**	**Fisk**	**lognorm**	**norm**
5m	6.365	3.526	143.245	1.3855	6.043	122.765
	(6.8E-04)	(5.1E-04)	(3.2E-03)	(3.2E-04)	(6.7E-04)	(3.0E-03)
10m	3.282	10.918	158.255	16.432	16.977	125.496
	(5.1E-04)	(9.3E-04)	(3.5E-03)	(1.1E-03)	(1.2E-03)	(3.2E-03)
25m	5.615	11.781	128.669	21.798	11.863	89.029
	(7.3E-04)	(1.1E-03)	(3.5E-03)	(1.4E-03)	(1.1E-03)	(2.9E-03)
50m	15.069	24.241	105.121	15.611	5.176	68.126
	(1.4E-03)	(1.7E-03)	(3.6E-03)	(1.4E-03)	(7.9E-04)	(2.9E-03)
70m	16.878	3.873	89.128	16.724	5.69	67.33
	(1.6E-03)	(7.6E-04)	(3.6E-03)	(1.6E-03)	(9.2E-04)	(3.2E-03)
100m	21.285	3.912	80.641	22.134	7.651	65.648
	(2.1E-03)	(9.1E-04)	(4.1E-03)	(2.2E-03)	(1.3E-03)	(3.7E-03)
150m	28.24	10.728	79.956	30.281	12.947	66.635
	(4.0E-03)	(2.5E-03)	(6.7E-03)	(4.1E-03)	(2.7E-03)	(6.1E-03)

The 50m roughness distribution follows a *log-normal* distribution (denoted as *lognorm*), indicating that the logarithms of the roughness values are normally distributed. As we consider larger radii (70 m in Fig. 5.16(e), 100 m in Fig. 5.16(f), and 150 m in Fig. 5.16(g)), the *exponentiated Weibull* distribution (denoted as *exponweib*) becomes a better fit for the roughness distribution. The *exponweib* distribution can model a wider range of shapes compared with the *exponnorm* or the *Fisk* distributions, which implies that, at larger scales, the roughness distribution is influenced by a more complex interplay of factors.

When comparing the roughness distributions at seven different scales, we can see that all the distributions are right-skewed to different degrees. When a larger radius is used, the distribution becomes less skewed, the curve around peak becomes less steep, and the data range becomes

smaller. This observation aligns with our findings on the maps that the roughness changes are less significant when a larger radius is used, but it is still capable of highlighting areas with complex surface.

5.6.3 Selecting parameters for ridge extraction

In this subsection, we discuss how the input parameters affect the ridge extraction results and how we select them. As mentioned in Section 5.5, there are four parameters in the ridge structure extraction algorithm: the alpha value α for the surface model generation process, the persistence threshold ω_m for the topological simplification, the peak height threshold H_m for selecting valid ridge peaks, and the 5 m roughness threshold θ_m for the roughness filter. Among them, the selection of α has been discussed in Section 5.3 and H_m is set to 0.6 m following previous studies [54, 67, 182]. Therefore, in this subsection, we focus on the selection of ω_m and θ_m.

Two independent ridge datasets obtained from other methods are used to validate the results from our method and to select ideal parameters. The first one is the ridge peaks extracted from ICESat-2 profile data through the UMD-RDA [54], denoted as R_I. The other dataset consists of ridge lines visually extracted from three CAMBOT images in the study area, denoted as R_M. Inspired by previous studies [54, 132], the visual inspection of ridge lines is performed by identifying lines separating the ground and shadows from images. Lines that pass through any area higher than 0.6 m are added as valid ridge lines to R_M. Note that R_I is a set of points representing ridge peaks and R_M is a set of lines representing ridges. The ridge structures extracted from OIB ATM data using our method are denoted as R_A.

We focus on a 72 km segment within the study area that matches the best with the ICESat-2 track. The modal 5 m roughness of all points in the 72 km segment with $H_A \leq 0$ is 0.09 m. We assume this value represents the roughness of level sea ice surface. Therefore, it is set as the maximum roughness threshold to test in our experiments. Recall that the persistence ω is used in the topological simplification step and is the elevation difference between two nodes connected by an arc in the surface network *G*. Since the goal of topological simplification is to remove noise and small variations on the surface, the maximum value of ω_m to be tested in the experiments should be less than H_m, otherwise the topological simplification would remove valid ridges from the extracted paths. Thus, the maximum ω_m to test in the experiments is set to 0.5 m. We extract ridge structures using different combinations of θ_m and ω_m values. Parameter θ_m is varied in increments of 0.01 m, ranging from 0 to 0.09 m. Parameter ω_m is varied in increments of 0.05 m, ranging from 0 to 0.5 m.

Three criteria are considered when selecting the parameters for ridge extraction in this study: (1) the number and total lengths of extracted ridges in R_A; (2) the ratio of matched ridges in independent datasets R_M and R_I; and (3) the matched length of ridge segments in the visually selected validation dataset R_M.

5.6.3.1 Influence of parameters on extracted ridge range

We evaluate the impacts of ω_m and θ_m on the number of extracted ridges in R_A that are within the ranges of independent datasets R_I and R_M. The range of R_I is the 5 m buffer zone around the ICESat-2 track in the study region. The range of R_M is the ranges of six inspected images. In general, an increase in ω_m results in fewer R_A ridges within the ranges of both datasets

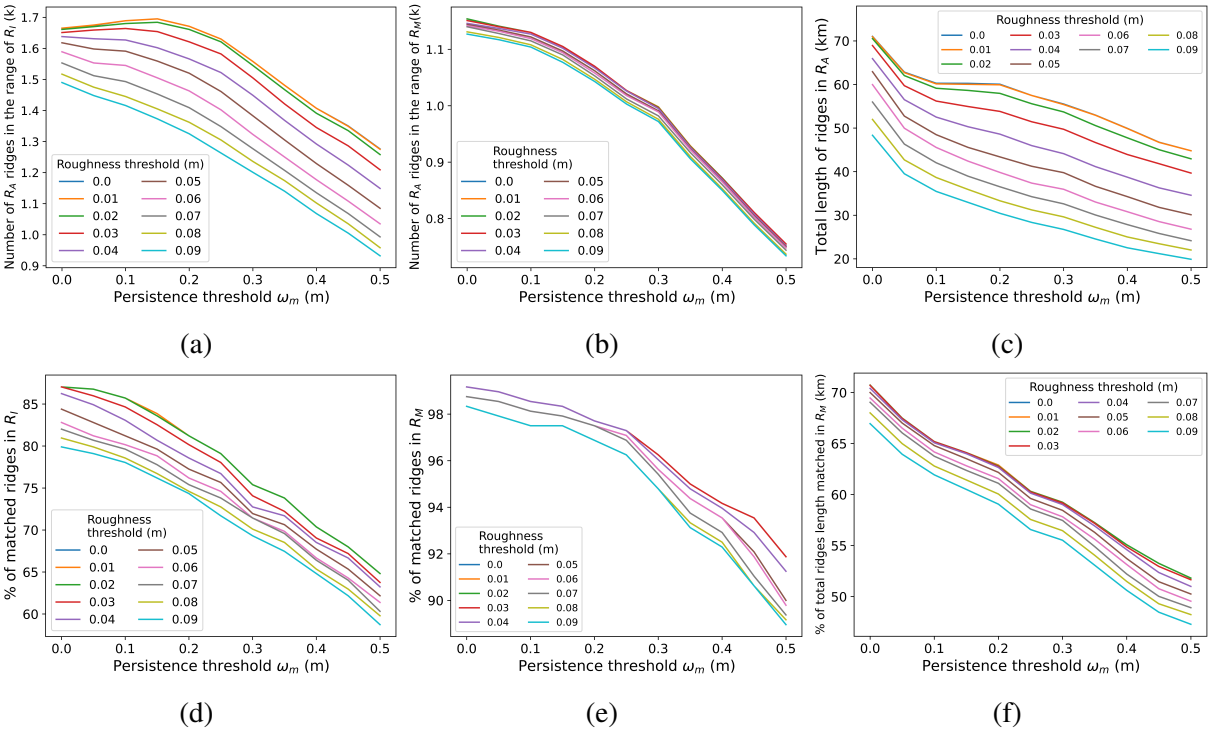


Figure 5.17: Relationship between roughness and persistence thresholds and (a) the percentage and the total number of R_A ridges that intersect with the 5 m buffer zone around the ICESat-2 track; (b) the total number of R_A ridges within the range of six visually inspected images; (c) the total length of R_A ridge parts within the range of six visually inspected images; (d) the percentage of R_I ridges that have a match in R_A; (e) the percentage of R_M ridges that have a match in R_A; (f) the total length of R_M ridge parts matched with R_A. R_I: the ridge peaks extracted from ICESat-2 profile data through the UMD-RDA [54]. R_M: ridge lines visually extracted from three CAMBOT images in the study area. R_A: ridge structures extracted from OIB ATM data using our method.

R_I and R_M (see Fig. 5.17(a) and 5.17(b)). The only exception occurs when both ω_m and θ_m are small ($\omega_m \leq 0.15$ m and $\theta_m \leq 0.02$ m). Under these conditions, an increased ω_m leads to more R_A ridges within the range of R_I. This unexpected outcome can be explained by the effect of topological simplification and the way we count ridges. As discussed in Section 5.5.2, though topological simplification consistently removes maxima and saddles, it can also lead to longer ridge structure lines. Thus, it is possible that the number of R_A ridges intersecting a region goes up when the total number of R_A ridges goes down. In fact, the relationship between the total

number of R_A and ω_m (not displayed here) shows a consistent negative correlation.

The value of θ_m and the number of ridges in the validation dataset range also shows a negative correlation. One interesting observation is that the difference of numbers in the range of R_M is considerably less significant compared to that of R_I. A plausible reasoning is that the roughness filter typically trims the ridges rather than removing them, unless the entire length of a ridge falls within low 5-m-roughness regions. Considering that the range of R_I spans a $10m \times 72km$ band, whereas the range of R_M consists of six $\sim 400m \times \sim 650m$ rectangles, the number of R_A ridges intersecting the range of R_I is more likely to be influenced by the reduced ridge lengths due to its narrower range.

Fig. 5.17(c) illustrates the change in the total length of R_A ridges in ranges of six inspected images. Similarly to the number of ridges, the length decreases when either ω_m or θ_m increases. However, when comparing the rate of change between Figs. 5.17(b) and (c), it is evident that θ_m has a more significant influence on the total length compared to the number of ridges. This result aligns with our earlier assumption that the roughness filter tends to trim the ridges without completely removing them.

5.6.3.2 Influence of parameters on matched ridge number in validation datasets

We explore the relationship between ω_m and θ_m and the percentage of matched ridges in both independent datasets. A ridge peak r_i from R_I is considered matched with a ridge structure r_a in R_A if r_a intersects with the 5 m buffer zone around r_i. Similarly, a ridge r_m from R_M is considered matched with a ridge structure $r_{a'}$ in R_A if $r_{a'}$ intersects with the 2 m buffer zone around r_m. we find that increasing the persistence threshold ω_m leads to a lower percentage of

matched ridges in both validation datasets (see Fig. 5.17(d) and 5.17(e)). This can be explained by the decrease in number of ridges when a larger ω_m is used. We also notice that when ω_m is smaller than 0.15 m, the decline in matched ridges in R_M is less pronounced. This phenomenon suggests that a large ω_m is more likely to significantly impact the matching result. When comparing different roughness thresholds θ_m, we observe that higher θ_m values tend to result in lower percentages of matched ridges in the independent datasets. Similar to the reasoning for ω_m, some ridges are shorten or removed by the roughness filter, resulting poorer alignment with the independent datasets. Thus, the percentages of matched ridges in both R_I and R_M decrease as θ_m increases. Additionally, we find that the impact is much less significant when θ_m is small (e.g., ≤ 0.02 m), likely because there are few areas in the study region with such a low 5 m roughness.

5.6.3.3 Influence of parameters on matched ridge length in validation datasets

Since R_M consists of lines, we also assess the percentage of matched ridge lines relative to the total length of R_M when different parameters were used. Differently from the way to calculate the number of matched ridges, we collect segments of ridges in R_M that fall in the 2 m buffer zone of R_A and considered them as matched segments of R_M. The percentage of matched ridge length *LP* is the total length of matched segments relative to the total length of R_M. The trends in *LP*, as depicted in Fig. 5.17 (f), are similar to the changes in the percentages of matched ridges in Fig. 5.17(e). *LP* decreases from 70% to around 50% when ω_m increases, and it also decreases when θ_m increases.

5.6.3.4 Discussion on parameter selection

Overall, the percentage of matched ridges in R_M is higher than that in R_I when the same set of parameters are used. There are two main reasons for this. Firstly, R_I contains only ridge peaks while R_M includes ridge lines and are more likely to intersect with the extracted result. This is also why we use a 5 m buffer zone for R_I as opposite to the 2 m buffer zone used for R_M. Secondly, despite drift corrections, the ICESat-2 profile elevations cannot perfectly align with the OIB ATM-TIN elevations. The spatial resolutions of the two instruments are different and the drift rates of different floes might not be the same. Conversely, the R_M dataset is derived from CAMBOT images taken concurrently with OIB ATM points, and as such, it provides better matching results.

Although the percentage of matched ridges in R_M is in general high ($\geq 87.9\%$ in all experiments), the percentage of matched ridge length is notably lower, ranging from 47% to 70%. This discrepancy may be caused by several factors. Some sea ice surface elevation changes are accurately reflected in the OIB ATM data due to limited spatial resolution, and consequently, some ridges are not extracted or have shorter structure lines than the actual ranges. Additionally, the accuracy of visual inspection is affected by the complexity of sea ice surface and by solar elevation angles. Manually selecting ridge lines through visual inspection may introduce inaccuracies, especially for lower or structurally complex ridges, which are more challenging to identify.

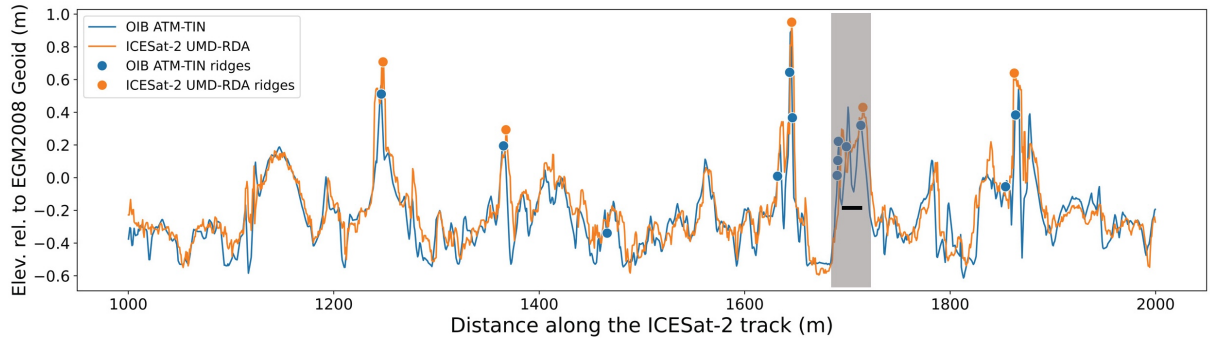
While all three measured criteria reflect the quality of extracted ridges, parameter selection depends on specific applications. In this study, the focus is on achieving a higher ridge match ratio without overestimating surface complexity. Therefore, from the results of all combinations of ω_m and θ_m, we select the pair of parameters that ensure 95% matched ridges in R_M and

70% matched ridges in R_I, while minimizing the total extracted ridge length. The percentage of matched ridge length is not used for parameter selection since most combinations do not have a satisfying result in this criterion and the above discussion shows that its value may be affected more by data quality. Following this rule, ω_m = 0.25 m and θ_m = 0.09 m are used for ridge extraction in the following experiments.

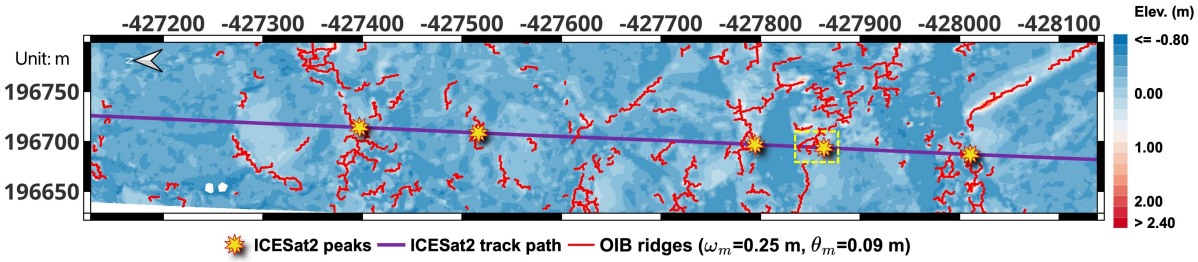
5.6.4 Results from ridge extraction

5.6.4.1 Comparison between extracted ridges and UMD-RDA result

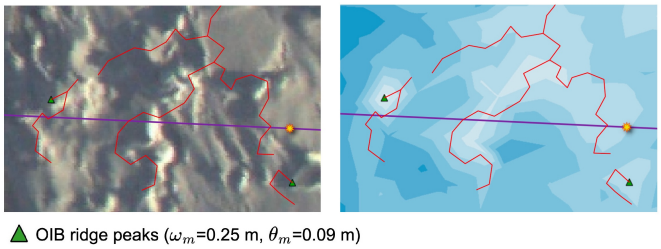
As mentioned in Section 5.3, we can calculate a profile from the OIB ATM-TIN model along the ICESat-2 track through interpolation. Fig. 5.18 shows a segment of the profiles from OIB ATM-TIN model and from ICESat-2 UMD-RDA model. Within this segment, five ridges are detected by the UMD-RDA, i.e., they are in R_I. By comparing these five ridges to the ridges extracted from the ATM-TIN model (i.e., ridges in R_A), we find that all five ridges match with ridges in R_A. Around some ridges, there are more than one intersections between ridges in R_A and the ICESat-2 track. Taking the segment highlighted in Fig. 5.18(a) as an example, when looking closer to the segment (as shown in Fig. 5.18(c)), we see that the ridge peak from R_I is detected by the propose method, but there are five intersections between ridges in R_A and the ICESat-2 track. This result is expected, considering that the ridge structures in R_A extend from peaks to bottoms instead of containing only ridge peaks. Besides, the elevation changes more rapidly around the ridge peak (marked with black underline) in the profile from OIB ATM-TIN model than the ICESat-2 UMD-RDA profile. Thus, two adjacent ridges are considered as separate ridges in each segment by our method, which matches with the observations on the image, where



(a)



(b)

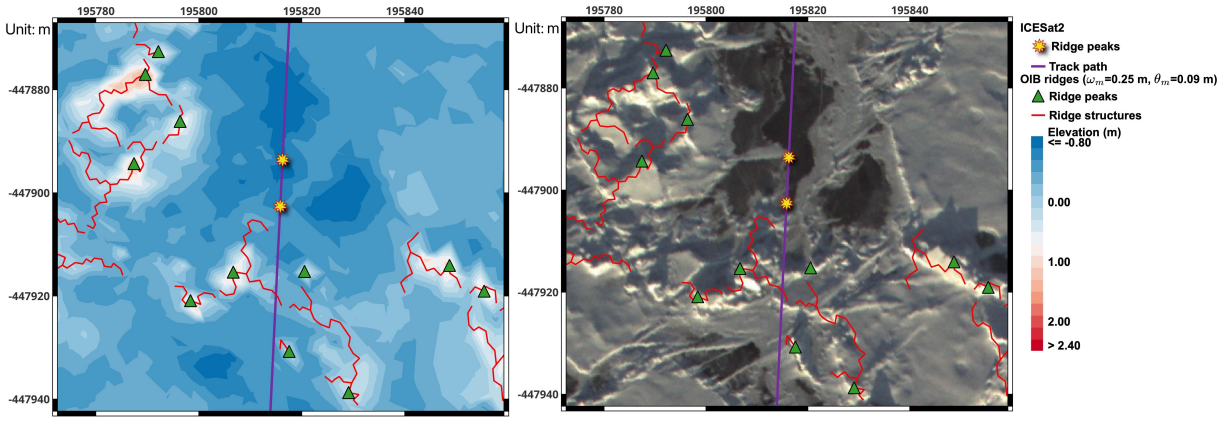


(c)

Figure 5.18: (a) The profiles from interpolation on OIB ATM-TIN model along ICESat-2 track (blue line) and the ICESat-2 ATL03 University of Maryland-Ridge Detection Algorithm (UMD-RDA) (orange line) along a 1 km segment. Orange circles indicate the ridge peaks identified by UMD-RDA and the blue circles indicate the intersections between the extracted ridges from ATM-TIN model and the ICESat-2 track; (b) elevation map of ATM TIN model with extracted ridges of the 1 km segment shown in (a), the region bounded by yellow box corresponds to the highlighted segment in (a); (c) Zoom-in images and TINs of the selected segment in (a - b). All ridges are extracted by our method when the persistence threshold ω_m = 0.25 m and roughness threshold θ_m = 0.09 m.

the two ridges are distinct.

Fig. 5.18(b) presents a map of extracted ridge structure lines within the selected segment. The result shows that most pronounced ridges, that can be visually identified from the TIN, are picked up by the proposed method. We also notice that there are many short ridge structures in

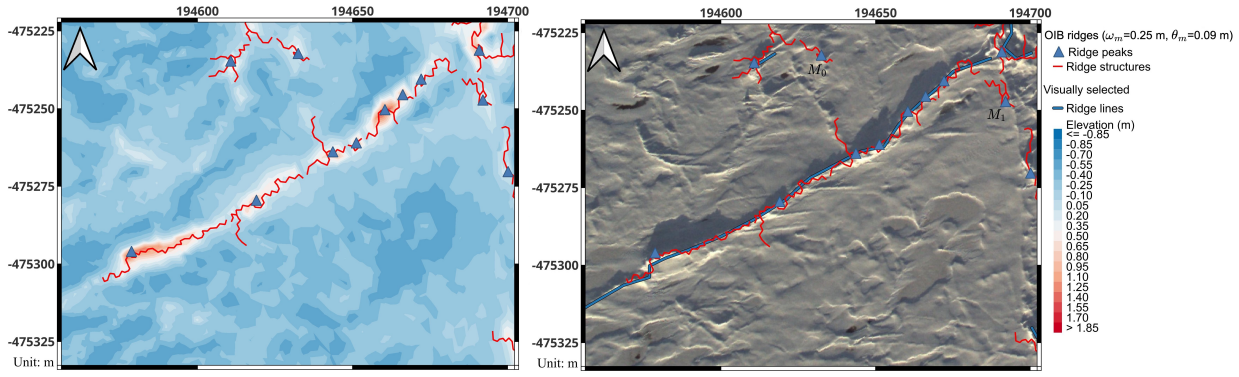


(a) (b)

Figure 5.19: Example where the ridge results from the ICESat-2 ATL03 UMD-RDA does not match with the results from our method operated on the OIB ATM-TIN model. The basemap in (a) is the α-shape generated from OIB ATM when $\alpha = 20$ as described in Section 5.3 and the image in (b) is the OIB CAMBOT image coincident with the OIB ATM data.

the selected segment. Those small length-scale ridge features are common in areas where sea ice dynamics are prevalent. Convergence between ice floes fractures the floes into smaller pieces. These small floes are pushed upward from the pressure causing a floe edge to protrude higher above the surface. A divergence event (i.e. opening of a lead) can cause these small upturned floes to drift away where they are eventually refrozen into the new ice growth within the lead.

Fig. 5.19 shows an example when ridges from ICESat-2 UMD-RDA are not picked up by the ATM-TIN method. In Fig. 5.19, the two ridges detected by UMD-RDA are in the range of melt ponds in OIB data. This result is likely caused by the time difference between the two datasets. Although the drift correction has been applied, it cannot ensure a perfect match at all locations along the track. Unmatched ridges, like in this case, can be found by selecting ridge peaks with very low elevation anomalies H_A in the ATM-TIN model. For instance, among 107 ridge peaks in R_I that are not matched with ridges in R_A, 43 of them have $H_A <= 0$ m from the ATM-TIN model.



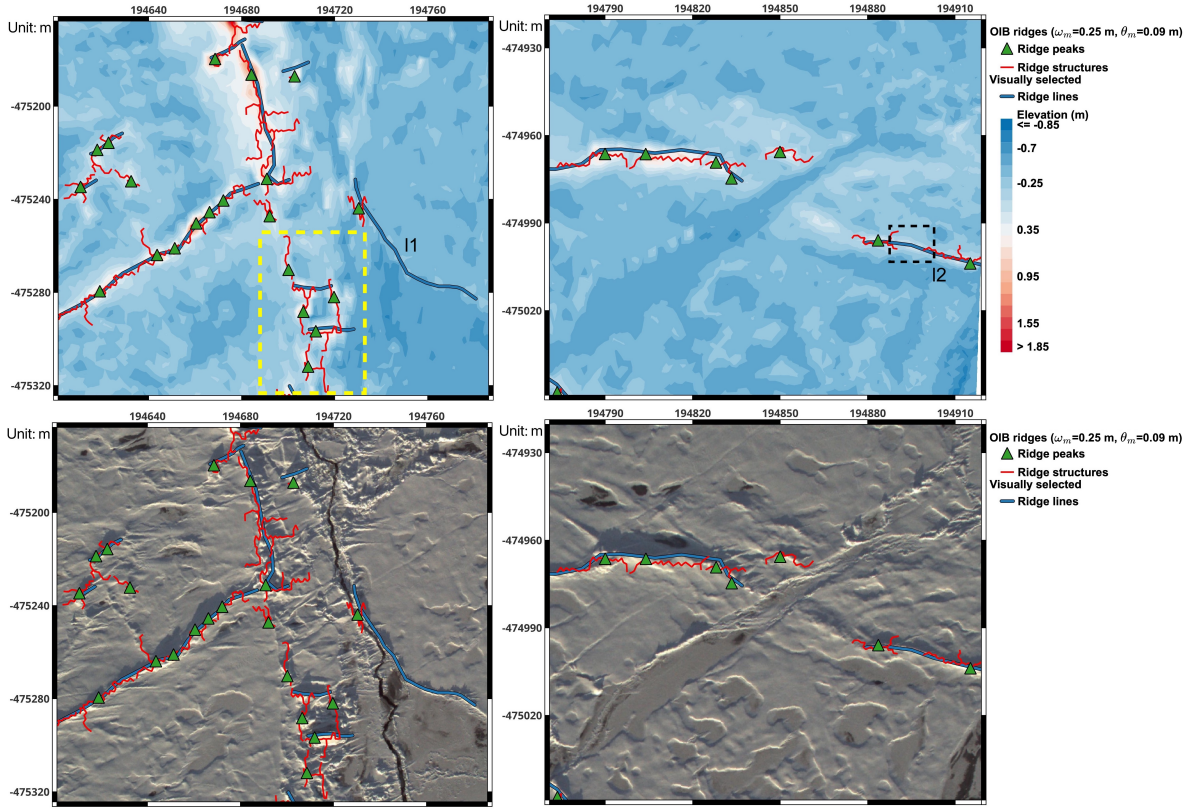
(a) (b)

Figure 5.20: Example of ridge extraction results when ω_m = 0.25 m, θ_m = 0.09 m. The basemap in (a) is the α-shape generated from OIB ATM when $\alpha = 20$ as described in Section 5.3 and the image in (b) is the OIB CAMBOT image coincident with the OIB ATM data. The blue lines in (b) are ridge lines selected through visual inspection for reference.

5.6.4.2 Comparison between extracted ridges and visually selected result

In Fig. 5.20, we compare the the ridges in R_A with the ridge lines picked through visual inspection, i.e., ridges in R_M, in a selected region. As we can see from the figure, the major ridge in the middle of the region has been picked up by our proposed method and the extracted ridge structure lines match well with the ridge line from R_M. Moreover, the propose method is able to pick up small ridges that are difficult to identify through visual inspection. For instance, maxima M_0 and M_1 in Fig. 5.20(b) are two ridge peaks detected by the proposed method. But they are hard to identify from the image as their ridge lines are short and the shadow around them is not evident.

In Figs. 5.21, we show another two factors that can lead to a discrepancy between R_A and R_M. Within Fig. 5.21(a), we can see that most ridge lines in R_M match well with ridge structures in R_A, except for ridge line l_1 from R_M. A ridge peak on l_1 has been detected correctly by our proposed method. The extracted ridge structure, however, does not match with the ridge line



(a) (b)

Figure 5.21: Examples where the ridge results from the visual inspection do not match with the results from our method operated on the OIB ATM-TIN model. The upper figures, colored with blue-red legend, are the α-shapes generated from OIB ATM when $\alpha = 20$ and the lower figures are OIB CAMBOT images coincide with the OIB ATM data. The yellow dashed box in (a) highlights a region with complex terrain features and our method is able to identify such features. The black dashed box in (b) highlights a segment of visually selected ridge line l_2 that does not match with ridges extracted by our method.

from the image. When checking the TIN of the study region, we notice that along l_1, there are significant variations in elevation. A few distinct low points along l_1 prevent the extension of the ridge structure lines along this path during the extraction. Besides, the long shadow above l_1 indicates that the elevation of the ridge is evidently higher than the local level sea ice, while it is not evident in the ATM-TIN model. One explanation is that the ridge is too narrow with respect to the spatial resolution of OIB ATM, and thus, only the elevations around the ridge are collected. Although the proposed ridge extraction method does not extract the complete line along l_1, it has

fairly good outcomes on other ridges. For instance, it successfully extracts the ridge lines on the left half of the shown range, even in complex regions (like the region highlighted with yellow dash box).

Fig. 5.21(b) shows a similar example. The extracted ridge structure from R_A breaks in the middle of a visually detected ridge l_2 from R_M. There are two saddles along l_2 in the ATM-TIN model, which split l_2 into three parts. The middle part (in the black dashed box) between these two saddle is not high enough to be considered as a separate ridge, but the two saddles are low enough to separate it from the two adjacent ridges. Such a ridge line mismatch is common in our results because it is not easy to tell the ridge height change from images, which makes it hard to decide the end points of ridge lines. Therefore, when we decided the optimal parameters, we considered mainly the number of matched ridges instead of the total matched length.

5.6.4.3 Statistics of extracted ridge structures

Fig. 5.22 presents a set of five histograms that describe the extracted ridges by our proposed method with ω_m = 0.25 m and θ_m = 0.09 m. In the 72 km long, 26.22 km^2 large study area, a total of 20558 ridges have been detected. 755 of them do not have ridge lines after the processing, and thus, are considered as small features and are excluded from the following statistics. Figs. 5.22 (b) and (c) show the distributions of ridge lengths and H_A of ridge peaks, which are indicators of the size of a ridge. Both distributions are right-skewed, with the majority of values clustered at the lower end and a tail extending towards the higher end. The modal ridge length is 4.38 m while the mean length is 25.13 m, which is much higher. This might be caused by the 1% ridges that are longer than 106 m in the result. The range of ridge lengths is quite broad, extending from

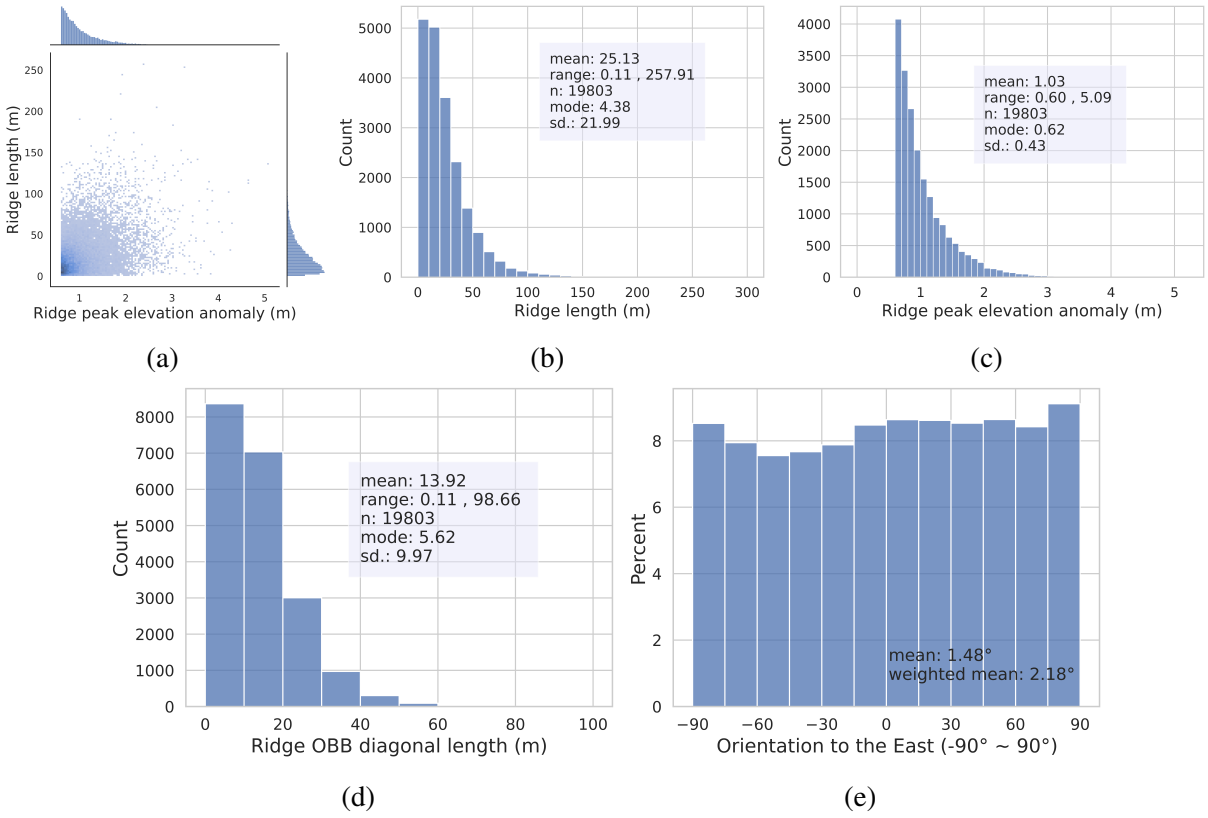


Figure 5.22: (a) Joint distribution of ridge lengths and ridge peak elevation anomalies H_A; (b) Ridge length distribution (bin width = 10 m); (c) Ridge peak H_A distribution (bin width = 0.1 m); (d) Diagonal length distribution of ridge Oriented Bounding Box (OBB) (bin width = 10 m); and (e) ridge OBB orientation distribution (bin width = 15°). The ridges results in all five figures are extracted with persistence threshold ω_m = 0.25 m and roughness threshold θ_m = 0.09 m.

0.11 m to 257.91 m, suggesting a high degree of variability in ridge length measurements. This is further supported by a standard deviation of 21.99 m. The ridge peak elevation anomaly ranges from 0.60 m to 5.09 m, with a small standard deviation 0.43 m. This might indicate that despite of a few higher ridges, most of the ridge peak anomalies are clustered around the mean, which is 1.03 m. Fig. 5.22(a) illustrates the joint distribution of ridge length and ridge peak H_A. It further demonstrates that the sizes of ridges in the study area tend to cluster on the smaller side.

It is noteworthy that the ridge structures extracted with our method are not limited to a single major ridge branch, and therefore, the ridge length might be overestimated. To assist the

evaluation of ridge lengths in the area, we calculate the Oriented Bounding Box (OBB) [42] of each ridge and use the length of OBB diagonal as another indicator of the ridge length. This value is the lower bound of real ridge length as the major branch is not necessary a straight line. As shown in Fig. 5.22(d), the variability in the ridge OBB diagonal lengths is much smaller compared to the ridge structure lines lengths, as the distribution has a smaller standard deviation and a smaller range). But interestingly, the OBB diagonal length has a greater mode (5.62 m) than the modal ridge structure length (4.38 m). This finding shows that many ridge structure lines are longer than the OBB diagonal and many of them might have similar major branch lengths, but with varied branches. Besides, we calculate the orientation of a ridge as the orientation of the longer edge of the ridge OBB relative to the East (see Fig. 5.22(e)). In general, the orientations show a random distribution, with a mean as 1.48°. A weighted mean is calculated by using the length difference between the longer edge and shorter edge of the OBB as the weight. The weighted mean (2.18°) is also close to 0 indicating there is no pronounced orientation in the extracted ridges.

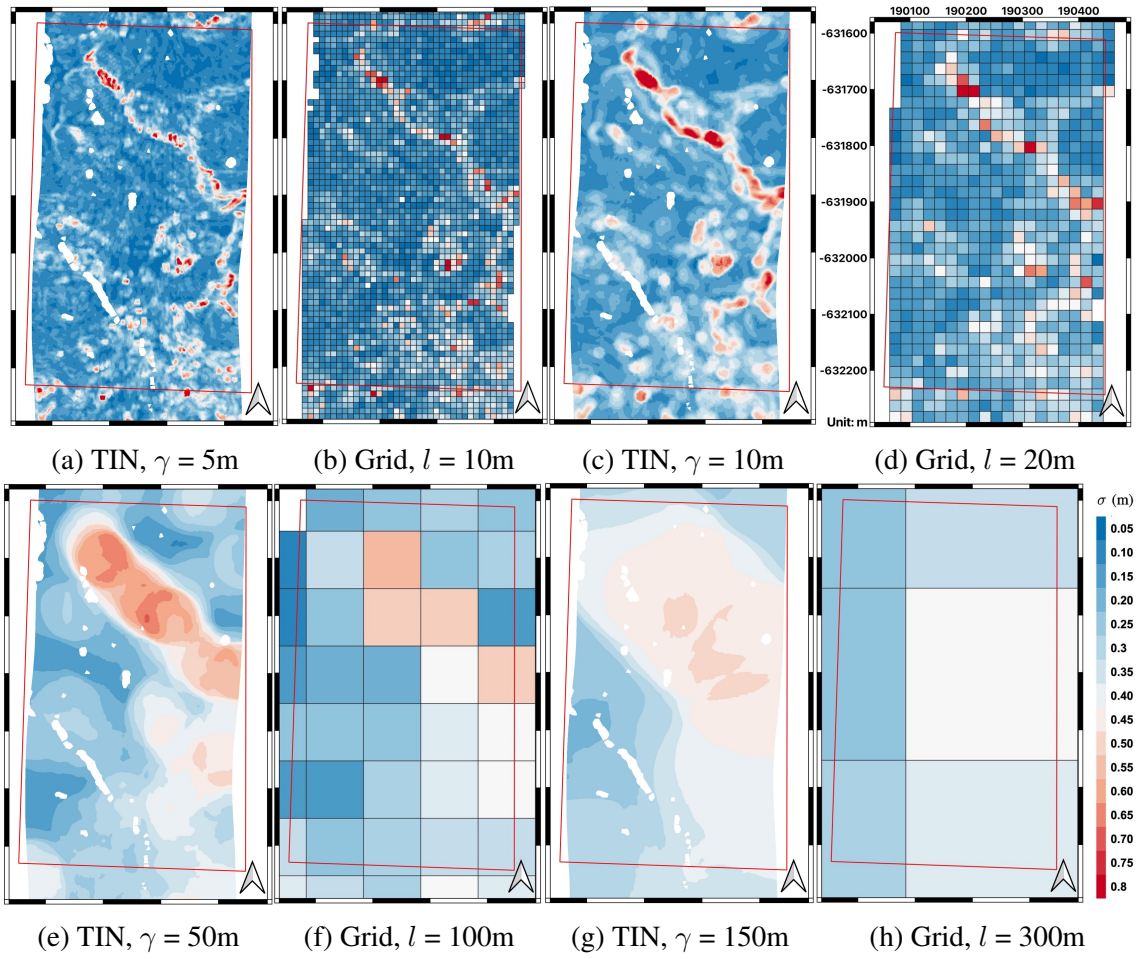
5.7 Discussion

When studying sea ice surface topography with LiDAR point cloud, one common way is to generate gridded surface models. In this study, we proposed an alternative way for representing the sea ice surface, i.e., a Triangulated Irregular Network (TIN). We described the method to generate TINs with accurate boundaries from OIB ATM point clouds and methods to extract two-dimensional sea ice topographic features from TINs. Compared to gridded models, TIN-based models have the advantage that no information is lost during the surface model generation. Our

experiments demonstrate the possibility of using TIN-based methods to calculate topographic statistics, such as density and roughness, to evaluate data quality by identifying regions without measurements, and to locate and extract the ranges of three-dimensional topographic features, like sea ice ridges.

Point density and surface roughness are calculated at the vertices of TIN. This is different from the calculation on gridded surface models, where the density and roughness are calculated within each grid cell. The TIN-based statistics provide a new localized view of the point cloud dataset. As shown in the roughness experiments in Section 5.6.2, the distribution of calculated measures is affected by the measuring scales, i.e., the search radii at each vertex. Compared to gridded surface models, TINs provide more flexibility when different scales of statistics are needed since the information of the input point cloud is not discarded. Taking the roughness as an example, the measure scale of roughness that can be calculated on a gridded model is limited by the spatial resolution of the grid. Conversely, different radii can be used to calculate roughness on a TIN.

One common way to calculate grid-based roughness is to put points in a grid and calculate the roughness for each cell. The roughness calculated in this way is sensitive to how the study region is subdivided, especially at larger measurement scales (see examples in Figs. 5.23(f, h)). For instance, roughness values can vary significantly depending on whether a cluster of points with high elevations is contained within a single grid cell or distributed across two separate grid cells along with other points at lower elevations. At smaller scales, TIN-based and grid-based roughness results are more similar. But areas with high TIN-based roughness still demonstrate a stronger correlation with features on sea ice surfaces and can contribute to the detection of feature boundaries (see examples in Figs. 5.23(b, d). It is noteworthy that directly comparing the



(a) TIN, γ = 5m (b) Grid, l = 10m (c) TIN, γ = 10m (d) Grid, l = 20m

(e) TIN, γ = 50m (f) Grid, l = 100m (g) TIN, γ = 150m (h) Grid, l = 300m

Figure 5.23: Roughness maps calculated with TIN-based method and grid-based method with similar measure scales. For instance, when the search radius γ on TINs equals to 5, the corresponding grid cell size l is twice the radius, which is 10. The region in the map is Region A introduced in Section 5.6.2.

distributions of measures (e.g., density or roughness) calculated on TINs with those calculated on grids may not be appropriate, even when the same measure scale is employed. This is because the two types of models use different spatial representations. A significant difference, for instance, is that there can be overlaps in the search ranges when statistics are calculated at vertices of a TIN while the statistics are calculated for non-overlapping blocks in a grid-based method.

The distributions of roughness using different radii can offer valuable insights for simulation models that require accurate representations of the sea ice surface. By examining how

roughness distribution varies across different spatial scales, these models can better simulate real-world conditions and phenomena. Meanwhile, the roughness distributions at different scales also provide valuable information about the selection of spatial resolution and sea ice topography research. The roughness at larger scales has smaller ranges showing that small features are averaged out when a large scale is used to study sea ice surface topography. Therefore, these distributions can guide the designers of raster data products to select a suitable resolution so that the features of interest are maintained with the minimum storage cost. Understanding roughness at different scales can also help scientists to decide suitable resolution for altimeters to collect information from sea ice surface.

In Section 5.4.1, we introduced methods to identify regions without measurements. These regions are crucial for assessing data quality and understanding the reasons for data loss. For instance, in this study, regions without measurements corresponded closely to open water regions, consistent with previous findings that data dropout happens over water surface in recent OIB ATM wide-scan datasets [106, 179]. This correlation can guide the detection of features such as melt ponds and leads, which are essential for understanding sea ice dynamics and freeboard measurements. The method to extract open water regions may not apply to other datasets, since not all LiDAR datasets experience data dropouts over water surfaces. However, it can still be used to locate regions without measurements in such datasets.

In Section 5.5, we proposed a new topology-based approach for identifying sea ice ridges and extracting their structure lines. The ridges extracted from TIN-based models demonstrate that more detailed information, including range, length, and orientation, can be extracted from scanned point cloud swaths compared to linear scanning results such as those from ICESat-2. This highlights the potential and necessity of large-scale LiDAR scanning data to enhance our

understanding of complex surface features like ridges in polar regions. In Section 5.6.3, we discussed the selection of parameters for the ridge extraction method. Despite our efforts to establish evaluation criteria for ridge extraction results, the lack of ground-truth data still makes it challenging to set a single optimal rule for parameter selection. Regardless of the selection of parameters, the proposed method extracts a significantly larger number of ridges compared to other two independent datasets. This discrepancy suggests that traditional profile-based methods and visual-inspection-based methods may underestimate the number of ridges and the region complexity. But further study with on-site ground-truth data is needed to validate this assumption.

Overall, our study demonstrates that more information about sea ice surface can be obtained from large-range LiDAR point cloud swaths compared to what laser profiles or aerial images can provide. Utilizing efficient data representations, such as Terrain trees, enables the efficient handling of large-scale LiDAR datasets. However, the spatial and temporal ranges of currently available LiDAR swaths are limited as they are mainly collected from airborne altimeters. Future research in sea ice topography stand to benefit from the growing availability of large-scale LiDAR swath data and the application of TIN-based surface models.

Chapter 6: Conclusions

6.1 Major contributions

The main research objective of this dissertation is to develop methods to enable direct terrain modeling and analysis on point cloud datasets by using TIN-based terrain models.

The major challenges in handling large TINs are their high time and storage costs during processing. A new efficient decomposition-based data representation, called Terrain trees, was introduced in 2017 [69] to address this issue. This dissertation extends this work by introducing the Terrain trees family, which contains spatial indexes based on different subdivision rules. This dissertation addresses main research objective by developing efficient terrain analysis methods based on Terrain trees and topology-aware mesh simplification methods to reduce sizes of terrain meshes without affecting topographic analysis and simulation results. Moreover, this dissertation uses the sea ice surface topography study as an example to demonstrate the possibility and advantages of TIN-based terrain analysis. In the following, I list the major contributions of each study and how they correspond to research questions raised in Section 1.2. Moreover, I also explain how these three studies are related to each other.

Chapter 3 addresses the first research question by introducing a family of spatial data structures, named Terrain trees family, for the representation and processing of TINs. A Terrain tree combines a minimal encoding of the connectivity of the TIN with a hierarchical spatial index.

Additionally, the indexes of vertices and triangles associated with each leaf node are encoded through a Sequential range encoding (SRE). This design of Terrain trees helps to solve sub-question 1-1, as it reduces storage costs by 36% compared to the IA data structure [145], a compact state-of-art data structure for TINs.

Additionally, this chapter presents a new open-source library for terrain analysis based on Terrain trees, named the Terrain trees library (TTL) [73]. It includes algorithms for morphological features extraction, such as roughness and curvature, and for topology-based analysis of terrains. Moreover, it includes a technique for multivariate visualization, which enables the analysis of multiple scalar fields defined on the same terrain. Sub-question 1-2 aims to lowering the memory and time requirements for terrain analysis on TINs. Within TTL, two designs contribute to this goal. Firstly, connectivity relations are extracted locally at run-time, within each leaf block of a Terrain tree, based on specific application needs. This design keeps the memory footprint low and speeds up the extraction of Vertex-Triangle and Vertex-Vertex relations compared to the IA data structure. Secondly, spatial queries are performed by exploring the hierarchical data structure, significantly reducing the number of required geometric operations. The optimization of these queries, which are fundamental to complex terrain analysis methods, facilitates efficient terrain analysis on TINs.

This study also includes an extensive evaluation of the three indexes in the Terrain trees family and a discussion on the calibration of capacity values guiding the generation of Terrain trees. To address sub-question 1-3, seven TINs with a number of vertices ranging from 34 million to 193 million generated from real-world datasets are used in the evaluation. The experimental results demonstrate that the analysis methods based on Terrain Trees scale well, even with large datasets. The work in this study is published on GeoInformatica journal in 2023 [72]. As men-

tioned in Foreword, this work is a collaborative work of our research group. The first three authors are listed in alphabetical order and have contributed equally to the paper, while the fourth author is the senior author and advisor.

Chapter 4 addresses the second research question outlined in Section 1.2, i.e., the problem of efficiently reducing the size of a TIN while considering the topology of its represented terrain. Building on the work of Iuricich et al. [99], we propose a new efficient topology-aware TIN simplification algorithm on Terrain trees, introduced in Chapter 3. This method coarsens a TIN without affecting the topology of its underlying terrain, i.e., without modifying critical features of the terrain, such as pits, saddles, peaks, and their connectivity. A batched processing strategy based on Terrain trees has been utilized to reduce the memory and time requirements for this process, effectively addressing sub-question 2-1.

For sub-question 2-2, the goal is to further accelerate this simplification method by adapting it to a parallel algorithm on a multi-core platform. However, the design of the topology-aware simplification algorithm makes it vulnerable to possible read/write conflicts among different cores. To address this problem, this study proposes a leaf-locking strategy for Terrain trees to prevent such conflicts during parallel simplification. Additionally, a Terrain tree after the simplification process becomes less compact and efficient. Thus, we propose an efficient post-processing step for updating hierarchical spatial decomposition. To address sub-question 2-3, we develop a topological cancellation operator on Terrain trees based on the method proposed by Iuricich et al. [123]. This operator removes critical simplices that originate from noise and insignificant features and can serve as a pre-processing step for the topology-aware mesh simplification.

Six TINs with a number of vertices ranging from 25 million to 113 million generated from topographic and bathymetric LiDAR data are used in the evaluation. Topology-aware simplifica-

tion on Terrain trees uses 40% less memory and half the time compared to the compact state-of-art connectivity-based data structure for TINs. Furthermore, the parallel simplification algorithm on the Terrain trees exhibits a 12x speedup with an OpenMP implementation. When evaluating the quality of output meshes, the meshes from the simplification on Terrain trees have a slightly worse approximation quality compared to the baseline data structure when the same number of edges are contracted. Thus, we also propose an optimized simplification strategy on Terrain trees that can effectively reduce such a quality difference. The study of the simplification algorithm and its computational performance was published as a 12-page conference paper at ACM SIGSPATIAL in 2021 [171]. The full research findings were published in the ACM Transactions on Spatial Algorithms and Systems in 2024 [172].

Chapter 5 addresses research question 3, i.e., the problem of sea ice topographic feature mapping on TIN-based surface models. In response to question 3-1, we propose a method to generate an accurate surface model from airborne LiDAR points by using the Delaunay triangulation and the α-shape method. To address question 3-2, several methods to extract topographic features are developed. Firstly, we propose methods to calculate statistical measures, such as point density and surface roughness, on TINs. By comparing the roughness results when different radii are used, we find that the measure scale of roughness computation can significantly affect the interpretation of roughness values. Roughness calculated with smaller radii (5m/10m) highlights important topographic features, including ridges and hummocks and can be used in the identification of such features. Roughness calculated with larger radii (100m/150m) averages out the small features and can be used to locate regions with major ridges or extremely varied terrain. These findings address sub-question 3-3 as well. Secondly, we propose a method to extract regions without LiDAR measurements from TINs by using the α-shape method. The ex-

tracted regions help to prove the conclusions from previous studies that the OIB ATM wide-scan data [176] has data dropout over water surfaces. Lastly, we design a novel method to extract pressure ridge structure lines from TINs, addressing sub-question 3-4. The proposed method extracts ridge structure lines as parts of geometries of the arcs from a maximum to connected saddles in a surface network by using the topology-based methods introduced in Chapter 3 and the topological simplification method introduced in Chapter 4. In this study, the Terrain trees data structure, introduced in Chapter 3, is used to store TINs and to conduct topographic analysis on TINs. The calculations of density and roughness and the extraction of ridge structure lines are implemented by extending exiting functions in TTL. The manuscript on this work is currently under preparation and is planned to be submitted to IEEE Transactions on Geoscience and Remote Sensing [169].

6.2 Additional Research Contributions

During my PhD study, I have contributed to two other research projects besides the dissertation research.

Subsection 6.2.1 introduces the first project, which is a collaborative work with Haoan Feng from Department of Computer Science at UMD. We develop a scale-space method to identify and track critical points from TINs. Subsection 6.2.2 introduces the second project is a collaborative work with Yuehui Qian from Department of Geographical Sciences. In this work, we propose a distributed framework for efficient computation of topology-based terrain segmentation using Apache Spark [198].

6.2.1 Critical Features Tracking on Irregular Terrain Meshes by a Scale-Space Method

Similar to the discrete Morse theory [76] reviewed in Section 2.1.2.2, the *piece-wise linear (PL) Morse theory* aims to extend the results of Morse theory to discrete data. Given a terrain approximated with a TIN Θ, PL Morse theory defines a PL function on each vertex in Θ [13], whereas Discrete Morse Theory extends the concepts in Morse theory to all simplices by creating a collection of gradient vectors to simulate the Morse function. Consequently, the definition of critical points of a Morse function corresponds to different concepts in two theories. Critical points defined in the PL Morse theory are always located at vertices of Σ, while discrete Morse theory defines critical simplices, which can be vertices, edges, or triangles of Σ. An in-depth comparison between the PL Morse theory and discrete Morse theory was conducted by Fugacci et al. [80]. It proves critical simplices defined in discrete Morse theory appear in correspondence of critical points defined in PL Morse theory [13, 56]: each critical point is necessarily a vertex of the corresponding critical simplex.

As shown in Fig. 6.1, a vertex *v* in a TIN Θ can be classified into one of five categories according to the PL Morse theory: maximum, minimum, regular, saddle, and k-fold saddle. *v* is a *maximum* when it has a higher elevation than all vertices in its Vertex-Vertex (VV) relations and it is a *minimum* when it has a lower elevation than all of them. We sort all vertices in the VV relations of *v* in clockwise-order $VV_{sorted} : [v_0, v_1, ..., v_n]$. For each vertex in VV_{sorted}, we compare its elevation against the elevation of *v* and the store the result to a list $VV_{rel} : [rel_0, rel_1, ...rel_n]$. Continuously lower/higher values in VV_{rel} are grouped together, and we call the number of lower groups k_{lower} and the number of higher groups k_{higher}. *v* is called a *k-fold saddle* when

$k_{higher} = k_{lower} = k + 1$. Specially, when $k = 1$, v is a simple *saddle*. When v does not fit any of the mentioned cases, it is a *regular* point.

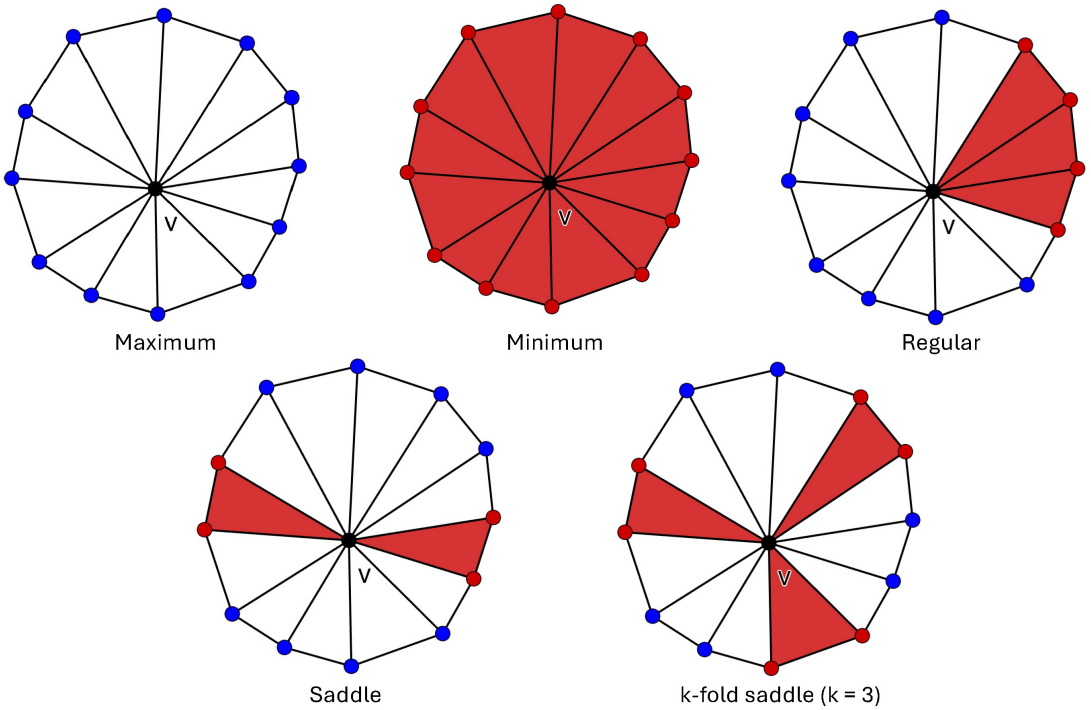


Figure 6.1: Critical point classes according to the piece-wise linear (PL) Morse theory [13]. The labels indicate the type of the central vertex v. Among vertices in the Vertex-Vertex relation of v, red vertices have lower elevations than v and blue vertices have higher elevations than v.

In this work, we focus on a scale-space method to identify and track critical points from TINs. A scale-space of a TIN Θ is generated through a sequence of smoothing operations on Θ. As the scale increases, some critical points disappear from Θ, i.e., they become regular points. By tracking the change of a critical point p, we can estimate the number of scales that p persists, which we refer to as the *life-span*. By identifying critical points with high life-spans, we are able to locate significant critical points, i.e., those that persist after smoothing. Such critical points usually correspond to topologically significant features and can be used in the generation of nautical charts [116] or the selection of spot heights [156]. A series of experiments have been

conducted to evaluate the proposed method, focusing on the accuracy of results compared to a scale-space method using gridded DEMs [149], the selection of life-span thresholds for spot height selection, and the robustness of resolution. Specifically, the experiment on the robustness of resolution evaluates how the resolution of the original TIN before smoothing impacts the accuracy of spot height selection, along with a comparison against the method based on gridded DEMs [156]. The results demonstrate that the proposed TIN-based method more effectively handle flat area cases and exhibits better resolution robustness compared to the method based on gridded DEMs.

My contributions to this work include participating in the design of the smoothing method for TINs and the scale-space method, as well as contributing to the experimental design. Based on this work, a manuscript is in preparation [75] and we plan to submit it to the 2024 ACM SIGSPATIAL conference.

6.2.2 Topology-based Terrain Segmentation Using Apache Spark

Distributed computing frameworks such as Apache Spark [198] or MPI [30] have revolutionized data processing by enabling scalable and efficient management of very large datasets. While Terrain trees have been proven to be the most efficient data structure for TINs, they are designed for single-machine use and cannot directly be extended to distributed computing frameworks due to the existence of two global arrays.

In this work, we develop a new distributed framework, named *Morse-Spark*, for handling TINs, with a focus on its application to topology-based terrain segmentation using Apache Spark [198]. The framework consists of a distributed data representation that stores the minimal

information for representing a TIN, a new compact representation for encoding a TIN topology in Spark, a new distributed algorithm for computing the Forman gradient field based on discrete Morse theory [76], and algorithms to calculate topological features introduced in Section 2.1.2.2, including critical net and regions of influence of critical simplices.

The encoding of a TIN Θ involves two DataFrames [8] storing the vertices and triangles of Θ. These are the minimal necessary information for extracting other connectivity relations and for TIN navigation. Based on this encoding, all other connectivity relations can be extracted. Depending on the application, different connectivity relations are required, and thus, there is no single optimal strategy to extract connectivity relations on this distributed data representation. In the computation of terrain segmentation, Vertex-Triangle, Vertex-Vertex, Vertex-Edge, Edge-Vertex, and Edge-Triangle relations are required. Therefore, in this work, we discuss the best order and strategy to extract such relations.

Based on the distributed data structure for TINs, we design an algorithm to compute the Forman gradient field using the homotopy expansion method [155] on Apache Spark. We also design methods to calculate the critical net and the influence regions of critical simplices by utilizing two auxiliary data structures encoded as GraphFrames [41].

By utilizing the scalability of a Spark cluster, our framework can easily process TINs consisting of more than 2 billion vertices. The computation of the Forman gradient field on Morse-Spark can achieve a 2.5-3.5x speedup compared to the same computation on the native Spark framework. The Morse-Spark framework has comparable performance when computing the influence regions and a 25x speedup compared to the native Spark in the computation of the critical net. Our experiments also show that the performance can be consistently improved by increasing the number of executors used in the cluster, which enable the processing of even larger datasets.

Specifically, my contribution to this work includes participating in the design of the data representation, the strategy for connectivity relations, the method to calculate the Forman gradient field, and methods to extract topological features. I have also contributed to the design of the experiments. Based on this work, a manuscript is in preparation [152], and we plan to submit it to International Journal of Geographical Information Science (IJGIS).

6.3 Future research

The experiments with Terrain trees demonstrate their compactness and efficiency on a single machine. As part of our future work, we plan to distribute the global arrays across the leaf blocks of the Terrain trees. This design would reduce the message exchanges among blocks when a leaf block is processed, which is important for distributed computing. One of our initial efforts in this direction is the study on Spark-based terrain segmentation, as introduced in Section 6.2. In this study, we designed a Spark-based data representation for TINs that stores only the minimal necessary geometric information while still supporting efficient retrieval of connectivity relations. Building on this foundation, we plan to develop new distributed terrain analysis methods for the Spark-based representation.

Furthermore, while storing a complete TIN is beneficial for global operations, it is not necessary for local operations. Consequently, we also plan to design an algorithm that computes the TIN at runtime, locally within each leaf block, and discards it when no longer needed. This approach will further reduce the memory footprint and enable the processing of even larger point clouds. A challenge we need to address is ensuring that the TIN computed locally within a leaf block is consistent with the subset of a complete TIN computed on the input point set.

In Chapter 4, we introduced sequential and parallel methods for topology-aware mesh simplification. In the future, we plan to extend these methods to also support generic updates to the TIN, like adding new points or removing TIN vertices. The generic mechanism to update the Terrain tree is like the process performed after a simplification, but it might be more challenging if larger portions of the TIN need to be re-triangulated.

Some applications require maintaining the triangle shape quality while simplifying the mesh. Especially in terrain analysis, a mesh with good triangle shape quality is required for subsequent interpolation or simulation. Future work can explore incorporating a triangle shape constraint into the gradient-aware edge contraction operator to ensure good triangle shape quality after simplification. We also plan to investigate optimized parameters to simplify meshes for coastal ocean modeling applications, especially in storm surge and tide simulation [15].

Our current parallel strategy is compatible with shared-memory processing using OpenMP [37]. In the future, we plan to explore whether the use of specialized compilers, such as Intel Implicit SPMD Program Compiler (ISPC) [4], and libraries like Intel oneAPI Threading Building Blocks (TBB) [5], can enhance its efficiency. Besides, a natural extension of our simplification method is to support large-scale data processing through the distributed-memory processing strategy based on MPI [30].

While our study described in Chapter 5 demonstrates the potential of using TIN-based models to extract sea ice topographic features, there is room for further research to enhance these results and uncover deeper insights. For instance, our proposed method successfully extracts the structure lines of ridges. The irregular decomposition employed by TINs opens the possibilities for extending from structure lines to precise ridge ranges in the future. Such advancements would facilitate the generation of 3D models of ridges, contributing to the understanding of sea ice

morphology and processes.

Moreover, drift correction is a crucial step in sea ice data analysis due to the movement of sea ice floes. Currently, the drift rate is usually measured in situ by tracking the movement of buoys [153] or estimated by comparing the locations of identical sea-ice features (e.g., ice floes and pressure ridges) in data collected at different times [35, 185]. Automated methods have been proposed to detect sea ice features in images and track their movements [48, 94]. However, research on tracking such features from ATM data is limited. TIN-base sea ice surface models allow us to extract the structures of sea ice features. Therefore, the next step of our research involves generating TIN-based sea ice surface models from ATM datasets collected at different times and estimating sea ice floe drift rate by matching sea ice features extracted from these models.

Although in this dissertation, we only discussed the application of Terrain trees and terrain analysis methods developed on Terrain trees in the field of sea ice, these techniques are not limited to any specific field of application. In the future, we also plan to explore the possibility of applying these techniques in other fields, such as bathymetric modeling and flood simulation.

Appendix A: Supplementary material

A.1 Letter to include co-authored paper

Dr. Stephan Roth, Dean

The Graduate School

University of Maryland

Dear Dr. Roth,

I hope this letter finds you well. My name is Yunting Song, and I am a PhD student in the Department of Geographical Sciences. I am writing to request your approval for including a co-authored work in my dissertation.

The paper titled "Terrain trees: a framework for representing, analyzing and visualizing triangulated terrains" has been published in Springer International Journal on Advances of Computer Science for Geographic Information Systems (GeoInformatica) in 2023. This is a collaborative work involving extensive algorithmic design and software development. The first three authors are listed in alphabetical order and have contributed equally to the paper, while the fourth author is the senior author and advisor.

The examining committee has agreed that I have made a substantial contribution to this work, and I have received approval to include it from the dissertation advisor and the Graduate

Director of our program, Dr. De Floriani. Thank you very much for your attention to this matter.

Sincerely,

Yunting Song

A.2 Chapter 3 - Additional results

In this section, we list the experiment results of the work presented in Chapter 3 but are not shown in the content of Chapter 3.

Figs. A.1, A.3, A.7, A.5, A.9, A.11 show storage costs for storing the index of the Terrain trees on datasets in Table 3.1 except for the GREAT SMOKEY MOUNTAIN dataset. Figs. A.2, A.4, A.8, A.6, A.10, A.12 show how different capacities affect the extraction of the Vertex-Triangle (VT) relation on datasets in Table 3.1 except for the GREAT SMOKEY MOUNTAIN dataset.

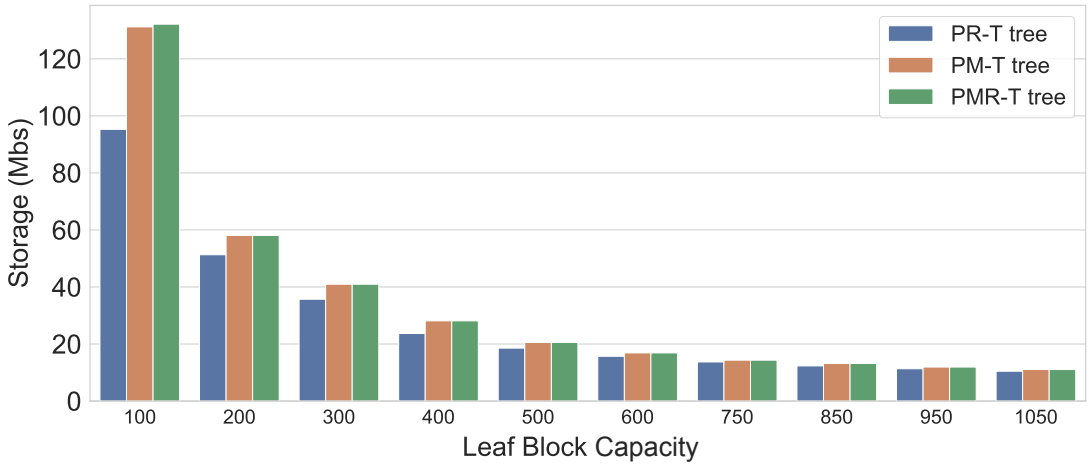
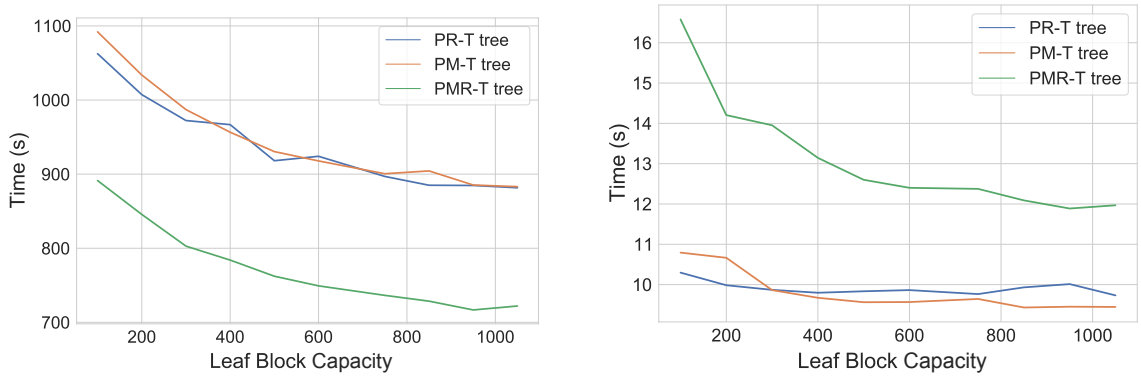


Figure A.1: The storage costs for storing the hierarchical index of the Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.2: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

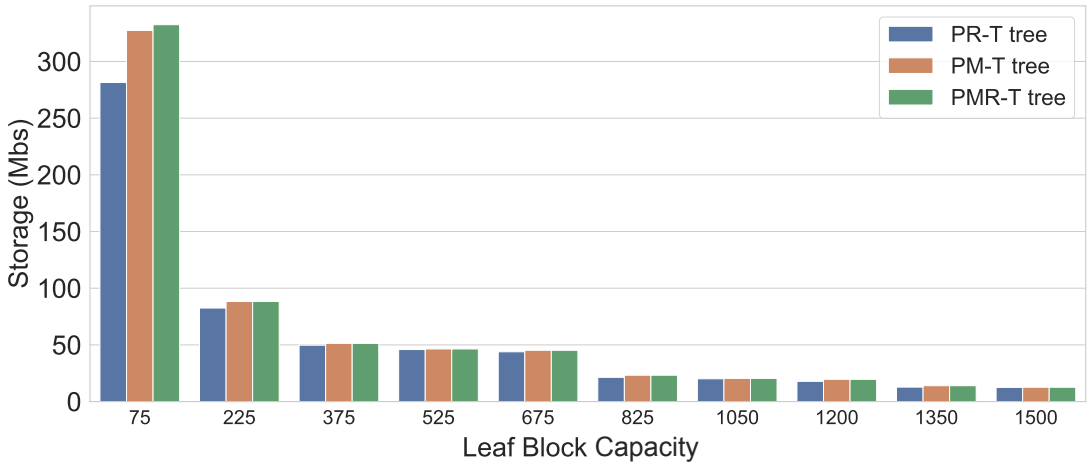
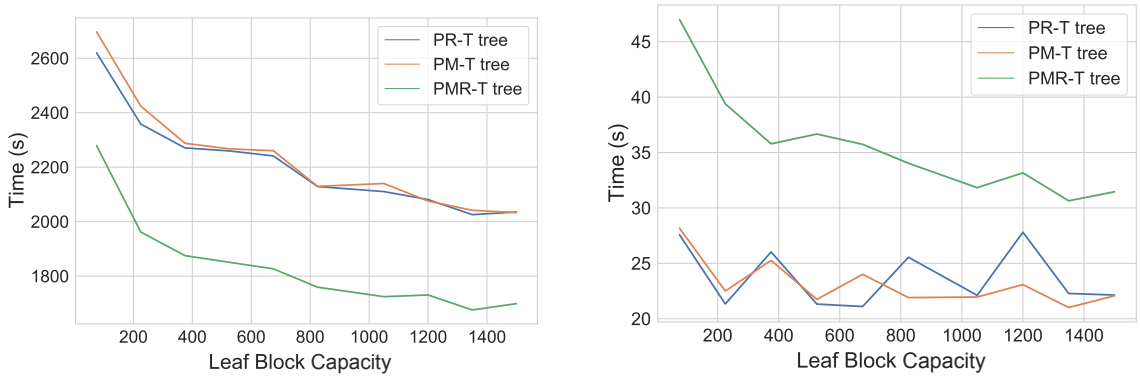


Figure A.3: The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.4: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

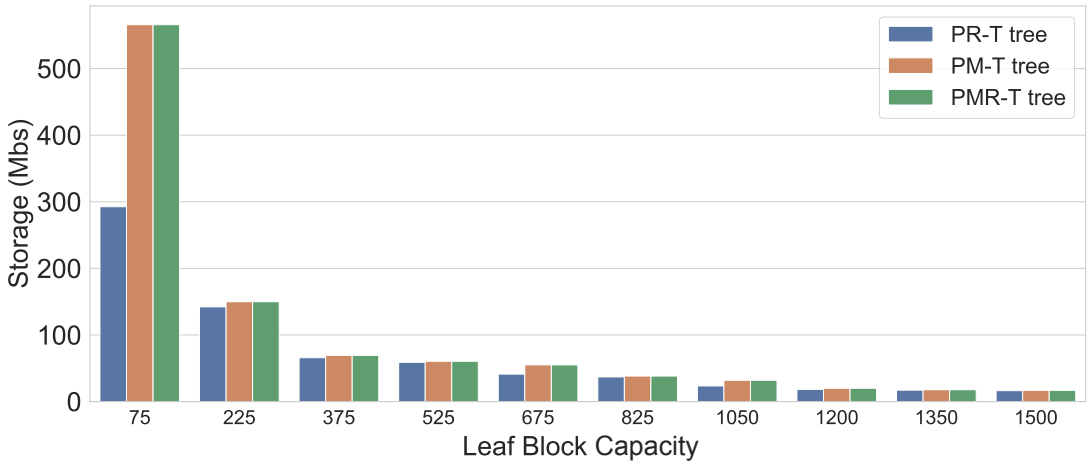
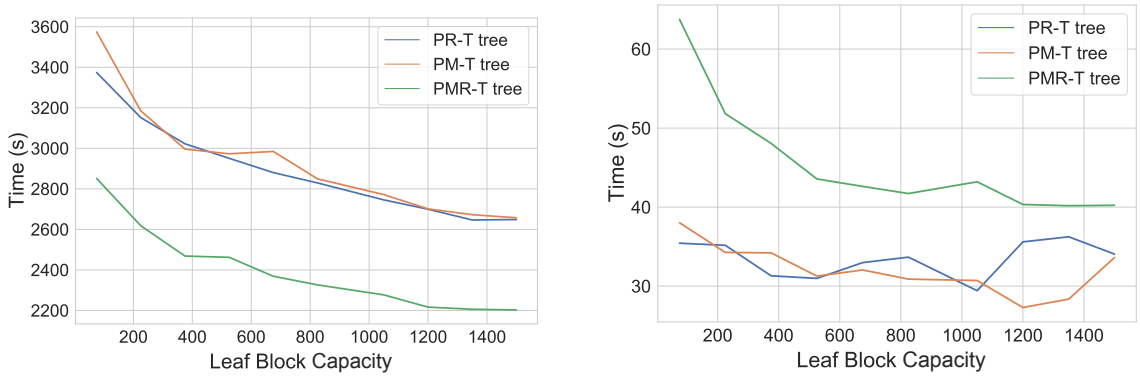


Figure A.5: The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.6: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

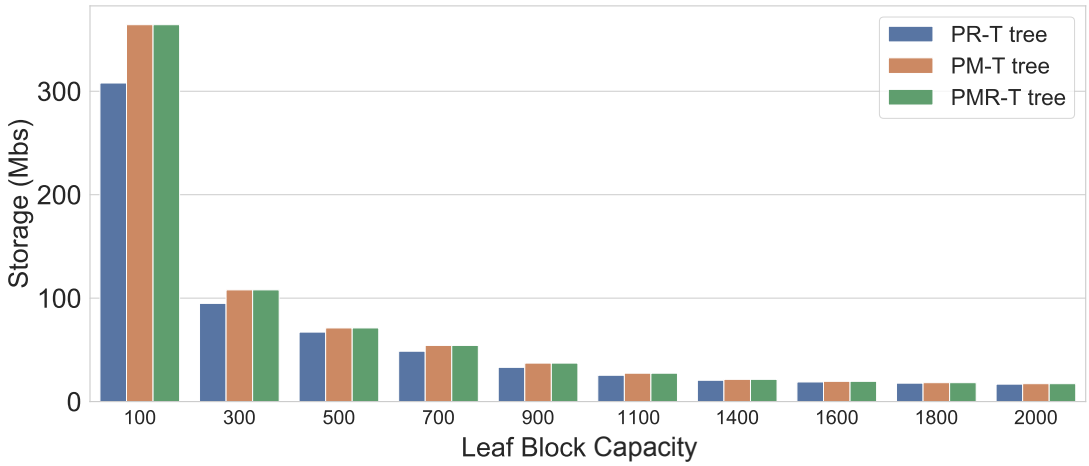
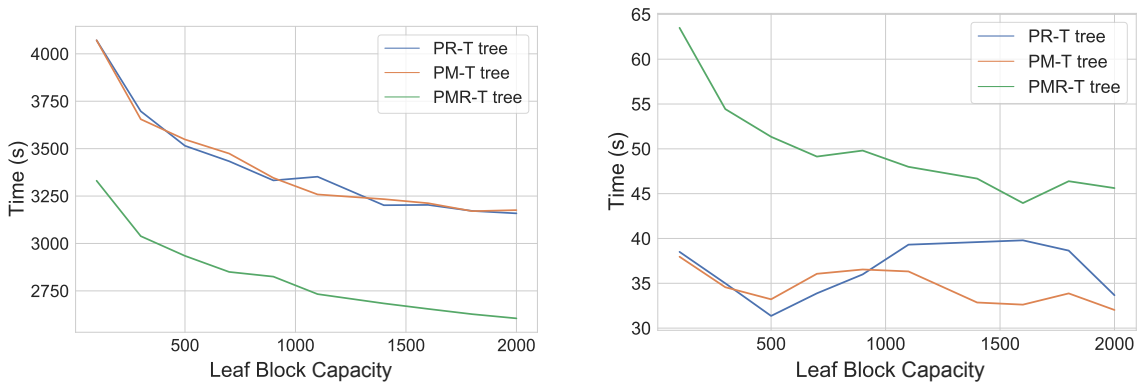


Figure A.7: The storage costs for storing the hierarchical index of the Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.8: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

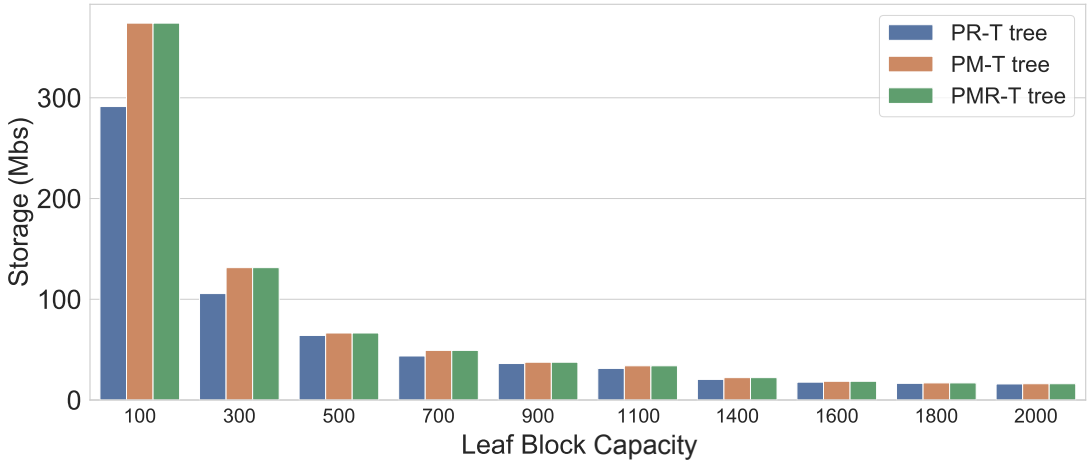
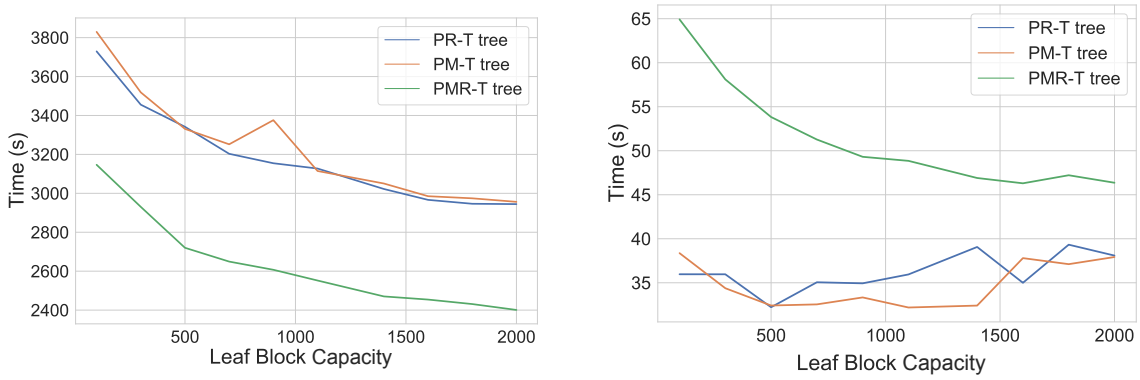


Figure A.9: The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.10: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

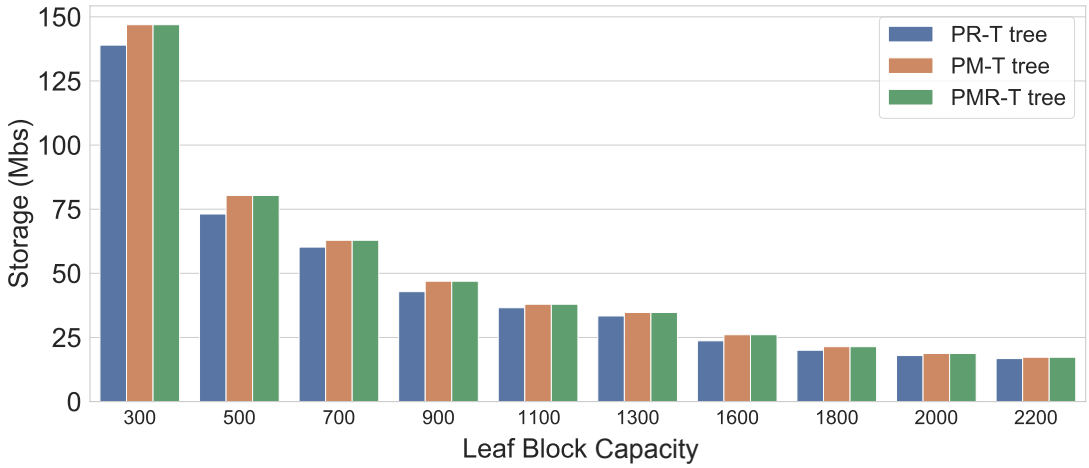
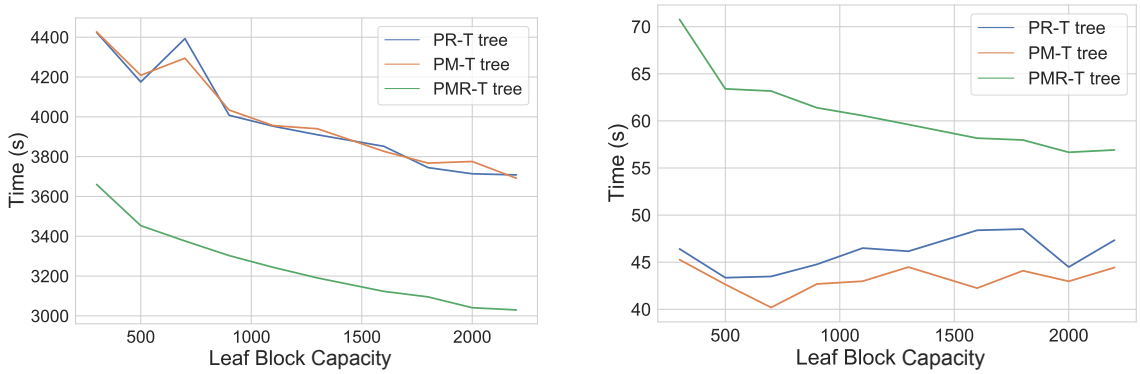


Figure A.11: The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the capacity value on the vertices.



(a) Generation time

(b) VT extraction time

Figure A.12: Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the vertex capacity value.

A.3 Chapter 4 - Computing the quadric error matrix on Terrain trees

The Quadric Error Metric (QEM) [83] defines the error at one vertex v of a triangle mesh Σ is as the sum of the squared distances to the planes of the triangles incident in v. Specifically, the error at v with respect to a plane Π is calculated as $\Delta_{\Pi}(v) = v^T K_{\Pi} v$, where K_{Π} is a 4×4 matrix called the *fundamental error quadric*. The overall error at v can be represented as $\Delta(v) = v^T Q_v v$. Q_v is called the *initial error quadric* at v, and it is the sum of the fundamental error quadric with respect to the plane defined by each triangle incident in v. The *cost*, or error, introduced by contracting edge $e = \{v_1, v_2\}$ is defined as $\Delta(v) = v^T(Q_1 + Q_2)v$, where Q_1 and Q_2 are the initial error quadrics at v_1 and v_2, respectively. The quadric error of v_2 is accumulated to v_1 when *e* is contracted to v_1. Therefore, the cost of *e* reflects the change from the original mesh to the approximation after the contraction of *e*.

In each leaf block *b*, the quadric error matrices *E* of vertices in *b* are computed during traversal of its triangle list. For each triangle t in *b*:

1. check if at least one vertex of t is contained in b. If not, skip t, otherwise, proceed to step (2);

2. calculate the fundamental error quadric K_Π of the plane on which t lies;

3. for each vertex v of t, if v is contained in b, add K_Π to its initial error quadric $E[v]$.

Note that the fundamental error quadric associated with a triangle may be computed more than once if its vertices are in different leaf blocks. This will slightly increase the computation time compared to traversing through the global triangle array Σ_T of Σ and calculating the corresponding fundamental error quadrics. However, the computation of the initial error quadrics at the vertices in different leaf blocks are completely independent and fully local to each leaf block, making it possible to compute the quadric error matrices of Σ in parallel.

A.4 Chapter 4 - Performing an edge contraction

Algorithm 3 CONTRACT(e, VT(ν_j), ET(e), E, Σ)

Input:
$e = \{\nu_i, \nu_j\}$: edge to be contracted to ν_i
$VT(\nu_j)$: the Vertex-Triangle relation of ν_j
$ET(e)$: the Edge-Triangle relation of e
E: the array of vertex error quadrics
Σ: the underlying triangle mesh of a TIN

1: **for each** t in $ET(e)$ **do**
2: $\quad \Sigma \leftarrow \Sigma - \{t\}$ // Remove t from Σ
3: **end for**
// For each triangle t incident in ν_j but not adjacent to e
4: **for each** t in $(VT(\nu_j) - ET(e))$ **do**
5: $\quad t \leftarrow (t - \nu_j) \cup \nu_i$ // Replace ν_j with ν_i in t
6: **end for**
7: $E[i] \leftarrow E[i] + E[j]$ // Update the error quadric at vertex ν_i
8: $\Sigma \leftarrow \Sigma - \{\nu_j\}$ // Remove ν_j from Σ

In this section, we describe how an edge contraction is performed. Algorithm 3 depicts the CONTRACTION operation at row 15 of Algorithm 1. The algorithm removes the two triangles adjacent to e and vertex ν_j (row 2 and row 8). In each remaining triangle in $VT(\nu_j)$, it replaces ν_j with ν_i (row 4 to 6). After the contraction, the error quadric of the remaining vertex ν_i is updated by adding the quadric of ν_j to it (row 7).

A.5 Chapter 4 - Experiments on leaf capacity selection

Table A.1: Time (in minutes) (denoted as **T**), peak memory usage (in Gigabytes) (denoted as **M**), and compression rate (in %) (denoted as **R**) of simplification when using different capacity values (denoted as **C**) in the Terrain tree generation. The capacity value in bold face is the capacity value used to generate the Terrain tree of each dataset.

Molokai				Great Smokey Mountains				Canyon Lake				Yosemite Rim Fire				Dragons Back Ridge				Moscow Mountain			
C	**T**	**M**	**R**	**C**	**T**	**M**	**R**	**C**	**T**	**M**	**R**	**C**	**T**	**M**	**R**	**C**	**T**	**M**	**R**	**C**	**T**	**M**	**R**
300	36.3	12.8	73.1	300	44.8	17.4	74.7	450	70.9	24.9	74.6	600	104.2	39.5	70.6	600	117.3	46.3	80.4	900	186.4	57.2	77.5
400	35.1	12.8	73.2	400	45.2	17.3	74.9	600	68.7	24.7	74.6	800	99.1	39.4	70.6	900	100.5	46.2	80.4	1200	185.7	57.6	77.5
500	29.5	12.7	73.3	500	39.4	17.3	74.9	750	68.8	24.9	74.6	**1000**	100.1	39.0	70.6	**1200**	99.8	45.9	80.5	1500	192.9	57.1	77.6
600	29.0	12.8	73.3	600	41.9	17.3	75.0	**900**	65.1	24.6	74.7	1200	100.5	39.4	70.6	1500	129.2	46.0	80.6	1800	188.7	57.4	77.6
700	29.0	12.7	73.3	700	39.8	17.3	75.0	1050	68.4	24.9	74.7	1400	100.3	39.4	70.6	1800	126.7	45.9	80.6	2100	172.0	57.5	77.6
800	29.5	12.7	73.4	**800**	39.1	17.2	75.0	1200	66.5	24.6	74.7	1600	100.9	39.3	70.6	2100	127.9	46.1	80.6	**2400**	170.9	57.4	77.7
900	29.1	12.7	73.4	900	39.4	17.3	75.0	1350	66.7	24.8	74.7	1800	102.2	39.2	70.6	2400	131.7	45.7	80.6	2700	176.9	57.3	77.7
1000	29.2	12.7	73.4	1000	39.4	17.3	75.0	1500	73.2	24.8	74.8	2000	106.7	38.9	70.7	2700	131.8	46.1	80.6	3000	176.3	56.9	77.7
1100	29.2	12.8	73.4	1100	39.6	17.4	75.0	1650	75.5	24.7	74.8	2200	111.4	39.3	70.7	3000	129.8	46.1	80.6	3300	177.2	57.3	77.7
1200	29.8	12.7	73.4	1200	39.5	17.3	75.1	1800	73.3	24.6	74.8	2400	106.8	39.2	70.7	3300	129.6	46.0	80.6	3600	173.9	57.2	77.7

In this section, we evaluate the performance of the simplification algorithm with different capacities on Terrain trees. Recall that a capacity defines the maximum number of vertices that each leaf block can contain. We established an initial range for capacity values between $1/100000$ and $1/30000$ of the total number of vertices in the data set. This is in order to have coarser hierarchical subdivisions, usually beneficial for tasks requiring intense navigation of the hierarchy. Within this range, we selected ten different capacity values for each dataset and compared the performance in sequentially simplifying the meshes encoded by the resulting Terrain tree. Our comparisons show that the memory footprint and the compression rate do not change

significantly when using different capacity values (up to 1.7%). Also, simplification times are highly dataset-dependent, and the best performances are achieved with values in the middle of the tested range.

Table A.1 shows the performances of sequential topology-aware mesh simplification on the Terrain tree. The memory footprint does not change significantly when using different leaf capacities. The same holds for the percentage of edges contracted. Depending on the dataset, timings may vary. For example, on the Molokai dataset, the simplification is 21% faster when a shallower hierarchy (larger capacity) is used, while on *Dragons Back Ridge*, using a deeper hierarchy (smaller capacity) reduces the simplification time by 24%. The simplification time is stable when the capacity value varies in a small range. Overall, the results show that even selecting a sub-optimal capacity for generating a Terrain Tree, the algorithm's performance is not severely affected, and it still performs well. For the experiments presented in the paper, we use only one capacity value for each dataset. Generally, we use the capacity value that results in the shortest simplification time. But when the variation in time is small (less than 1%), we also consider the memory cost and the compression rate. The capacity value selected for each dataset is denoted in Table A.1 in bold face.

A.6 Chapter 4 - Additional results

In this section, we list the experiment results of the work presented in Chapter 4 but are not shown in the content of Chapter 4.

Fig. A.13 shows the time for updating the Terrain trees after simplifying the same mesh with different cost thresholds using the *Yosemite Rim*, *Dragons Back Ridge*, and *Moscow Moun-*

tain datasets. Fig. A.14 shows the average triangle shape measure of simplified meshes obtained from different topology-aware simplification methods using the *Molokai* dataset, the *Great Smokey Mountains* dataset, and the *Canyon Lake* dataset.

In Fig. A.15, we show how the number of critical simplices changes when the mesh is simplified by the Geometric-TT method in four smaller datasets in Table 4.1, i.e., the *Molokai* dataset, the *Great Smokey Mountains* dataset, the *Canyon Lake* dataset, and the *Yosemite Rim* dataset. Figs. A.16 and A.17 show the simplification time and quality of the simplified meshes using the IA-based method, the Para-TT method, and the Pro-Para-TT method introduced in Section 4.8. The *Molokai* dataset and the *Canyon Lake* dataset are used in this experiment.

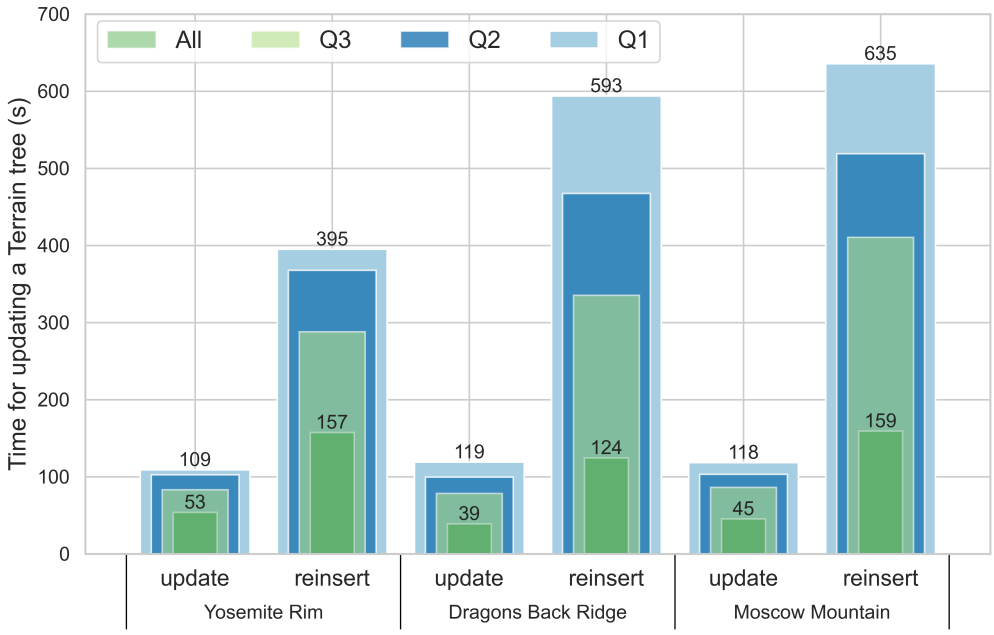


Figure A.13: The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original **Q1**, **Q2**, and **Q3** represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. **All** represents the result when contracting all contractible edges without a cost threshold. The text labels on the boxes in figure indicate the time when **Q1** cost threshold is used and when **All** edges are contracted.

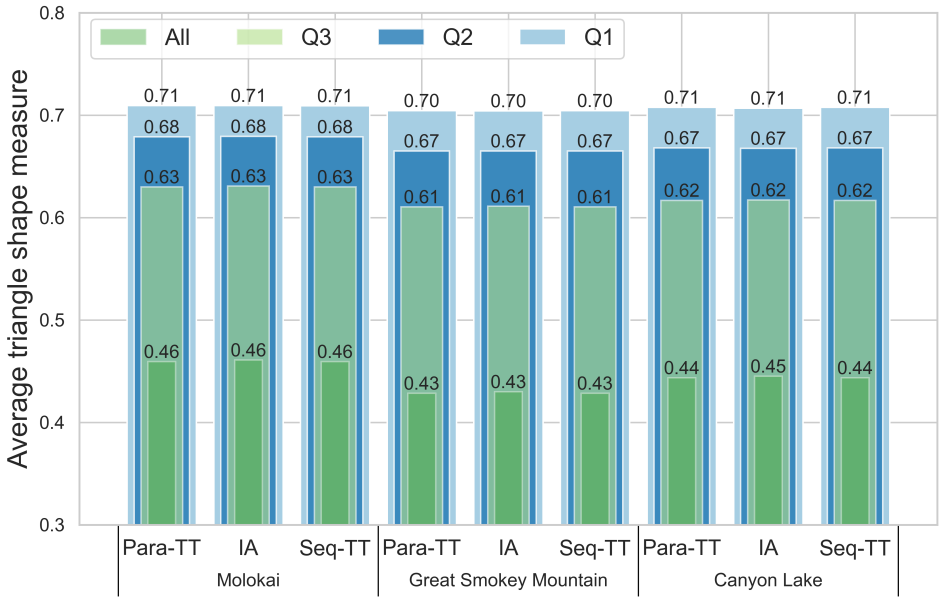
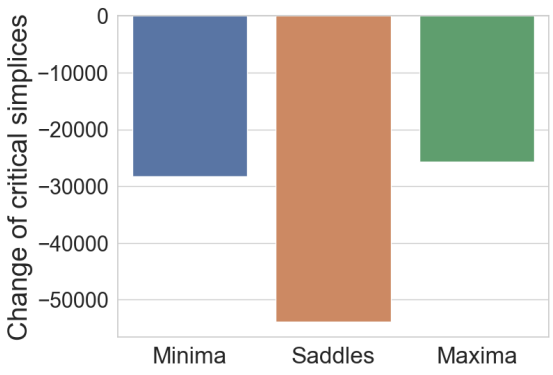
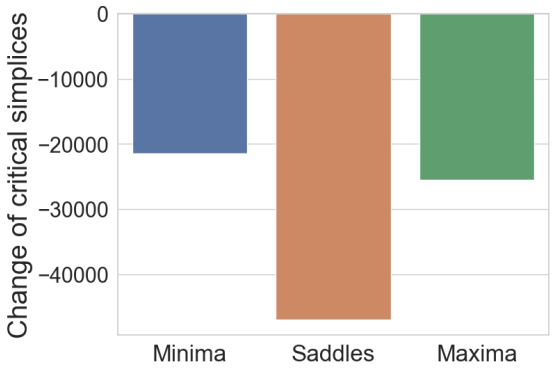


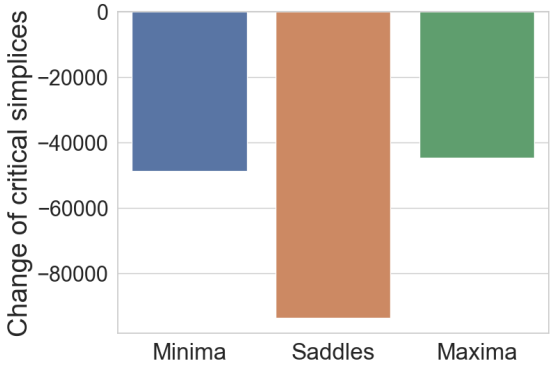
Figure A.14: The average triangle shape measure of meshes obtained from three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (**Q1**, **Q2**, and **Q3**) and when all contractible edges are contracted (**All**).



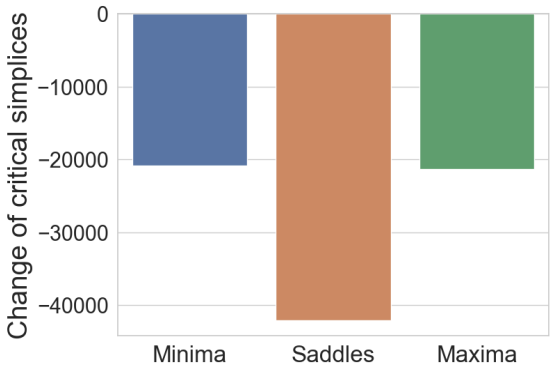
(a) Molokai



(b) Great Smokey Mountains

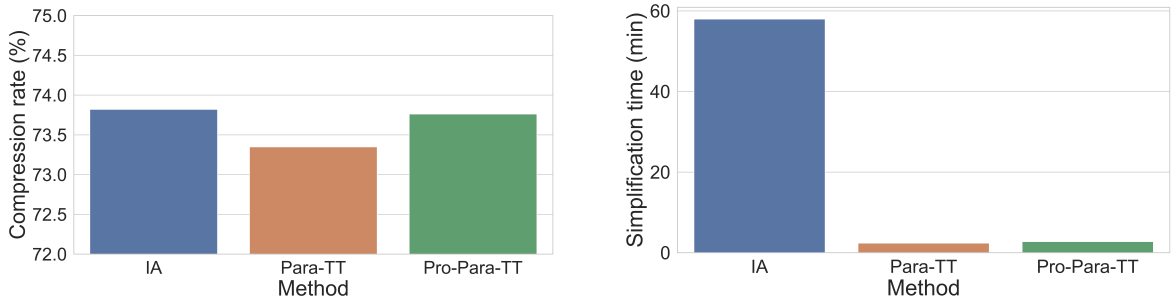


(c) Canyon Lake



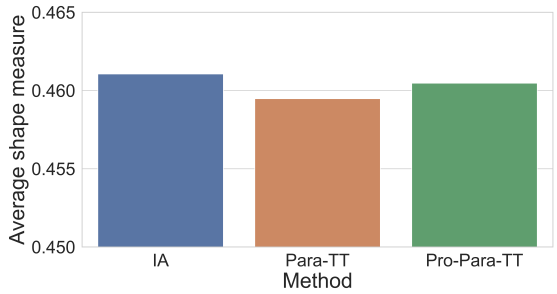
(d) Yosemite Rim

Figure A.15: The change in the number of critical simplices in each dataset when it is simplified with the Geometric-TT method.

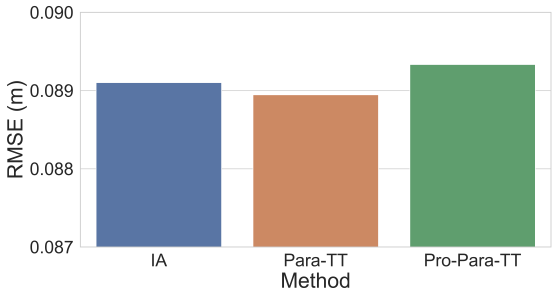


(a) Compression rate

(b) Simplification time

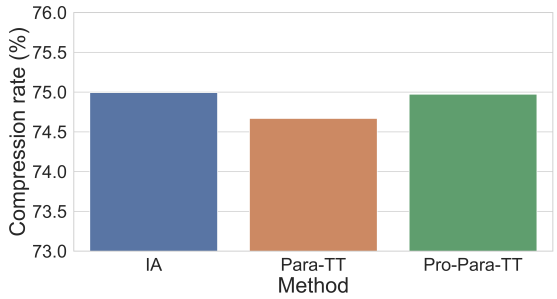


(c) Average shape measure

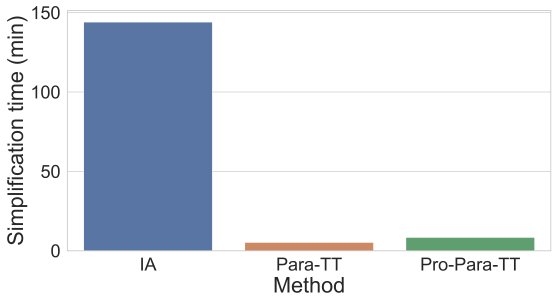


(d) RMSE of vertical distances

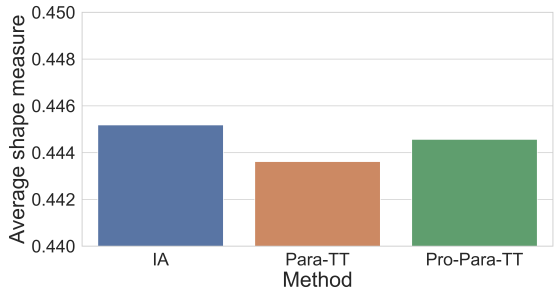
Figure A.16: The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Molokai* dataset.



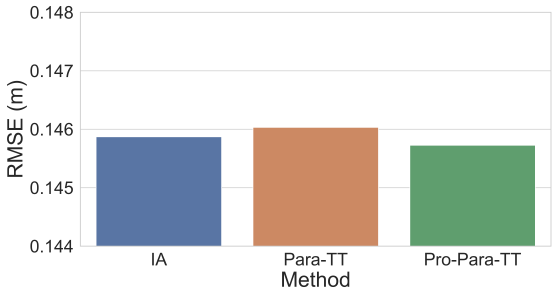
(a) Compression rate



(b) Simplification time



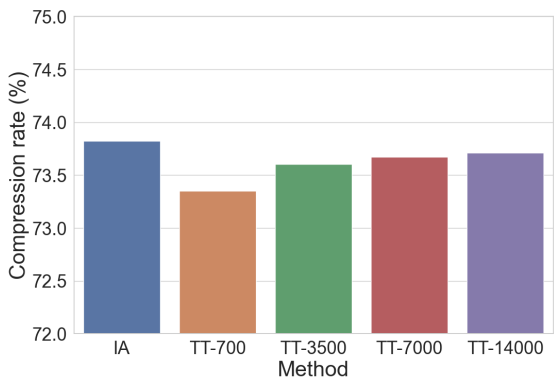
(c) Average shape measure



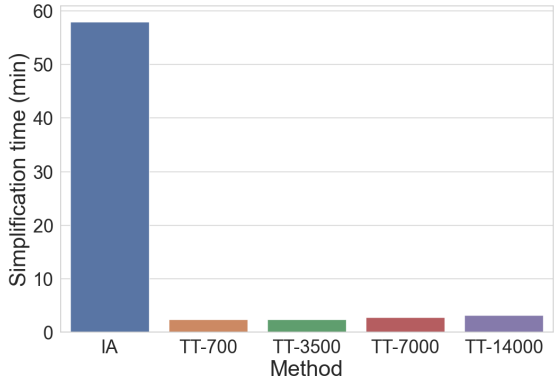
(d) RMSE of vertical distances

Figure A.17: The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Canyon Lake* dataset.

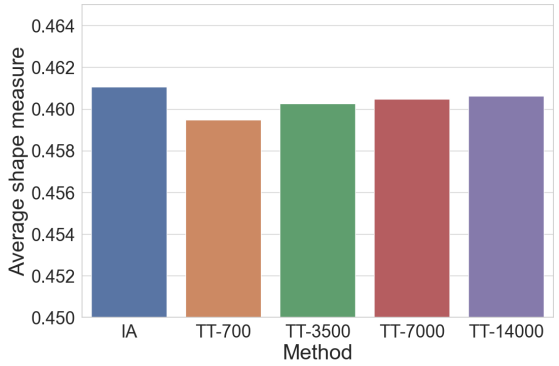
Figs. A.18 and A.19 show the simplification time and several quality measures when different leaf capacity values are used for generating the Terrain trees on the *Molokai* dataset and the *Canyon Lake* dataset, respectively.



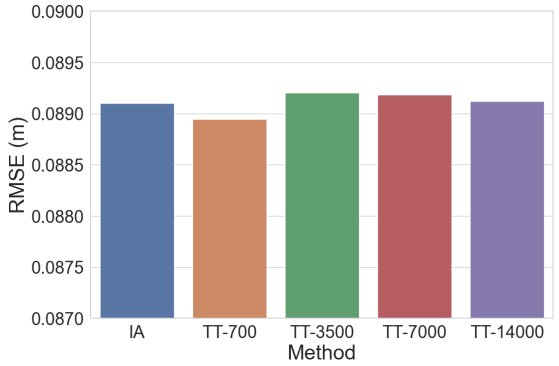
(a) Compression rate



(b) Simplification time

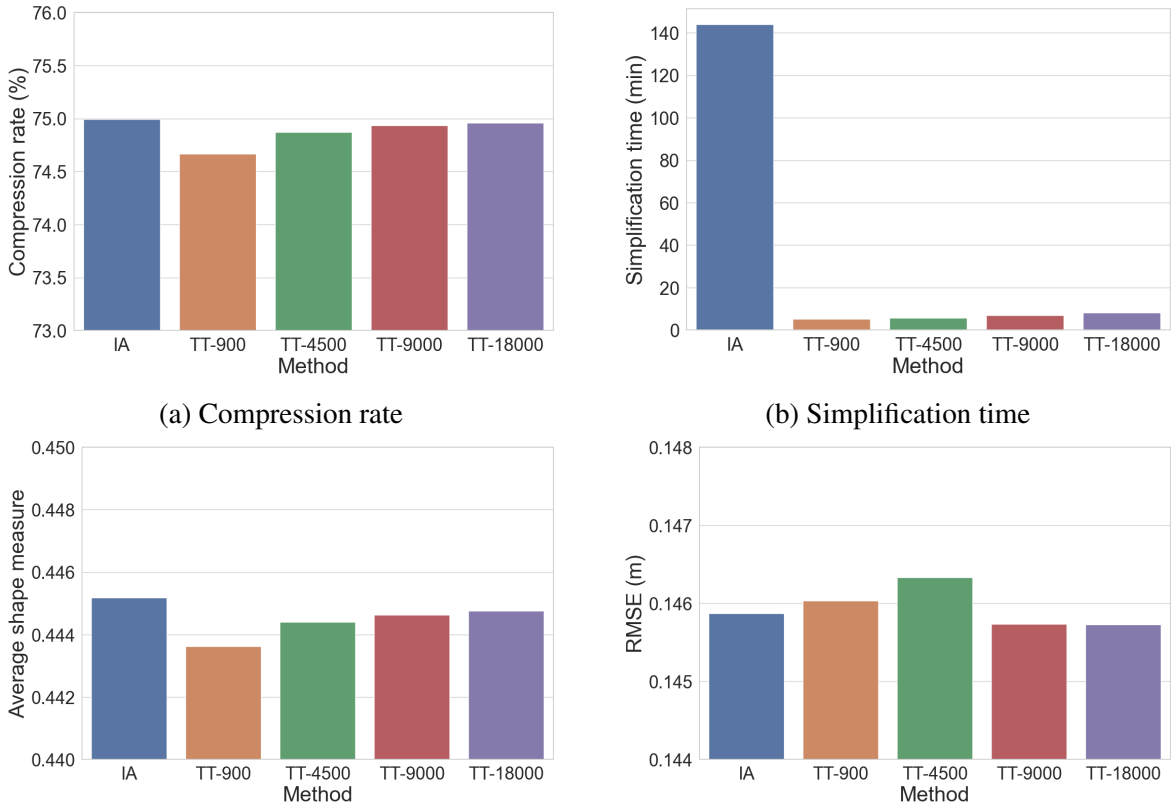


(c) Average shape measure



(d) RMSE of vertical distances

Figure A.18: The performance and quality evaluation of the simplification of the *Molokai* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds leaf capacity value used during the Terrain tree generation.



(a) Compression rate

(b) Simplification time

(c) Average shape measure

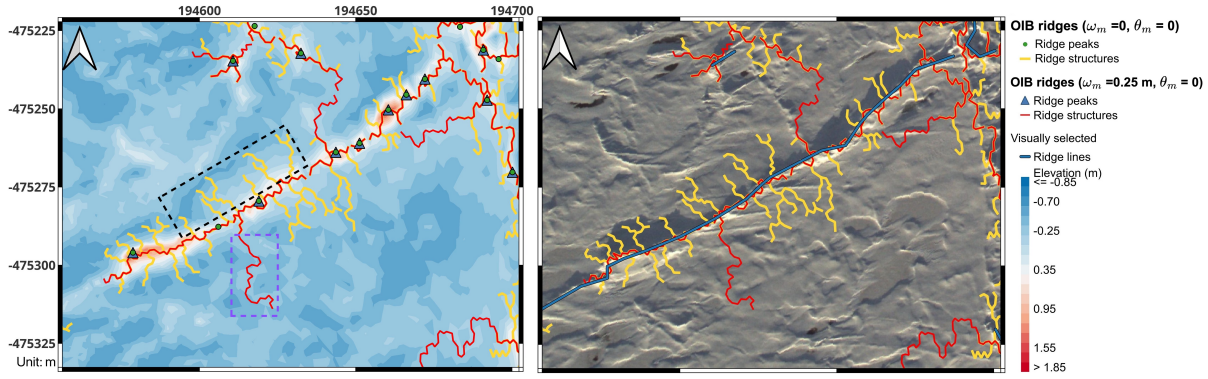
(d) RMSE of vertical distances

Figure A.19: The performance and quality evaluation of the simplification of the *Canyon Lake* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds leaf capacity value used during the Terrain tree generation.

A.7 Chapter 5 - Influence of parameters on ridge extraction results

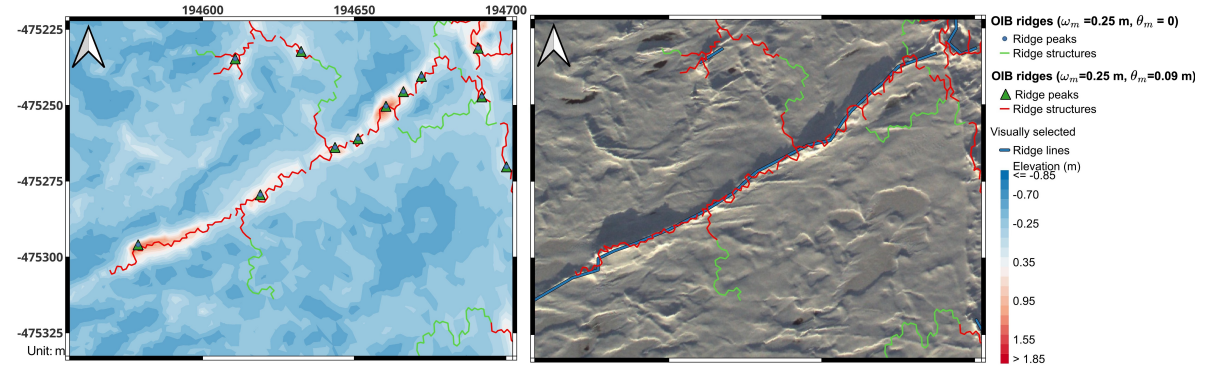
In this section, we discuss how topological simplification and roughness filter affects the ridge extraction results by comparing the results obtained with optimal parameters selected in Section 5.6.3 and results obtained with other sets of parameters.

Fig. A.20 compares the results when the persistence threshold ω_m is set to 0 (depicted by yellow lines) and when it is set to 0.25 m (depicted by red lines) during ridge extraction. In both cases, the roughness filter is not applied. The topological simplification has two different



(a) (b)

Figure A.20: Comparison between the ridge extraction results when the persistence threshold ω_m = 0 m, the persistence threshold θ_m = 0 m and when ω_m = 0.25 m, θ_m = 0 m. The basemap is (a) the surface model generated from OIB ATM as described in Section 5.3 and (b) the CAMBOT image. The blue lines are manually selected ridge lines for reference. The value after **P** in the legend labels indicates the persistence threshold and the value after **R** indicates the roughness threshold. The black dashed box in (a) highlights the ridge lines removed during the topological simplification. The purple dashed box in (a) highlights the ridge line extended during the topological simplification.



(a) (b)

Figure A.21: Comparison between the ridge extraction results when ω_m = 0.25 m, θ_m = 0 m and when ω_m = 0.25 m, θ_m = 0.09 m. The basemap is (a) the surface model generated from OIB ATM as described in Section 5.3 and (b) the CAMBOT image. The blue lines are manually selected ridge lines for reference. The value after *P* in the legend labels indicates the persistence threshold and the value after *R* indicates the roughness threshold.

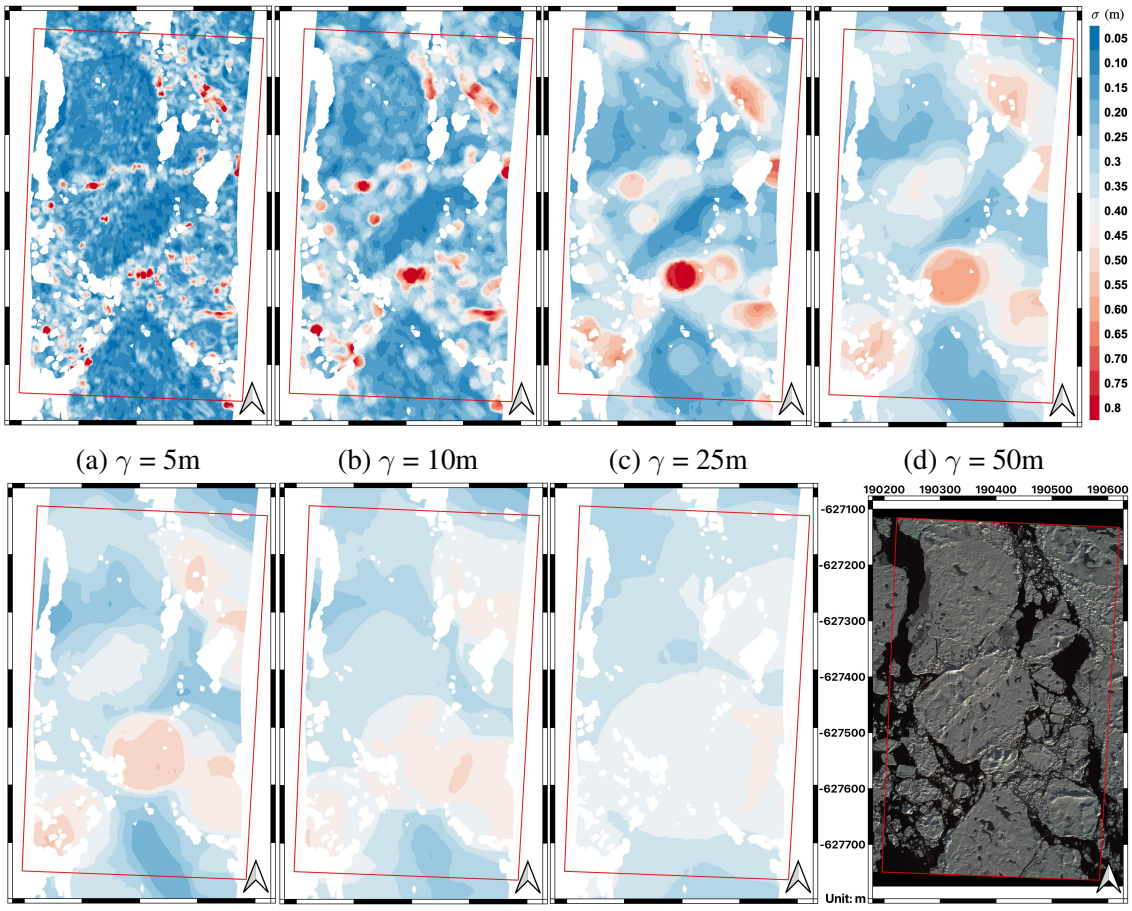
effects on the results. When ω_m is set to 0.25 m, most ridge structures become smaller. For instance, the long ridge in the middle of the figure has much longer cross ridge structure lines (e.g., lines within the black dashed box) in the ω_m = 0 result compared to the ω_m = 0.25 m result.

When most critical points in a region are removed, a cancellation operation is more likely to be applied to boundary maxima/minima, and this leads to the removal of arcs. Typically, this case happens when the arc to be removed is in a flat region, i.e., there is no other significant surface features around such arc. Thus, when topological simplification is applied, paths from the peaks (i.e., maxima) toward the flat sea ice areas tend to be removed. On the other hand, when the maximum to be removed is internal, a new arc with a longer chain of geometries is created after the simplification (shown in the purple dashed box). It is most likely to occur along the ridge line or when two maxima are in close proximity.

Fig. A.21 compares the results when no roughness filter is applied (depicted by red lines) or when a filter with roughness threshold $\theta_m = 0.09$ m is applied (depicted by green lines) during ridge extraction. In both cases, the persistence threshold ω_m is set to 0.25 m. As we can see, the roughness filter removes the parts of ridge structures that are on the flat sea ice surface (i.e., with a lower roughness value). The comparison between the two results illustrates that the roughness filter can effectively trim the extracted ridge structure without removing the lines that match the connections between ridge peaks, i.e., the ridge lines.

A.8 Chapter 5 - Additional results

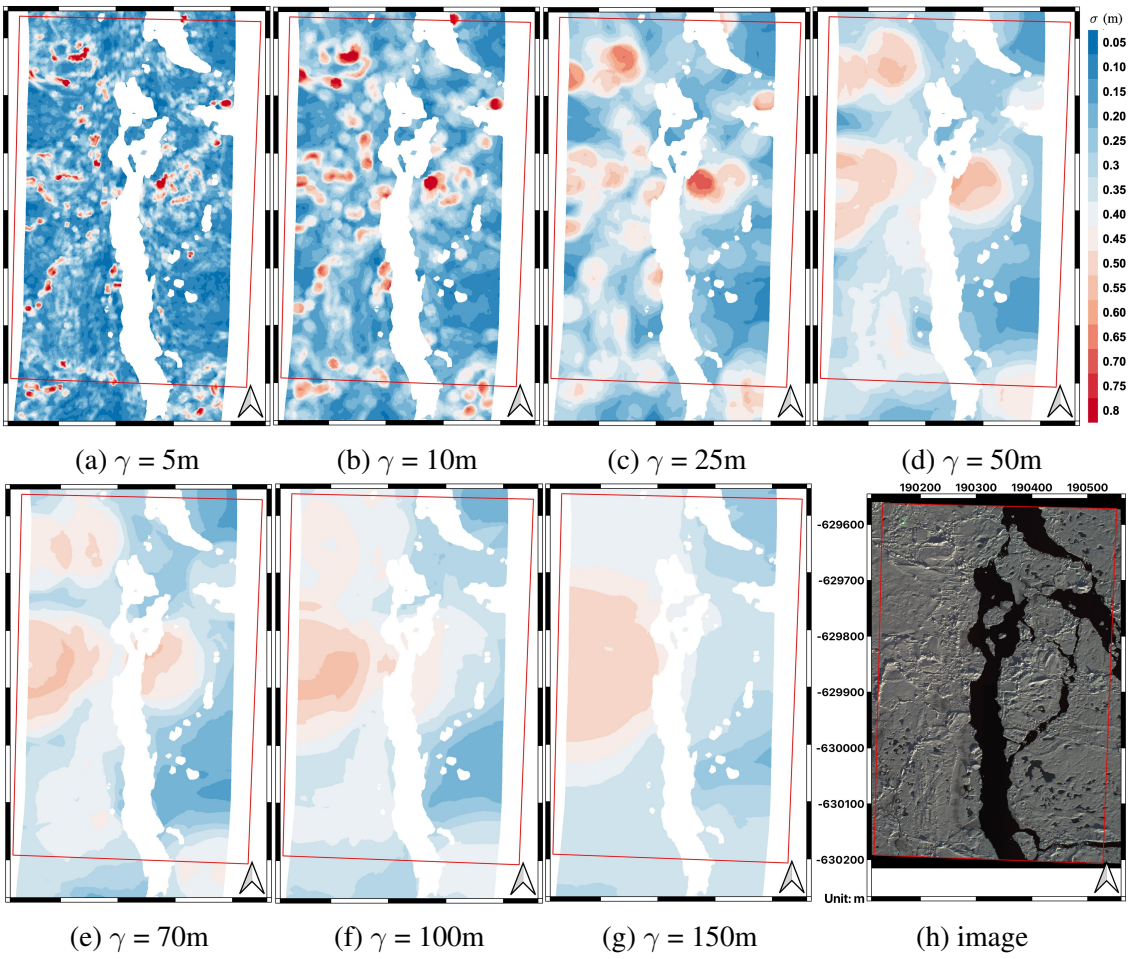
In this section, we list the experiment results of the work presented in Chapter 5 but are not shown in the content of Chapter 5. Specifically, Figs. A.22, A.23, and A.24 show the roughness maps of region B, C, and D with different radii, respectively.



(a) γ = 5m (b) γ = 10m (c) γ = 25m (d) γ = 50m

(e) γ = 70m (f) γ = 100m (g) γ = 150m (h) image

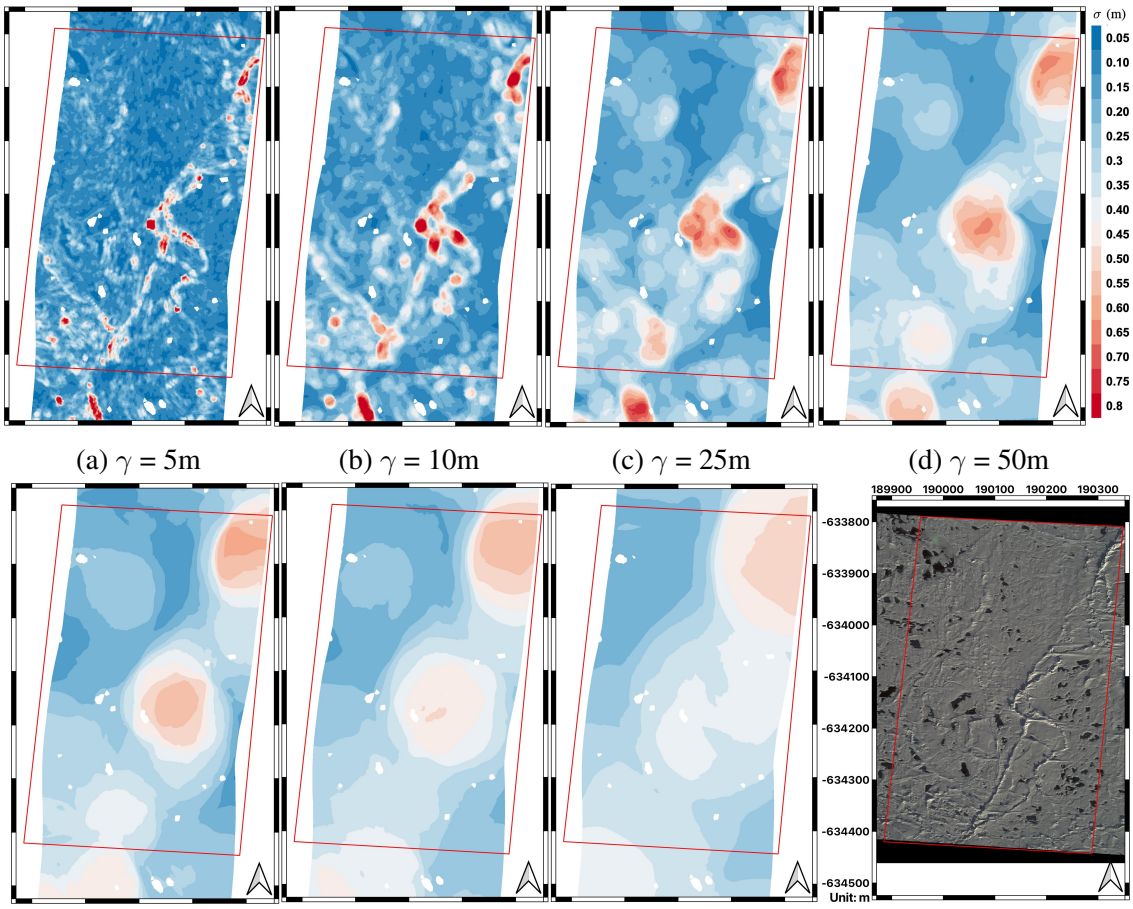
Figure A.22: Region B: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image.



(a) γ = 5m (b) γ = 10m (c) γ = 25m (d) γ = 50m

(e) γ = 70m (f) γ = 100m (g) γ = 150m (h) image

Figure A.23: Region C: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image.



(a) γ = 5m (b) γ = 10m (c) γ = 25m (d) γ = 50m

(e) γ = 70m (f) γ = 100m (g) γ = 150m (h) image

Figure A.24: Region D: (a - g) maps of roughness calculated on TINs with different measure radii. (h) corresponding OIB CAMBOT image of the same region. The red boundaries in figures highlight the overlap between the OIB ATM data and OIB CAMBOT image.

Bibliography

[1] About ArcGIS, 2024. https://www.esri.com/en-us/arcgis/about-arcgis/overview [Online; accessed February-2024].

[2] Arctic SDI Topographic Basemap. Norwegian Mapping Authority., 2024. https://arctic-sdi.org/services/topografic-basemap/ [Online; accessed February-2024].

[3] GRASS GIS, 2024. https://grass.osgeo.org/ [Online; accessed February-2024].

[4] Intel Implicit SPMD Program Compiler, 2024. https://ispc.github.io/ [Online; accessed February-2024].

[5] Intel oneAPI Threading Building Blocks, 2024. https://software.intel.com/en-us/oneapi/onetbb [Online; accessed February-2024].

[6] Welcome to the QGIS project, 2024. https://www.qgis.org/en/site/ [Online; accessed February-2024].

[7] Pankaj K. Agarwal, Alex Beutel, and Thomas Mølhave. TerraNNI: Natural Neighbor Interpolation on 2D and 3D Grids Using a GPU. *ACM Transactions on Spatial Algorithms and Systems*, 2(2):1–31, July 2016.

[8] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 1383–1394, New York, NY, USA, 2015. Association for Computing Machinery.

[9] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings - 2002 IEEE International Conference on Multimedia and Expo, ICME 2002*, volume 1, pages 705–708, 2002.

[10] Alexandros Avdis, Adam S. Candy, Jon Hill, Stephan C. Kramer, and Matthew D. Piggott. Efficient unstructured mesh generation for marine renewable energy applications. *Renewable Energy*, 116:842–856, 2018.

[11] Chandrajit L. Bajaj and Daniel R. Schikore. Topology preserving data simplification with error bounds. *Computers & Graphics*, 22(1):3–12, 1998.

[12] Wayman E. Baker, George D. Emmitt, Franklin Rivers Robertson, Robert Atlas, John E Molinari, David A. Bowdle, J. N. Paegle, R. Michael Hardesty, Robert T. Menzies, Tiruvalam N. Krishnamurti, Robert A. Brown, Madison John Post, John Anderson, Andrew C. Lorenc, and James L. McElroy. Lidar-measured winds from space: A key component for weather and climate prediction. *Bulletin of the American Meteorological Society*, 76:869–888, 1995.

[13] Thomas F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *The American Mathematical Monthly*, 77(5):475–485, 1970.

[14] Justin F. Beckers, Angelika H.H. Renner, Gunnar Spreen, Sebastian Gerland, and Christian Haas. Sea-ice surface roughness estimates from airborne laser scanner and laser altimeter observations in Fram Strait and north of Svalbard. *Annals of Glaciology*, 56(69):235–244, 2015.

[15] Matthew V. Bilskie, Scott C. Hagen, and Stephen C. Medeiros. Unstructured finite element mesh decimation for real-time Hurricane storm surge forecasting. *Coastal Engineering*, 156(October 2019):103622, 2020.

[16] Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Computational Geometry*, 22(1-3):185–203, May 2002.

[17] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and Radon Wasserstein Barycenters of Measures. *Journal of Mathematical Imaging and Vision*, 51(1):22–45, 2015.

[18] Peer-Timo Bremer, Valerio Pascucci, and Bernd Hamann. Maximizing Adaptivity in Hierarchical Topological Models Using Cancellation Trees. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[19] Ellen M. Buckley, Sinéad L. Farrell, Kyle Duncan, Laurence N. Connor, John M. Kuhn, and Rose Anne T. Dominguez. Classification of Sea Ice Summer Melt Features in High-Resolution IceBridge Imagery. *Journal of Geophysical Research: Oceans*, 125(5), 2020.

[20] Daniela Cabiddu and Marco Attene. Large mesh simplification for distributed environments. *Computers & Graphics*, 51:81–89, 2015.

[21] Moira Callens, Niko E. C. Verhoest, and Malcolm Davidson. Parameterization of tillage-induced single-scale soil roughness from 4-m profiles. *IEEE Transactions on Geoscience and Remote Sensing*, 44(4):878–888, 2006.

[22] Adam S. Candy, Alexandros Avdis, Jon Hill, Gerard Gorman, and Matthew D. Piggott. Integration of Geographic Information System frameworks into domain discretisation and meshing processes for geophysical models. *Geoscientific Model Development Discussions*, 7:5993–6060, 2014.

[23] Giulia Castellani, Christof Lüpkes, Stefan Hendricks, and Rüdiger Gerdes. Variability of Arctic sea-ice topography and its impact on the atmospheric surface drag. *Journal of Geophysical Research: Oceans*, 119(10):6743–6762, 2014.

[24] Computational Geometry Algorithms Library (CGAL), 2021. `https://www.cgal.org/` [Online; accessed February-2021].

[25] Patrick Ciarlet and Françoise Lamour. Does contraction preserve triangular meshes? *Numerical Algorithms*, 13:201–223, 1996.

[26] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. BDAM – Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. *Computer Graphics Forum*, 22(3):505–514, 2003.

[27] Paolo Cignoni, Claudio Montani, Raffaele Perego, and Roberto Scopigno. Parallel 3D Delaunay Triangulation. *Computer Graphics Forum*, 12(3):129–142, 1993.

[28] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[29] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[30] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The mpi message passing interface standard. In Karsten M. Decker and René M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218, Basel, 1994. Birkhäuser Basel.

[31] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. In *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*, SCG '05, page 263–271, New York, NY, USA, 2005. Association for Computing Machinery.

[32] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz functions have L p-stable persistence. *Foundations of computational mathematics*, 10(2):127–139, 2010.

[33] Lidija Čomić, Leila De Floriani, and Federico Iuricich. Dimension-independent multiresolution Morse complexes. *Computers & Graphics*, 36(5):541–547, August 2012.

[34] Josefino C. Comiso, Walter N. Meier, and Robert Gersten. Variability and trends in the Arctic Sea ice cover: Results from different techniques. *Journal of Geophysical Research: Oceans*, 122(8):6883–6900, August 2017.

[35] Laurence N. Connor, Sinéad Louise Farrell, David C. McAdoo, William B. Krabill, and Serdar Manizade. Validating ICESat over thick sea ice in the Northern Canada Basin. *IEEE Transactions on Geoscience and Remote Sensing*, 51(4):2188–2200, 2013.

[36] Judith A. Curry, Julie L. Schramm, and Elizabeth E. Ebert. Sea ice-albedo climate feedback mechanism. *Journal of Climate*, 8(2):240–247, 1995.

[37] Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering*, 5(1):46–55, 1998.

[38] Michele Dalponte, Lorenzo Bruzzone, and Damiano Gianelle. Fusion of hyperspectral and LIDAR remote sensing data for classification of complex forest areas. *IEEE Transactions on Geoscience and Remote Sensing*, 46(5):1416–1427, 2008.

[39] Dyre Oliver Dammann, Hajo Eicken, Andrew R. Mahoney, Eyal Saiet, Franz J. Meyer, and John C. Craig George. Traversing Sea Ice-Linking Surface Roughness and Ice Trafficability Through SAR Polarimetry and Interferometry. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2):416–433, 2018.

[40] Emanuele Danovaro, Leila De Floriani, Paola Magillo, Mohammed Mostefa Mesmoudi, and Enrico Puppo. Morphology-Driven Simplification and Multiresolution Modeling of Terrains. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*, GIS '03, pages 63–70, New York, NY, USA, 2003. Association for Computing Machinery.

[41] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. Graphframes: an integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, New York, NY, USA, 2016. Association for Computing Machinery.

[42] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[43] Leila De Floriani, Marianna Facinoli, Paola Magillo, and Debora Dimitri. A hierarchical spatial index for triangulated surfaces. In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, pages 86–91, 2008.

[44] Leila De Floriani, Ulderico Fugacci, Federico Iuricich, and Paola Magillo. Morse Complexes for Shape Segmentation and Homological Analysis: Discrete Models and Algorithms. *Computer Graphics Forum*, 34(2):761–785, 2015.

[45] Leila De Floriani and Annie Hui. Data structures for simplicial complexes: An analysis and a comparison. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, pages 119–128, Goslar, Germany, 2005. Eurographics Association.

[46] Christopher DeCoro and Natalya Tatarchuk. Real-time mesh simplification using the GPU. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 161–166, 2007.

[47] Frank Dehne, Chiristian Langis, and Gerhard Roth. Mesh simplification in parallel. In *Algorithms And Architectures For Parallel Processing*, ICA3PP 2000, pages 281–290. World Scientific, 2000.

[48] Denis Demchev, Vladimir Volkov, Eduard Kazakov, Pablo F. Alcantarilla, Stein Sandven, and Viktoriya Khmeleva. Sea Ice Drift Tracking From Sequential SAR Images Using Accelerated-KAZE Features. *IEEE Transactions on Geoscience and Remote Sensing*, 55(9):5174–5184, September 2017.

[49] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut Mathématique*, 66:23–45, 1998.

[50] Tamal K. Dey and Ryan Slechta. Edge contraction in persistence-generated discrete Morse vector fields. *Computers and Graphics (Pergamon)*, 74:33–43, 2018.

[51] Tamal K. Dey, Jiayuan Wang, and Yusu Wang. Improved Road Network Reconstruction using Discrete Morse Theory. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–4. ACM, November 2017.

[52] Margaret F. J. Dolan, Anthony J. Grehan, Janine C. Guinan, and Colin Brown. Modelling the local distribution of cold-water corals in relation to bathymetric variables: Adding spatial context to deep-sea video data. *Deep Sea Research Part I: Oceanographic Research Papers*, 55(11):1564–1579, 2008.

[53] Kyle Duncan and Sinead L. Farrell. Determining variability in arctic sea ice pressure ridge topography with icesat-2. *Geophysical Research Letters*, 49(18), September 2022.

[54] Kyle Duncan, Sinéad L. Farrell, Laurence N. Connor, Jackie Richter-Menge, Jennifer K. Hutchings, and RoseAnne T. Dominguez. High-resolution airborne observations of sea-ice pressure ridge sail height. *Annals of Glaciology*, 59(76pt2):137–147, 2018.

[55] Kyle Duncan, Sinead L. Farrell, Jennifer Hutchings, and Jacqueline Richter-Menge. Late winter observations of sea ice pressure ridge sail height. *IEEE Geoscience and Remote Sensing Letters*, 18(9):1525–1529, July 2020. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[56] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*, volume 6 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, 2001.

[57] Herbert Edelsbrunner and John Harer. Jacobi Sets of Multiple Morse Functions. In *Foundations of Computational Mathematics*, volume 312 of *London Mathematical Society Lecture Note Series*, pages 37–57. Cambridge University Press, 2004.

[58] Herbert Edelsbrunner and John Harer. Persistent homology-a survey. *Contemporary Mathematics*, 453:257–282, 2008.

[59] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, jan 1994.

[60] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, January 1990.

[61] Herbert Edelsbrunner and Nimish R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15(3):223–241, 1996.

[62] Hajo Eicken, Bodil A. Bluhm, Roy Eric Collins, Christian Haas, Malcom Ingham, Rolf Gradinger, Andrew Richard Mahoney, Marcel Nicolaus, and Donald K. Perovich. Field techniques in sea-ice research. In *Cold regions science and marine technology*, pages 1–20. Eolss Publishers, 2014.

[63] Hajo Eicken, Thomas C. Grenfell, Donald K. Perovich, Jacqueline Richter-Menge, and Karoline Frey. Hydraulic controls of summer Arctic pack ice albedo. *Journal of Geophysical Research: Oceans*, 109, 2004.

[64] Hajo Eicken, Walter B. Tucker, and Donald K. Perovich. Indirect measurements of the mass balance of summer Arctic sea ice with an electromagnetic induction technique. *Annals of Glaciology*, 33:194–200, 2001.

[65] Zachary Fair, Mark Flanner, Kelly M. Brunt, Helen Amanda Fricker, and Alex Gardner. Using ICESat-2 and Operation IceBridge altimetry for supraglacial lake depth retrievals. *Cryosphere*, 14(11):4253–4263, 2020.

[66] Sinéad L. Farrell, Kelly M. Brunt, Julia M. Ruth, John M. Kuhn, Laurence N. Connor, and Kaitlin M. Walsh. Sea-ice freeboard retrieval using digital photon-counting laser altimetry. *Annals of Glaciology*, 56(69):167–174, 2015.

[67] Sinéad L. Farrell, Kyle Duncan, Ellen M. Buckley, Jackie Richter-Menge, and Ruohan Li. Mapping Sea Ice Surface Topography in High Fidelity With ICESat-2. *Geophysical Research Letters*, 47(21), 2020.

[68] Sinéad L. Farrell, Thorsten Markus, Ron Kwok, and Laurence Connor. Laser altimetry sampling strategies over sea ice. *Annals of Glaciology*, 52(57 PART 1):69–76, 2011.

[69] Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. Efficient representation and analysis of triangulated terrains. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–4, 2017.

[70] Riccardo Fellegara, Federico Iuricich, Leila De Floriani, and Kenneth Weiss. Efficient computation and simplification of discrete Morse decompositions on triangulated terrains. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 223–232. ACM, 2014.

[71] Riccardo Fellegara, Federico Iuricich, and Yunting Song. Terrain Analysis on the IA data structure, 2022. https://github.com/UMDGeoVis/Terrain_Analysis_on_IA [Online; accessed February-2024].

[72] Riccardo Fellegara, Federico Iuricich, Yunting Song, and Leila De Floriani. Terrain trees: a framework for representing, analyzing and visualizing triangulated terrains. *GeoInformatica*, 27(3):525–564, 2023.

[73] Riccardo Fellegara and Yunting Song. Terrain trees library, 2023. https://zenodo.org/records/10714553[Online; accessed February-2024].

[74] Riccardo Fellegara, Kenneth Weiss, and Leila De Floriani. The Stellar decomposition: A compact representation for simplicial complexes and beyond. *Computers & Graphics*, 98:322–343, August 2021.

[75] Haoan Feng, Yunting Song, and Leila De Floriani. Critical Features Tracking on Irregular Terrain Meshes by a Scale-Space Method (in preparation). In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems 2024*.

[76] Robin Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134:90–145, 1998.

[77] Martin Franc and Václav Skala. Parallel triangular mesh reduction. In *Proceedings of scientific computing*, ALGORITMY 2000, pages 357–367, 2000.

[78] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On Visible Surface Generation by a Priori Tree Structures. *SIGGRAPH Computer Graphics*, 14(3):124–133, July 1980.

[79] Ulderico Fugacci, Michael Kerber, and Hugo Manet. Topology-Preserving Terrain Simplification. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '20, pages 36–47, New York, NY, USA, 2020. ACM.

[80] Ulderico Fugacci, Claudia Landi, and Hanife Varlı. Critical sets of PL and discrete Morse theory: A correspondence. *Computers & Graphics*, 90:43–50, August 2020.

[81] Ulderico Fugacci, Sara Scaramuccia, Federico Iuricich, and Leila De Floriani. Persistent Homology: a Step-by-step Introduction for Newcomers. In *Eurographics Italian Chapter Conference*, pages 1–10, 2016.

[82] Guoping Gao, Changsheng Chen, Jianhua Qi, and Robert C. Beardsley. An unstructured-grid, finite-volume sea ice model: Development, validation, and application. *Journal of Geophysical Research: Oceans*, 116(9):1–15, 2011.

[83] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997*, pages 209–216, 1997.

[84] GeoPandas, 2024. https://geopandas.org/en/stable/index.html# [Online; accessed January-2024].

[85] Nico Grund, Evgenij Derzapf, and Michael Guthe. Instant Level-of-Detail. In *Vision, Modeling, and Visualization*. The Eurographics Association, 2011.

[86] André Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):168–189, 1999.

[87] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[88] Topraj Gurung, Daniel Laney, Peter Lindstrom, and Jarek Rossignac. SQuad: Compact representation for triangle meshes. *Computer Graphics Forum*, 30(2):355–364, 2011.

[89] Topraj Gurung and Jarek Rossignac. SOT: A compact representation for tetrahedral meshes. In *Proceedings SIAM/ACM Geometric and Physical Modeling*, SPM '09, pages 79–88, San Francisco, USA, 2009.

[90] Attila Gyulassy, Mark Duchaineau, Vijay Natarajan, Valerio Pascucci, Eduardo Bringa, Andrew Higginbotham, and Bernd Hamann. Topologically clean distance fields. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1432–1439, November 2007.

[91] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. Carnegie mellon university technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Pittsburgh, PA, 1997.

[92] Gilbert Held and Thomas Marshall. *Data compression; techniques and applications: Hardware and software considerations*. John Wiley & Sons, Inc., 1991.

[93] Gísli R. Hjaltason and Hanan Samet. Speeding up construction of PMR quadtree-based spatial indexes. *The VLDB Journal*, 11(2):109–137, 2002.

[94] Thomas Hollands and Wolfgang Dierking. Performance of a multiscale correlation algorithm for the estimation of sea-ice drift from SAR images: initial results. *Annals of Glaciology*, 52(57):311–317, 2011.

[95] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 19–26. ACM, New York, NY, USA, 1993.

[96] Veysi İşler, Rynson W. H. Lau, and Mark Green. Real-time multi-resolution modeling for complex virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '96, pages 11–19, New York, NY, USA, 1996. Association for Computing Machinery.

[97] Polona Itkin, Stefan Hendricks, Melinda Webster, Luisa Von Albedyll, Stefanie Arndt, Dmitry Divine, Matthias Jaggi, Marc Oggier, Ian Raphael, Robert Ricker, Jan Rohde, Martin Schneebeli, and Glen E. Liston. Sea ice and snow characteristics from year-long transects at the MOSAiC Central Observatory. *Elementa: Science of the Anthropocene*, 11(1):00048, February 2023.

[98] Federico Iuricich. Persistence cycles for visual exploration of persistent homology. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4966–4979, 2021.

[99] Federico Iuricich and Leila De Floriani. Hierarchical Forman Triangulation: A multiscale model for scalar field analysis. *Computers and Graphics (Pergamon)*, 66:113–123, 2017.

[100] Michael J de Smith, Michael Goodchild, and Paul Longley. *Geospatial Analysis: A Comprehensive Guide.* The Winchelsea Press, S.l., 2018.

[101] Thomas Johnson, Michel Tsamados, Jan-Peter Muller, and Julienne Stroeve. Mapping Arctic Sea-Ice Surface Roughness with Multi-Angle Imaging SpectroRadiometer. *Remote Sensing*, 14(24):6249, December 2022.

[102] Henry King, Kevin Knudson, and Neža Mramor. Generating discrete Morse functions from point data. *Experimental Mathematics*, 14(4):435–444, 2005.

[103] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In *Proceedings of the 7th Conference on Visualization '96*, VIS '96, pages 311–318, Washington, DC, USA, 1996. IEEE Computer Society Press.

[104] William B. Krabill, Waleed Abdalati, Earl B. Frederick, Serdar S. Manizade, Chreston F. Martin, John G. Sonntag, Robert N. Swift, Robert H. Thomas, and James K. Yungel. Aircraft laser altimetry measurement of elevation changes of the greenland ice sheet: technique and accuracy assessment. *Journal of Geodynamics*, 34(3-4):357–376, October 2002.

[105] Nathan T. Kurtz, Thorsten Markus, Donald J. Cavalieri, William Krabill, John G. Sonntag, and Jeffrey Miller. Comparison of ICESat data with airborne laser altimeter measurements over arctic sea ice. *IEEE Transactions on Geoscience and Remote Sensing*, 46(7):1913–1924, 2008.

[106] Ron Kwok, Glenn F. Cunningham, Serdar S. Manizade, and William B. Krabill. Arctic sea ice freeboard from IceBridge acquisitions in 2009: Estimates and comparisons with ICESat. *Journal of Geophysical Research: Oceans*, 117(2), 2012. Publisher: Blackwell Publishing Ltd.

[107] Ron Kwok, Glenn F. Cunningham, H. Jay Zwally, and Donghui Yi. ICESat over Arctic sea ice: Interpretation of altimetric and reflectivity profiles. *Journal of Geophysical Research: Oceans*, 111(6), 2006.

[108] Ron Kwok, H. Jay Zwally, and Donghui Yi. ICESat observations of Arctic sea ice: A first look. *Geophysical Research Letters*, 31(16):1–5, 2004.

[109] Jack C. Landy, Alek A. Petty, Michel Tsamados, and Julienne C. Stroeve. Sea Ice Roughness Overlooked as a Key Source of Uncertainty in CryoSat-2 Ice Freeboard Retrievals. *Journal of Geophysical Research: Oceans*, 125(5):1–18, 2020.

[110] Jack C. Landy, Michel Tsamados, and Randall K. Scharien. A Facet-Based Numerical Model for Simulating SAR Altimeter Echoes from Heterogeneous Sea Ice Surfaces. *IEEE Transactions on Geoscience and Remote Sensing*, 57(7):4164–4180, 2019.

[111] Benjamin A. Lange, Justin F. Beckers, J. Alec Casey, and Christian Haas. Airborne observations of summer thinning of multiyear sea ice originating from the lincoln sea. *Journal of Geophysical Research: Oceans*, 124(1):243–266, January 2019.

[112] Andrew Lanier, Chris Romsos, and Chris Goldfinger. Seafloor Habitat Mapping on the Oregon Continental Margin: A Spatially Nested GIS Approach to Mapping Scale, Mapping Methods, and Accuracy Quantification. *Marine Geodesy*, 30(1-2):51–76, 2007.

[113] Guillaume Lavoue and Massimiliano Corsini. A comparison of perceptually-based metrics for objective evaluation of geometry processing. *IEEE Transactions on Multimedia*, 12(7):636–649, 2010.

[114] Hyunho Lee and Min-Ho Kyung. Parallel mesh simplification using embedded tree collapsing. *The Visual Computer*, 32(6):967–976, 2016.

[115] Seoungjun Lee, Dongsoo Har, and Dongsuk Kum. Drone-assisted disaster management: Finding victims via infrared camera and lidar sensor fusion. In *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pages 84–89. IEEE, 2016.

[116] Mengyuan Li, Anmin Zhang, Dianjun Zhang, Mingwei Di, and Qingju Liu. Automatic Sounding Generalization Maintaining the Characteristics of Submarine Topography. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:10278–10286, 2021. Publisher: IEEE.

[117] Ron W. Lindsay and Drew Andrew Rothrock. Arctic sea ice leads from advanced very high resolution radiometer images. *Journal of Geophysical Research: Oceans*, 100(C3):4533–4544, March 1995.

[118] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 259–262, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[119] Peter Lindstrom and Valerio Pascucci. Terrain Simplification Simplified: a General Framework for View-Dependent Out-of-core Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[120] Stefanie Linow and Wolfgang Dierking. Object-Based Detection of Linear Kinematic Features in Sea Ice. *Remote Sensing*, 9(5):493, May 2017.

[121] Raymond T. Lowry and Peter Wadhams. On the statistical distribution of pressure ridges in sea ice. *Journal of Geophysical Research: Oceans*, 84(C5):2487–2494, May 1979.

[122] Jonas Lukasczyk, Christoph Garth, Ross Maciejewski, and Julien Tierny. Localized topological simplification of scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):572–582, February 2021.

[123] Federico luricich and Leila De Floriani. A combined geometrical and topological simplification hierarchy for terrain analysis. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 493–496, 2014.

[124] Claudio Mancinelli, Marco Livesu, and Enrico Puppo. Gradient Field Estimation on Triangle Meshes. In Marco Livesu, Gianni Pintore, and Alberto Signoroni, editors, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2018.

[125] Thorsten Markus, Tom Neumann, Anthony Martino, Waleed Abdalati, Kelly Brunt, Beata Csatho, Sinead Farrell, Helen Fricker, Alex Gardner, David Harding, Michael Jasinski, Ron Kwok, Lori Magruder, Dan Lubin, Scott Luthcke, James Morison, Ross Nelson, Amy Neuenschwander, Stephen Palm, Sorin Popescu, Ck Shum, Bob E. Schutz, Benjamin Smith, Yuekui Yang, and Jay Zwally. The Ice, Cloud, and land Elevation Satellite-2 (ICESat-2): Science requirements, concept, and implementation. *Remote Sensing of Environment*, 190:260–273, March 2017.

[126] Torge Martin, Michel Tsamados, David Schroeder, and Daniel L. Feltham. The impact of variable sea ice roughness on changes in Arctic Ocean surface stress: A model study. *Journal of Geophysical Research: Oceans*, 121(3):1931–1952, March 2016.

[127] Andrew Meigs. Active tectonics and the LiDAR revolution. *Lithosphere*, 5(2):226–229, 2013.

[128] Mohammed Mostefa Mesmoudi, Leila De Floriani, and Paola Magillo. Morphological analysis of terrains based on discrete curvature and distortion. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 415–418, 2008.

[129] Mohammed Mostefa Mesmoudi, Leila De Floriani, and Paola Magillo. Discrete distortion for surface meshes. In *International Conference on Image Analysis and Processing*, pages 652–661. Springer, 2009.

[130] Mohammed Mostefa Mesmoudi, Leila De Floriani, and Umberto Port. Discrete Distortion in Triangulated 3-Manifolds. *Computer Graphics Forum*, 27(5):1333–1340, 2008.

[131] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Proceedings VisMath 2002*, pages 35–57, 2003.

[132] Xin Miao, Hongjie Xie, Stephen F. Ackley, and Songfeng Zheng. Object-based arctic sea ice ridge detection from high-spatial-resolution imagery. *IEEE Geoscience and Remote Sensing Letters*, 13(6):787–791, June 2016.

[133] John Willard Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.

[134] Mohamed H. Mousa and Mohamed K. Hussein. High-performance simplification of triangular surfaces using a GPU. *Plos one*, 16(8):e0255832, 2021.

[135] Suthambhara Nagaraj, Vijay Natarajan, and Ravi S. Nanjundiah. A gradient-based comparison measure for visual analysis of multifield data. *Computer Graphics Forum*, 30(3):1101–1110, 2011.

[136] Randal C. Nelson and Hanan Samet. A consistent hierarchical representation for vector data. *ACM SIGGRAPH Computer Graphics*, 20(4):197–206, 1986.

[137] Thomas A. Neumann, Anita Brenner, David Hancock, John Robbins, Aimée Gibbons, Jeffrey Lee, Kaitlin Harbeck, Jack Saba, Scott B. Luthcke, and Timothy Rebold. ATLAS/ICESat-2 L2A Global Geolocated Photon Data, Version 6. 2023, Distributed by NASA National Snow and Ice Data Center Distributed Active Archive Center., 2023. doi:`https://doi.org/10.5067/ATLAS/ATL03.006`. [Online; accessed January-2024].

[138] Gregory M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In *Scientific Visualization: overviews, Methodologies and Techniques*, chapter 20, pages 429–525. IEEE Computer Society, 1997.

[139] Anne Nolin and Eugene Mar. Arctic Sea Ice Surface Roughness Estimated from Multi-Angular Reflectance Satellite Imagery. *Remote Sensing*, 11(1):50, December 2018.

[140] OCM Partners. 2013 USACE NCMP Topobathy Lidar: Molokai (HI) , 2021. NOAA National Centers for Environmental Information, `https://www.fisheries.noaa.gov/inport/item/49753`.

[141] Małgorzata Olejniczak, André Severo Pereira Gomes, and Julien Tierny. A Topological Data Analysis perspective on noncovalent interactions in relativistic calculations. *International Journal of Quantum Chemistry*, 120(8), April 2020.

[142] Vincent-De-Paul Onana, Nathan T. Kurtz, Sinead Louise Farrell, Lora S. Koenig, Michael Studinger, and Jeremy P. Harbeck. A Sea-Ice Lead Detection Algorithm for Use With High-Resolution Airborne Visible Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 51(1):38–56, January 2013.

[143] Opentopography - high-resolution topography data and tools, 2020. `http://www.opentopography.org/` [Online; accessed January-2020].

[144] Jack A. Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(4):150–157, 1982.

[145] Alberto Paoluzzi, Fausto Bernardini, Carlo Cattani, and Vincenzo Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics (TOG)*, 12(1):56–102, 1993.

[146] Alexandros Papageorgiou and Nikos Platis. Triangular mesh simplification on the GPU. *Visual Computer*, 31(2):235–244, 2015.

[147] Nikolaos K. Pavlis, Simon A. Holmes, Steve C. Kenyon, and John K. Factor. The development and evaluation of the earth gravitational model 2008 (egm2008). *Journal of geophysical research: solid earth*, 117(B4), 2012.

[148] Donald K. Perovich and Christopher Polashenski. Albedo evolution of seasonal arctic sea ice. *Geophysical Research Letters*, 39(8), 2012.

[149] Persim 0.3.1 documentation, 2022. `https://persim.scikit-tda.org/en/latest/index.html` [Online; accessed June-2022].

[150] Alek A. Petty, Michel C. Tsamados, and Nathan T. Kurtz. Atmospheric form drag coefficients over Arctic sea ice using remotely sensed ice topography data, spring 2009-2015. *Journal of Geophysical Research: Earth Surface*, 122(8):1472–1490, 2017.

[151] Alek A. Petty, Michel C. Tsamados, Nathan T. Kurtz, Sinéad L. Farrell, Thomas Newman, Jeremy P. Harbeck, Daniel L. Feltham, and Jackie A. Richter-Menge. Characterizing Arctic sea ice topography using high-resolution IceBridge data. *Cryosphere*, 10(3):1161–1179, 2016.

[152] Yuehui Qian, Yunting Song, Federico Iuricich, and Leila De Floriani. Topology-based Terrain Segmentation Using Apache Spark (in preparation). *International Journal of Geographical Information Science*.

[153] Jean Rabault, Malte Müller, Joey Voermans, Dmitry Brazhnikov, Ian Turnbull, Aleksey Marchenko, Martin Biuw, Takehiko Nose, Takuji Waseda, Malin Johansson, Øyvind Breivik, Graig Sutherland, Lars Robert Hole, Mark Johnson, Atle Jensen, Olav Gundersen, Yngve Kristoffersen, Alexander Babanin, Paulina Tedesco, Kai Haakon Christensen, Martin Kristiansen, Gaute Hope, Tsubasa Kodaira, Victor De Aguiar, Catherine Taelman, Cornelius P. Quigley, Kirill Filchuk, and Andrew R Mahoney. A dataset of direct observations of sea ice drift and waves in ice. *Scientific Data*, 10(1):251, May 2023.

[154] Lasse Rabenstein, Stefan Hendricks, Torge Martin, A Pfaffhuber, and Christian Haas. Thickness and surface-properties of different sea-ice regimes within the Arctic Trans Polar Drift: Data from summers 2001, 2004 and 2007. *Journal of Geophysical Research: Oceans*, 115(C12):2009JC005846, December 2010.

[155] Vanessa Robins, Peter John Wood, and Adrian P. Sheppard. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1646–1658, 2011.

[156] Luigi Rocca, Bernhard Jenny, and Enrico Puppo. A continuous scale-space method for the automated placement of spot heights on maps. *Computers and Geosciences*, 109(October 2015):216–227, 2017. Publisher: Elsevier Ltd.

[157] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, pages 455–465. Springer, Berlin, Heidelberg, 1993.

[158] Jarek Rossignac, Alla Safonova, and Andrzej Szymczak. 3D compression made simple: Edge-Breaker on a Corner Table. In *Proceedings Shape Modeling International 2001*, Genova, Italy, May 2001. IEEE Computer Society.

[159] Hanan Samet. *The Design and analysis of spatial data structure*. Addison Wesley, 1990.

[160] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[161] Hanan Samet and Robert E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics (TOG)*, 4(3):182–222, 1985.

[162] Samantha E. Saye, Daphne van der Wal, Kenneth Pye, and Simon J. Blott. Beach-dune morphological relationships and erosion/accretion: An investigation at five sites in England and Wales using LIDAR data. *Geomorphology*, 72(1):128–155, 2005.

[163] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 65–70. ACM, New York, NY, USA, 1992.

[164] Shapely 2.0.3 documentation, 2024. `https://shapely.readthedocs.io/en/stable/index.html` [Online; accessed March-2024].

[165] Michael K. Shepard, Bruce A. Campbell, Mark H. Bulmer, Tom G. Farr, Lisa R. Gaddis, and Jeffrey J. Plaut. The roughness of natural terrain: A planetary and remote sensing perspective. *Journal of Geophysical Research: Planets*, 106(E12):32777–32795, 2001.

[166] Nithin Shivashankar and Vijay Natarajan. Parallel computation of 3D Morse-Smale complexes. *Computer Graphics Forum*, 31(3):965–974, 2012.

[167] Andriani Skopeliti, Leda Stamou, Lysandros Tsoulos, and Shachak Pe'eri. Generalization of soundings across scales: From DTM to harbour and approach nautical charts. *ISPRS International Journal of Geo-Information*, 9(11):693, 2020.

[168] Pierre J. Soille and Marc M. Ansoult. Automated basin delineation from digital elevation models using mathematical morphology. *Signal Processing*, 20(2):171–182, 1990.

[169] Yunting Song, Leila De Floriani, Kyle Duncan, and Sinead Farrell. Extracting sea ice topographic features from triangulated surface models (in preparation). *IEEE Transactions on Geoscience and Remote Sensing*.

[170] Yunting Song and Riccardo Fellegara. Topology-aware simplification on terrain trees, 2023. `https://https://zenodo.org/records/10723706`[Online; accessed February-2024].

[171] Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. Efficient topology-aware simplification of large triangulated terrains. In *Proceedings of the 29th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 576–587. ACM, 2021.

[172] Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. Parallel Topology-aware Mesh Simplification on Terrain Trees. *ACM Transactions on Spatial Algorithms and Systems*, 10(2), 2024.

[173] Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila Floriani. Terrain trees library: a tool for efficient and scalable terrain mesh processing. In *AGU Fall Meeting Abstracts*, volume 2021, pages IN45B–01, 2021.

[174] John Sonntag. Summer 2019 IceBridge Arctic Flight Plans, June 2019. `https://icebridge.gsfc.nasa.gov/wp-content/uploads/2019/06/flightplans_20190625.pdf` [Online; accessed February-2024].

[175] Lucie Strub-Klein and Denise Sudom. A comprehensive analysis of the morphology of first-year sea ice ridges. *Cold Regions Science and Technology*, 82:94–109, 2012.

[176] Michael Studinger. IceBridge ATM L1B Elevation and Return Strength, Version 2. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center, 2013, updated 2020. doi:`https://doi.org/10.5067/19SIM5TXKPGT`. [Online; accessed November-2021].

[177] Michael Studinger. IceBridge Narrow Swath ATM L1B Elevation and Return Strength, Version 2. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center, 2014, updated 2020. doi:`https://doi.org/10.5067/CXEQS8KVIXEI`. [Online; accessed November-2023].

[178] Michael Studinger and Jeremy Harbeck. IceBridge CAMBOT L1B Geolocated Images, Version 2. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center, 2019. doi:`https://doi.org/10.5067/B0HL940D452L`. [Online; accessed November-2021].

[179] Michael Studinger, Serdar S. Manizade, Matthew A. Linkswiler, and James K. Yungel. High-resolution imaging of supraglacial hydrological features on the greenland ice sheet with nasa's airborne topographic mapper (atm) instrument suite. *Cryosphere*, 16:3649–3668, 9 2022. Probably explain why we have points in narrow scan but not wide scan.¡br/¿.

[180] Peter Su and Robert L. Scot Drysdale. A comparison of sequential Delaunay triangulation algorithms. *Proceedings of the Annual Symposium on Computational Geometry*, Part F1293:61–70, 1995.

[181] Tyler C. Sutterley, Isabella Velicogna, Xavier Fettweis, Eric Rignot, Brice Noël, and Michiel van den Broeke. Evaluation of Reconstructions of Snow/Ice Melt in Greenland by Regional Atmospheric Climate Models Using Laser Altimetry Data. *Geophysical Research Letters*, 45(16):8324–8333, 2018.

[182] Bing Tan, Zhi Jun Li, Peng Lu, Christian Haas, and Marcel Nicolaus. Morphology of sea ice pressure ridges in the northwestern weddell sea in winter. *Journal of Geophysical Research: Oceans*, 117(6):1–13, 2012.

[183] Victor J. D. Tsai. Delaunay triangulations in TIN creation: An overview and a linear-time algorithm. *International Journal of Geographical Information Systems*, 7(6):501–524, 1993.

[184] Michel Tsamados, Daniel L. Feltham, David Schroeder, Daniela Flocco, Sinead L. Farrell, Nathan Kurtz, Seymour W. Laxon, and Sheldon Bacon. Impact of Variable Atmospheric and Oceanic Form Drag on Simulations of Arctic Sea Ice*. *Journal of Physical Oceanography*, 44(5):1329–1353, May 2014.

[185] Mark Tschudi, Charles Fowler, James Maslanik, and Julienne Stroeve. Tracking the Movement and Changing Surface Characteristics of Arctic Sea Ice. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 3(4):536–540, 2010.

[186] Akash Anil Valsangkar, Joy Merwin Monteiro, Vidya Narayanan, Ingrid Hotz, and Vijay Natarajan. An exploratory framework for cyclone identification and tracking. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1460–1473, March 2019.

[187] Niko E. C. Verhoest, Hans Lievens, Wolfgang Wagner, Jesús Álvarez-Mozos, Mary Susan Moran, and Francesco Mattia. On the Soil Roughness Parameterization Problem in Soil Moisture Retrieval of Bare Surfaces from Synthetic Aperture Radar. *Sensors*, 8(7):4213–4248, 2008.

[188] Ruisheng Wang, Jiju Peethambaran, and Dong Chen. Lidar point clouds to 3-d urban models : a review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2):606–627, 2018.

[189] Frank W. Warner. *Foundations of differentiable manifolds and Lie groups*, volume 94. Springer Science & Business Media, 1983.

[190] Tim L. Webster, Donald L. Forbes, Steve Dickie, and Roger Shreenan. Using topographic lidar to map flood risk from storm-surge events for Charlottetown, Prince Edward Island, Canada. *Canadian Journal of Remote Sensing*, 30(1):64–76, 2004.

[191] Kenneth Weiss, Federico Iuricich, Riccardo Fellegara, and Leila De Floriani. A primal/dual representation for discrete Morse complexes on tetrahedral meshes. *Computer Graphics Forum*, 32(3):361–370, 2013.

[192] Stephen A. White and Yong Wang. Utilizing DEMs derived from LIDAR data to analyze morphologic change in the North Carolina coastline. *Remote sensing of environment*, 85(1):39–47, 2003.

[193] Nicholas C. Wright and Chris M. Polashenski. Open-source algorithm for detecting sea ice surface features in high-resolution optical imagery. *The Cryosphere*, 12(4):1307–1329, April 2018.

[194] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. *Proceedings of the IEEE Visualization Conference*, pages 327–334, 1996.

[195] Xin Xu, Federico Iuricich, and Leila De Floriani. A Persistence-Based Approach for Individual Tree Mapping. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, pages 191–194. ACM, November 2020.

[196] Bisheng Yang, Wenzhong Shi, and Qingquan Li. A dynamic method for generating multi-resolution TIN models. *Photogrammetric Engineering & Remote Sensing*, 71(8):917–926, 2005.

[197] Donghui Yi, Alejandro Egido, Walter H.F. Smith, Laurence Connor, Christopher Buchhaupt, and Dexin Zhang. Arctic Sea-Ice Surface Elevation Distribution from NASA's Operation IceBridge ATM Data. *Remote Sensing*, 14(13), July 2022. Publisher: MDPI.

[198] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[199] Xianwei Zheng, Hanjiang Xiong, Jianya Gong, and Linwei Yue. A morphologically preserved multi-resolution TIN surface modeling and visualization method for virtual globes. *ISPRS Journal of Photogrammetry and Remote Sensing*, 129:41–54, 2017.