Parallel Topology-aware Mesh Simplification on Terrain Trees

YUNTING SONG, University of Maryland, College Park, USA
RICCARDO FELLEGARA, German Aerospace Center (DLR), Institute for Software Technology, Braunschweig, Germany
FEDERICO IURICICH, Clemson University, Clemson, USA
LEILA DE FLORIANI, University of Maryland, College Park, USA

We address the problem of performing a topology-aware simplification algorithm on a compact and distributed data structure for triangle meshes, the Terrain trees. Topology-aware operators have been defined to coarsen a Triangulated Irregular Network (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. However, their scalability is limited for large-scale meshes. Our proposed algorithm uses a batched processing strategy to reduce both the memory and time requirements of the simplification process, and thanks to the spatial decomposition on the basis of Terrain trees, it can be easily parallelized. Also, since a Terrain tree after the simplification process becomes less compact and efficient, we propose an efficient post-processing step for updating hierarchical spatial decomposition. Our experiments on real-world TINs, derived from topographic and bathymetric LiDAR data, demonstrate the scalability and efficiency of our approach. Specifically, topology-aware simplification on Terrain trees uses 40% less memory and half the time compared to the most compact and efficient connectivity-based data structure for TINs. Furthermore, the parallel simplification algorithm on the Terrain trees exhibits a 12× speedup with an OpenMP implementation. The quality of the output mesh is not significantly affected by the distributed and parallel simplification strategy of Terrain trees, and we obtain similar quality levels compared to the global baseline method.

CCS Concepts: • **Computing methodologies → Mesh geometry models**; *Shape analysis*; *Shared memory algorithms;* • **Information systems → Data structures**; *Geographic information systems*;

Additional Key Words and Phrases: Terrain simplification, edge contraction, spatial indexes, topological methods, shared memory processing

ACM Reference Format:
Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. 2024. Parallel Topology-aware Mesh Simplification on Terrain Trees. *ACM Trans. Spatial Algorithms Syst.* 10, 2, Article 7 (May 2024), 39 pages. https://doi.org/10.1145/3652602

This work has been partially supported by the US National Science Foundation under grant number IIS-1910766. It has also been performed under the auspices of the German Aerospace Center (DLR) under grant DLR-SC-2712024 (VisPlore). The Great Smokey Mountains, Canyon Lake, Yosemite Rim Fire, Dragons Back Ridge, and Moscow Mountain point clouds are kindly provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1948997, 1948994 & 1948857.

Authors' addresses: Y. Song and L. De Floriani, University of Maryland, 4600 River Road, Riverdale, Maryland 20737, USA; e-mails: ytsong@umd.edu, deflo@umd.edu; R. Fellegara, German Aerospace Center (DLR), Institute for Software Technology, Lilienthalpl. 7, Braunschweig, Germany, 38108; e-mail: riccardo.fellegara@dlr.de; F. Iuricich, Clemson University, McAdams Hall 305, Clemson, South Carolina 29631, USA; e-mail: fiurici@clemson.edu.

ACM 2374-0353/2024/05-ART7
https://doi.org/10.1145/3652602

1 INTRODUCTION

Morse theory is a powerful mathematical framework that 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 [41], chemistry [56], environmental science [69], forest monitoring [72], and urban analysis [25], to mention a few. Terrain analysis, particularly, the segmentation of a terrain according to its critical points (i.e., peaks and pits), provides information regarding terrain morphology that is fundamental for assessing the risk of landslides or floods. Terrain surfaces are usually described by either **Triangulated Irregular Networks (TINs)** or raster-based **Digital Elevation Models (DEMs)**. Although TINs can better adapt to irregularly distributed data, their usage is constrained by their large storage costs compared to DEMs. At the same time, the increasing availability of large point clouds [57] intensifies the need for scalable data representations for TINs.

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 [6, 16, 52]. These approaches reduce the morphological complexity of the dataset, while leaving the underlying digital 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. A possible solution is to lower the resolution of a terrain model, which, however, may affect its topology in an uncontrolled way, thereby deteriorating the simulation and segmentation results.

In Reference [46], we recently addressed this problem by defining a local simplification operator, called *gradient-aware edge contraction*. This operator is capable of reducing the resolution of a TIN while preserving the topology of the underlying terrain. 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 to reduce memory usage. Second, performing a large number of simplifications sequentially is time-consuming [46], directly affecting user interactions during data exploration. Thus, there is a need to develop a parallel simplification strategy.

In this work, we address both issues by designing and implementing a new simplification approach for triangulated terrains. The algorithm performs topology-aware simplification by extending gradient-aware edge contraction on a highly efficient data structure, the Terrain trees [27], which have been shown to be the most compact representation for triangulated terrains. Our approach reduces the geometric complexity of large triangulated terrains without affecting their morphology or incurring limitations due to processing time or space constraints. The major contributions of this article, which extends the work in Reference [68], include the following:

(1) the design and implementation of a topology-aware mesh simplification method on a compact 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 design and implementation of the simplification algorithm on Terrain trees and the evaluation of the computational performance were previously presented at ACM SIGSPATIAL 2021 [68], this article contains the full description of the Terrain tree update algorithm, experiments on simplified mesh quality, the discussion on the relationship between leaf capacity and mesh quality, and the description of an enhanced quality-oriented simplification strategy. In addition, compared to the conference paper [68], we provide a more comprehensive review of the background notions and related works.

The remainder of the article is organized as follows. Section 2 reviews some related background notions on discrete Morse theory. Section 3 discusses related work on triangle mesh simplification and on topology-based simplification. Section 4 briefly reviews the Terrain trees, which is the underlying data structure of our algorithms, while Section 5 provides the definition of a topology-aware edge contraction operator. Section 6 describes the proposed topology-aware simplification algorithm, and Section 7 defines a new parallel simplification algorithm relying on the distributed nature of Terrain trees. Section 8 describes an algorithm for updating a Terrain tree as a post-processing step, which is necessary to keep the downstream operations efficient. Section 9 discusses several mesh quality metrics for evaluating the output meshes generated by the proposed simplification algorithms. In Section 10, 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 a state-of-the-art compact data structure for meshes. Finally, in Section 11, we draw some concluding remarks and discuss directions for future work.

2 BACKGROUND

In this section, we review fundamental elements of discrete Morse theory, which is the basis for two-dimensional (2D) scalar field topology, but restricting ourselves to triangle meshes. Interested readers are referred to other work [19, 31] for a comprehensive overview of the theory and its application in shape analysis and visualization.

Morse theory [53] is a mathematical tool studying the relationships between the topology of a manifold shape M and the critical points of a smooth scalar function f defined over M. Based on Morse theory, we can define segmentations for shape M based on the regions of influence and the connectivity of its critical points. *Discrete Morse Theory* [31] is a combinatorial counterpart to Morse theory, which nicely extends the results of Morse theory to discrete data, and thus it has been used for analysis of 2D and 3D scalar fields [27, 41, 61, 70].

We consider a pair (Σ, F), where Σ is a triangle mesh and $F : \Sigma \longrightarrow \mathbb{R}$ 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 such an exception exists, then 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 pairs is called a *critical* simplex. We call a *V-path* a sequence of simplices $[\sigma_0, \tau_0, \ldots, \sigma_i, \tau_i, \ldots, \sigma_r, \tau_r]$ such that σ_i and σ_{i+1} are on the boundary of τ_i and (σ_i, τ_i) are paired simplices, where $i = 0, \ldots, r$. 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 is no closed V-path. A *separatrix V_j-path* is a V-path connecting two critical simplices of dimension $j+1$ and j, respectively.

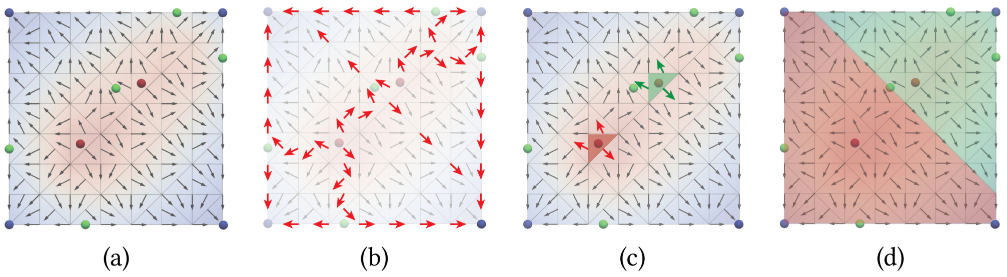


Fig. 1. (a) Forman gradient and (b) separatrix V-paths connecting pairs of critical simplices. Regions of influence of maxima are computed by visiting the V-paths (c) at the maxima and (d) until the whole region of influence is visited.

Given a triangle mesh Σ with elevation values defined at the vertices, we can extend it to a Forman function F defined on all simplices of Σ using the Robin's algorithm [61]. 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. Figure 1(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. It has been proved by Fugacci et al. [35] that critical simplices appear in correspondence of critical points defined for the piecewise linear surface approximation according to the theory by Banchoff [3].

For terrain analysis, the Forman gradient can be seen as the combinatorial counterpart of the gradient of the elevation function f [61, 70] and allows direct computation of different topological structures [19]. Figure 1(b) shows the separatrix V-paths connecting pairs of critical simplices, and the collection of such paths is referred to as a *critical net.* Moreover, the Forman gradient is also used to segment a dataset based on the regions of influence of its critical cells. Figure 1(c) shows the regions of influence for two critical triangles (maxima). Each region of influence is computed by starting from the gradient vectors outgoing the critical triangle and expanding the region recursively until no more gradient vector can be visited (see Figure 1(d)).

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* [26].

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. Figure 2 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.

3 RELATED WORK

In this section, we review state-of-the-art research related to this work. In Section 3.1, we review triangle mesh simplification methods, with a focus on the improvement of memory and time efficiency. In Section 3.2, we review topology-based simplification methods.

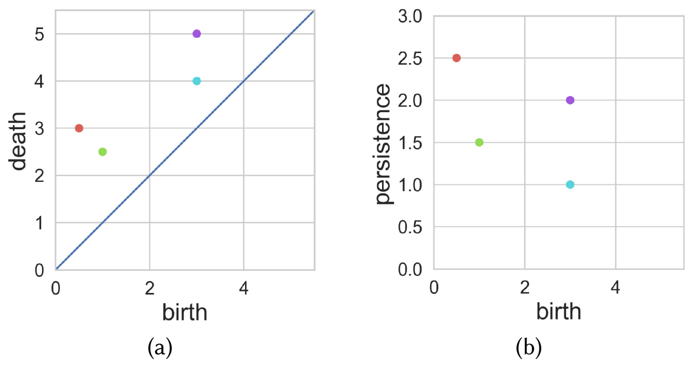


Fig. 2. Example of (a) a persistence diagram formed by four points and (b) its birth-persistence chart. Points with the same color in (a) and (b) correspond to the same point.

3.1 Triangle Mesh Simplification

The task of mesh simplification has been extensively studied in the literature [11, 42, 64]. In this section, we provide an overview of some techniques and challenges in triangle mesh simplification.

Popular techniques for mesh simplifications include vertex decimation [65], edge collapse [43], vertex clustering [62], and triangle collapse [44]. *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 [71]. Several more sophisticated metrics have been defined for optimizing the quality of output meshes, such as minimizing an energy function [43], setting a threshold on the Hausdorff distance between the original and the simplified models [47], minimizing the error quadrics [36], or setting a threshold on the error volume [38]. The **Quadric Error Metric (QEM)** [36] is one of the few approaches that 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* [23] and *fold condition* [9], 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 [46] 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 [7, 50, 51]. 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 [32] and vertex clustering [21], most parallel approaches rely on the edge contraction operator due to its intrinsic flexibility [7, 22, 32, 37, 49, 59]. 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. [32] first 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 [59] 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. [37] 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. [22] 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 [49] 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. [7] designed a parallel algorithm for distributed systems. The mesh is divided by a binary space partitioning [33] 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 [54] 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. Our parallel simplification method extends [7, 54] by considering the conflicts between different blocks in a more comprehensive way, by using a hierarchical decomposition of the domain, and by processing independent blocks at the same time.

3.2 Topology-based Mesh Simplification

In this work, we focus on topology-preserving simplifications, i.e., a simplification combining the need to reduce the size of a mesh with the need to maintain its topological properties, such as peaks and valleys. Typically, simplification operators are not topology-preserving, meaning that they can modify the number or connectivity of critical points of a terrain in an uncontrolled way.

This problem was first addressed in Reference [2] by introducing a simplification operator that preserves the critical points of the terrain. The operator removes a vertex from the terrain and re-triangulates the neighborhood such 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.

In Reference [18], the first efficient topology-aware operator based on edge contraction was introduced. 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 [46]. 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, then 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 [24] relaxed the original criteria to allow the removal of more critical simplices. While this operator does not lead to a multi-resolution model like HFT, it increases the number of admissible edge contractions and, thus, the compression rate. Recently, a new approach based on vertex removal was proposed by Fugacci et al. [34]. The simplification operator is similar to the one introduced in Reference [2], but in this case, the topology is preserved by checking a descriptor whose definition is rooted in algebraic topology, i.e., persistent homology [26]. 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.

In this work, we use the gradient-aware edge contraction from Reference [46], which allows for constructing the HFT multi-resolution model. Unlike vertex removal [34], the edge contraction operator comes with several metrics for controlling and optimizing the quality of the output meshes [36, 38, 43, 47].

4 TERRAIN TREES

In this section, we briefly introduce the data structure used in our work. We refer the reader to Reference [27] for more details. A variety of data structures have been developed for triangle meshes, and the most compact of them encode only the vertices and the triangles of the mesh [20]. Within this class of data structures we can find the **Indexed data structure with Adjacencies (IA data structure)** [58], the Corner Table [63], and the Sorted Opposite Table [40]. Recently, a new compact family of spatial data structures designed for triangulated terrain meshes, called *Terrain Trees*, has been developed [27]. Based on Terrain trees, we have developed the **Terrain Trees library (TTL)** [29], a library for terrain analysis, which contains a kernel for connectivity and spatial queries, as well as modules for morphological terrain analysis and for extracting topological structures, based on the discrete Morse gradient. Terrain trees are based on different nested subdivision strategies of the TIN domain D, which led to three data structures, called *PR*, *PM*, and *PMR Terrain trees*, respectively. In our experiments, the PR Terrain tree has been shown to be slightly more compact and efficient than the other two strategies. 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 article for the sake of simplicity.

A Terrain tree on a triangle mesh Σ consists of the following:

(1) a global vertex array Σ_V, encoding, for each vertex of Σ, its coordinates and elevation,
(2) a global triangle array Σ_T, encoding, for each triangle of Σ, a triplet of vertex indices in the global vertex array,
(3) a bucketed PR-quadtree T describing the nested subdivision of D, which acts as a bucketing structure for the mesh vertices, and
(4) a list of leaf blocks B obtained from the subdivision of D, where each leaf block b contains the vertices of the mesh that fall in b plus the triangles that intersect b.

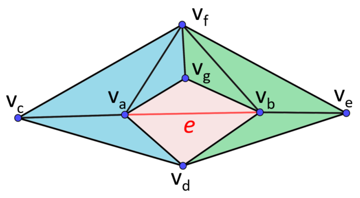


Fig. 3. Example of edge contraction not satisfying the link condition. Given edge $e = \{v_a, v_b\}$, we have $Lk(v_a) \bigcap Lk(v_b) = \{v_d, v_g, v_f, \{v_f, v_g\}\}$ and $Lk(e) = \{v_g, v_d\}$. The condition is not satisfied, since $Lk(v_a) \cap Lk(v_b)$ contains vertex v_f and edge $\{v_f, v_g\}$ that are not in $Lk(e)$.

The domain D of a TIN is referred to as the *root block* of a Terrain tree. The subdivision of D is guided by a capacity value kv on the vertices of the TIN. If a block contains more than kv vertices, then it is recursively split into four rectangle blocks of the same size. A block is called *internal* if it is split in the subdivision, and it is called *leaf* if it is not further split. When a block b is split, we call the resulting four blocks the *children* of b, and conversely, b is called the *parent* of its four children.

Each leaf block contains the minimum amount of information required for extracting all connectivity relations, encoded through a compression method based on **sequential range encoding (SRE)**, introduced in Reference [30]. This method, combined with a reindexing of the two global vertex and triangle arrays, enables a Terrain tree to encode a triangle mesh with low storage cost. It requires approximately 36% less storage than the most compact state-of-the-art mesh data structure (the IA data structure), while maintaining comparable performance in extracting connectivity relations. Moreover, the hierarchical domain decomposition of Terrain trees makes them well suited for parallel computation, as several leaf blocks can be processed simultaneously. These features make Terrain trees more scalable than other triangle-based data structures and desirable for representing large triangle meshes.

5 TOPOLOGY-AWARE EDGE CONTRACTION

Edge contraction is a widely used operator for triangle mesh simplification [43]. 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. An edge contraction operator can modify the shape of the TIN 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* [23] ensures that the simplified mesh has the same homological properties as the original one. The *link* $Lk(v)$ of a vertex v consists of all vertices adjacent to 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 $Lk(v_1) \cap Lk(v_2) \subseteq Lk(e)$. Figure 3 shows an example of an edge contraction that violates the link condition. If edge e is contracted to v_a, then 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* [9] 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* [46].

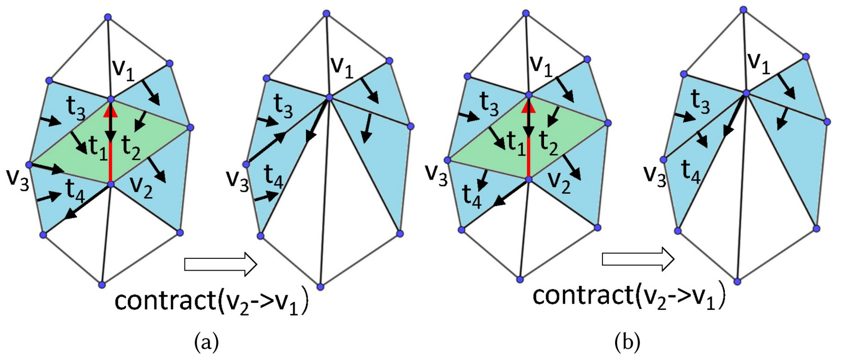


Fig. 4. Two possible gradient configurations and corresponding updates after contracting edge $\{v_1, v_2\}$ (in red) to v_1. Black arrows represent gradient pairs and red arrows represent contraction direction.

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 Figure 4(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.

A gradient-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, then, 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 Figure 4(a)), or with another triangle, t_4 (see Figure 4(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 Figure 4(a)), or with another triangle t_4 (see Figure 4(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 discrete gradient [46].

6 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. 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 [70]. As an error metric for edge contraction, we use the QEM [36]. Appendix A.1 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 *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, then 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, then function GET_VT directly retrieves $VT(v_1)$ and $VT(v_2)$ from array *local_vts*. Conversely, if e is a cross edge, and v_1 is contained by another leaf block b_1, then GET_VT must extract the VT relations of b_1. To optimize this latter step, we use an auxiliary **Least Recent Used (LRU)** cache C 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 C. If such relations are not in C, then they are extracted and saved to C. 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 v_1 or v_2 are extracted on the fly in LINK_CONDITION by traversing the VT relations of v_1 and v_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 v_2, the ET relation of e, the array of vertex error quadrics E, and TIN Σ. It removes vertex v_2 as well as the two triangles adjacent to e. In each remaining triangle in $VT(v_2)$, it replaces v_2 with v_1. After the contraction, the error quadric of v_1 is updated by adding the quadric of v_2 to it. The pseudo-code of function CONTRACT is in Algorithm 3 (see Appendix A.2).

After the contraction, both the Forman gradient V and the Terrain tree T are updated (rows 16–18). The update of gradient V (UPDATE_GRADIENT procedure at row 16) follows the method introduced in Section 5. This involves up to four triangles adjacent to triangles in $ET(e)$ (see Figure 4 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*, then the current leaf block b is updated by removing the index of v_2 and the indexes of the triangles incident in e. If e is a *cross edge* and v_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 v_2 but not encoded in b_1 are also added to b_1. The VT relation of vertex v_1 is updated by adding the triangles in the VT relation of v_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,

ALGORITHM 1: LEAF_SIMPLIFICATION(b, Σ, V, E, ω, C, b_R)

```
Input:
        b: current leaf block
        Σ: the TIN
        V: the Forman gradient on Σ
        E: the array of vertex error quadrics
        ω: the edge cost threshold
        C: 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($b$, $\Sigma_T$)
    // 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($b$, $\Sigma_T$, $E$, $\omega$)
 4: while $Q \neq \emptyset$ do
 5:     $e \leftarrow$ DEQUEUE($Q$) // $e = \{v_1, v_2\}$
        // 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:     $VT(v_1) \leftarrow$ GET_VT($v_1$, $local\_vts$, $C$, $b_R$, $\Sigma$)
10:     $VT(v_2) \leftarrow$ GET_VT($v_2$, $local\_vts$, $C$, $b_R$, $\Sigma$)
11:     $ET(e) \leftarrow$ GET_ET($e$, $VT(v_1)$)
        // Check three conditions introduced in Section 5 for $e$
12:     if LINK_CONDITION($e$, $VT(v_1)$, $VT(v_2)$, $ET(e)$)
13:         and FOLD_CONDITION($e$, $VT(v_2)$, $ET(e)$)
14:         and GRADIENT_CONDITION($e$, $VT(v_2)$, $ET(e)$, $V$) then
15:         CONTRACT($e$, $VT(v_2)$, $ET(e)$, $E$, $\Sigma$)
16:         UPDATE_GRADIENT($e$, $VT(v_1)$, $VT(v_2)$, $ET(e)$, $V$)
17:         UPDATE_INDEX($e$, $VT(v_2)$, $b$, $b_R$)
            // Update the VT relation of $v_1$
18:         VT($v_1$) $\leftarrow VT(v_1) \cup VT(v_2)$ - $ET(e)$
            // Update the cost of edges, and add these edges to $Q$
19:         $updated\_edges \leftarrow$ UPDATE_COSTS($v_1$, $VT(v_1)$, $E$, $Q$)
20:     end if
21: end while
22: $C \leftarrow C \cup local\_vts$ // Add $local\_vts$ to the LRU-cache
```

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 C (row 22).

7 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 6. In Section 7.1, we introduce the design of this parallel algorithm. The main challenge

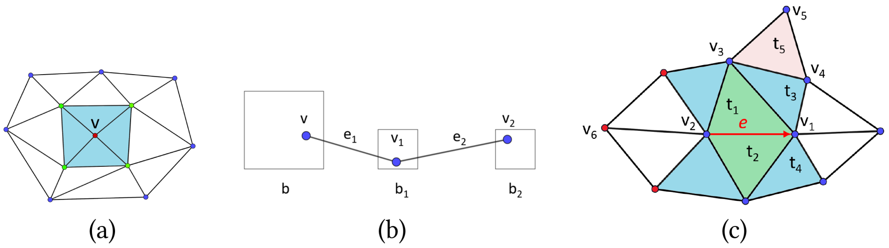


Fig. 5. (a) Example of 1-neighborhood (vertices in green) and 2-neighborhood (vertices in green and vertices in blue) of a vertex v. (b) Example of a vertex v and two vertices in its 2-neighborhood. (c) Example of edge contraction on edge $e = \{v_1, v_2\}$ (in red), where v_2, t_1, and t_2 are removed 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.

of parallel mesh simplification is to prevent conflicts that may occur if two threads modify the same vertex or triangle concurrently. In Section 7.2, we discuss why the proposed parallel algorithm can avoid such conflicts.

7.1 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.1.
(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 Figure 5(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, then the edge connecting v_6 and v_1 is a cross edge and may create a new conflict block for b_1.

To update the list of conflict blocks after contracting a cross edge, we modify the LINK_CONDITION procedure (row 12 of Algorithm 1) to extract 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. Similarly 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 Section 7.2, we show why the local caching strategy is thread-safe thanks to the leaf locking strategy.

We use OpenMP [17] 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.

7.2 Discussion on the Leaf Locking Strategy

We prove here that the *leaf locking strategy* introduced in Section 7.1 ensures that no conflict occurs between threads during a parallel simplification process. Given a vertex v in 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 Figure 5(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 V, 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 Figure 5(b)) or has a sharing adjacent vertex with v (e.g., v_2 in Figure 5(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, and (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 that shares a conflict block b_1 with b as shown in Figure 5(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 7.1. *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 7.1, 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 V during parallel simplification. From the description of the gradient-aware edge contraction in Section 5, we have the following proposition:

Proposition 7.2. *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 Figure 5(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 7.2, 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 Figure 5(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 7.1.

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 Figure 6(a)), then it is still possible that e_1 and 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 Figure 6(b)). Otherwise, like the example in Figure 6(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.

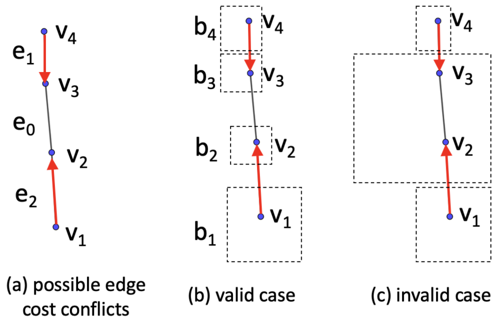


Fig. 6. (a) Example of a possible conflict occurring when two edges are contracted at the same time (triangles are not displayed for clarity). Edge $e_1 = \{v_1, v_2\}$ is contracted to v_2 and edge $e_2 = \{v_3, v_4\}$ is contracted to v_3. (b) A case that blocks encoding v_1 and v_4 can be active at 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.

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.

8 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.

As introduced in Section 6, 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 in which v_1 falls. 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.

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

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:
    $m$: a hash table stores triangles that need to be checked against other blocks
1:  if $b$ is leaf block then // Leaf block case
2:      $h_t \leftarrow \{\}$ // Create a hash set storing the triangles that should be kept in $b_t$
3:      for triangle $t$ in $b_t$ do
4:          if $t \in h_t$ 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:              $n \leftarrow$ VERTEX_IN_BLOCK($t$, $b$) // Count the number of vertices of $t$ in block $b$
9:              if $n \neq 3$ then // If $t$ is not fully contained by $b$
10:                 add $(t, l_b)$ to $m$
11:             end if
12:         end if
13:     end for
14:     $b_t \leftarrow h_t$ // Set the triangle list of current block to $h_t$
15: else // Internal block case
16:     $m_I \leftarrow \{\}$ // Create a hash table to store the triangles returned by its four sub-blocks
17:     for $i \leftarrow$ GET_CHILDREN($b$) do
18:         $c \leftarrow$ GET_CHILD($b$, $i$) // Get the i-th children of $b$
19:         add UPDATE_TRIANGLE_LISTS($c$, $i$) to $m_I$ // Merge the results to $m_I$
20:     end for
21:     for $t_i$ in GET_KEYS($m_I$) do
22:         $n \leftarrow$ VERTEX_IN_BLOCK($t_i$, $b$)
23:         if $n \neq 3$ then
24:             add $l_b$ to $m[t_i]$
25:         end if
26:         for $i \leftarrow$ GET_CHILDREN($b$) do
27:             $c \leftarrow$ GET_CHILD($b$, $i$)
                // If $t_i$ has not been checked against the sub-block $c$ and it intersects with $c$
28:             if $i \notin m_I[t_i]$ 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 $m$
```

(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 [27]. 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 kv 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 kv vertices, then 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, UPDATE_TRIANGLE_LISTS takes b and its label with respect to its parent block as inputs and returns a hash table m 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 m 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, then 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. Similarly 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 (rows 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 [30], 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, then 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 m (rows 9–11). The triangle list of b is updated to h_t (row 14).

When the current block b is an internal block (rows 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 m_I (rows 17–20). Given a triangle t in m_I, the value of m_I[t] is a list of sub-blocks of b that 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 m_I, we check first if t partially overlaps b and, if so, then the pair (t, l_b) is added to m and passed to the parent of b (rows 22–25). Besides, if t intersects with a sub-block c of b and c is not in $m_I[t]$, then t is inserted into the sub-tree rooted in c (rows 28–30, INSERT_TRIANGLE procedure).

9 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 [48], which evaluate the visual appearance difference between two meshes or by geometric-based metrics [64]. 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 [64].

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* [1, 12], 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 [10, 73, 74]. 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* [39]. 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 article, we use the triangle shape measure proposed by Guéziec [38],

$$\gamma = \frac{4\sqrt{3}\delta}{l_1^2 + l_2^2 + l_3^2}, \tag{1}$$

where δ is the area of the triangle and l_1, l_2, and 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 [66] and tree segmentation [72], 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, one of the most popular topological descriptors is the *persistence diagram* [26]. Numerous algorithms have been defined to compute the persistence diagram by efficiently pairing points on an input scalar function [45].

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 [14], the Wasserstein distance [15], or the sliced Wasserstein distance [5].

In this work, we use the sliced Wasserstein distance to measure the distance between two persistence diagrams, because faster to compute [5]. Formally, the qth Wasserstein distance between two persistence diagrams X and Y [15] is defined as

$$W_q(X, Y) = \left[\inf_{f:X \to Y} \sum_{x \in X} \|x - f(x)\|^q \right]^{1/q}, \tag{2}$$

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 [5] involves sampling this space by projecting all points onto a line passing through the origin. Subsequently, the Wasserstein distance is calculated in one dimension. In practice, the sliced Wasserstein distance is obtained by computing the one-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.

10 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 Section 10.1, we evaluate the computing performance of both algorithms. In Section 10.2, 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 Section 10.3, 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 Section 10.4. Additionally, we discuss how the selection of leaf block capacity for Terrain tree generation affects the output mesh quality in Section 10.5. 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 @2.20 GHz CPU (20 cores in total) and 64 GB of RAM. A total of six TINs, generated from raw point clouds using the *CGAL* library [8], are used in our comparisons. The number of vertices per TIN varies from 25 to 113 million (see Table 1). *Molokai* is a dataset consisting of both hydrographic and topographic point cloud data provided by NOAA National Centers for Environmental Information [55]. *Great Smokey Mountains*, *Canyon Lake*, *Yosemite Rim Fire*, *Dragons Back Ridge*, and *Moscow Mountain*, are topographic LiDAR point clouds from the OpenTopography repository [57]. 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 in Reference [67].

Table 1. Overview of Experimental Datasets

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

For each terrain, the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$ are listed.

Table 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

	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**

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 and blue.

10.1 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 Section 10.1.1, we compare the performance of the sequential topology-aware simplification on the Terrain trees against our implementation on the most compact triangle-based data structure for meshes, the IA data structure [58]. In Section 10.1.2, we compare the performance of the sequential and parallel simplification strategies implemented on the Terrain trees.

The generation of the Terrain tree we use in this article 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*. Reference [27] introduced a strategy to select the suitable block capacity for each dataset. 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.3. In the following, for each dataset, we use the capacity value showing the best tradeoff between simplification time and memory requirements.

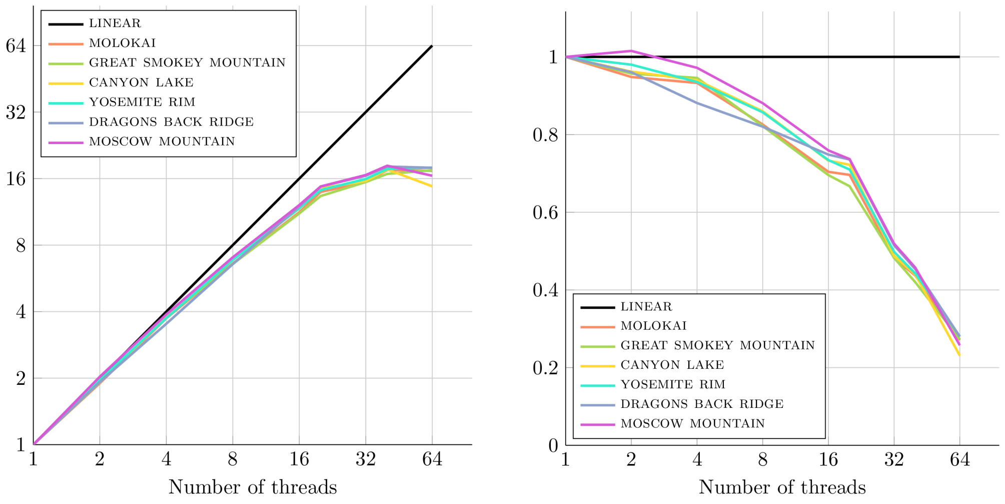
10.1.1 Topology-aware Mesh Simplification on the Terrain Tree and IA Data Structure. The IA data structure encodes a vertex array, containing the coordinates of the vertices of the TIN plus the elevation, and a triangle array that encodes, for each triangle t, the indexes to its three vertices plus the indexes in the triangle array of the three triangles sharing an edge with t. In our implementation [28], 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 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

Table 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

	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

The best performance value is denoted in bold and blue.



(a) Speedup (b) Efficiency

Fig. 7. (a) Speedup and (b) efficiency achieved by the parallel simplification algorithm when different numbers of threads are used.

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

Table 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.

10.1.2 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 7.1 when using from 1 to 64 threads.

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. Figure 7(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

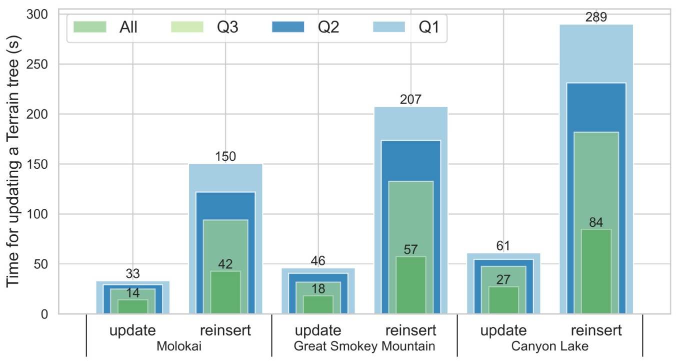


Fig. 8. The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original *reinsert* method. Q1, Q2, and Q3 represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. *All* represents 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.

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. Figure 7(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 tradeoff 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 12× speedup compared to the sequential strategy (see Table 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.

10.1.3 Performance of Terrain Trees Update. In Section 8, 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.4. Figure 8 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.

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

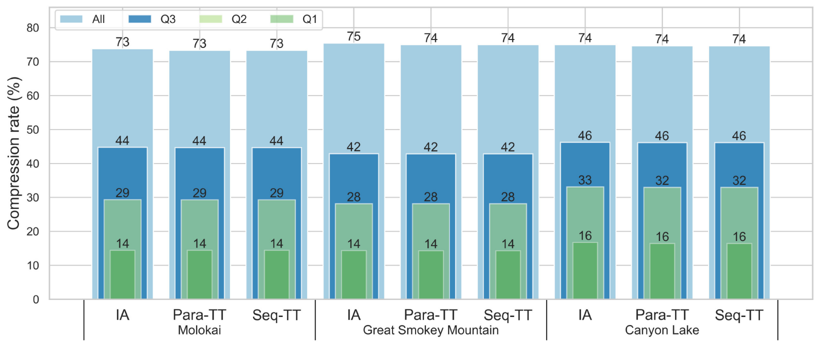


Fig. 9. The compression rates of three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (Q1, Q2, and Q3) and when all contractible edges are contracted (*All*).

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.

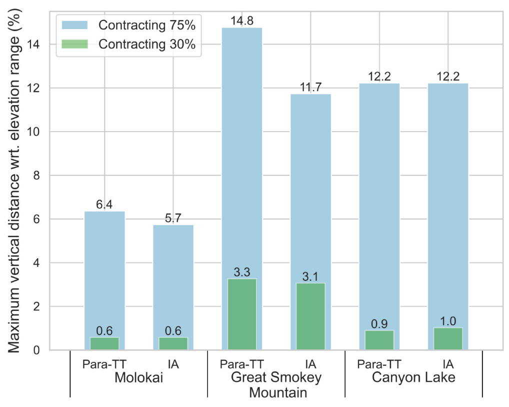
10.2 Compression Rate Evaluation

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.

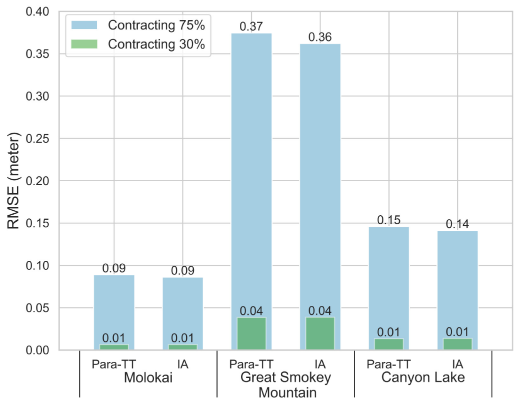
Similarly to the experiments in Section 10.1, we compute the compression rates of these three methods using different cost thresholds: the first quartile edge costs (Q1), the second quartile edge costs (Q2), and the third quartile edge costs (Q3) in the original mesh. Additionally, we compute the compression rate when we simplify all contractible edges, which we refer to as *All*. The results, illustrated in Figure 9, 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. However, the simplification process on the Terrain trees simplifies the mesh block by block, so the sequence of contractions is not the global optimum.

10.3 Mesh Quality Evaluation After Simplification

In this subsection, we evaluate the quality of the simplified meshes by using the metrics introduced in Section 9. In Section 10.3.1, we evaluate the triangle shape quality of the simplified mesh using the triangle shape measure. In Section 10.3.2, we evaluate the approximation error of a simplified



(a) Maximum distance wrt. range



(b) RMSE

Fig. 10. (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.

mesh by calculating the maximum and the RMSE of vertical distances. In Section 10.3.3, 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 5. Specifically, we refer to the parallel geometric simplification on Terrain trees as the *Geometric-TT* method.

10.3.1 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 Appendix A.4. 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.

10.3.2 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. Figure 10(a) and (b) show the maximum and the RMSE of vertical

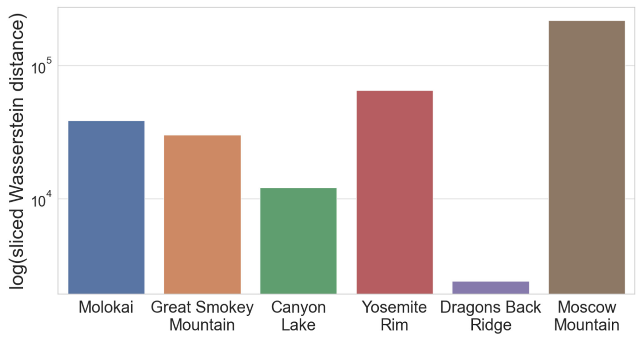


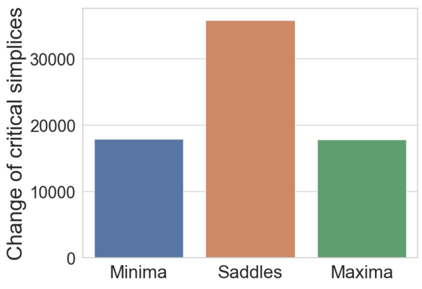
Fig. 11. The sliced Wasserstein distance between the original mesh and the simplified mesh when the mesh is simplified with the Geometric-TT method.

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

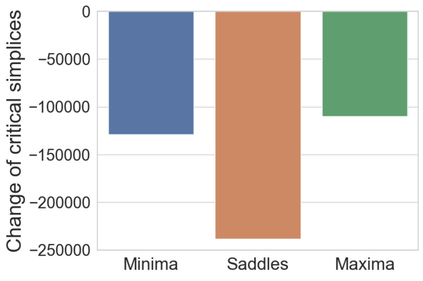
When 75% edges are contracted, the IA-based method produces meshes with 10–21% smaller maximum distance and 3% smaller RMSE compared to the Terrain trees. The usage of the local queues in the Terrain trees changes the sequence of contractions, which leads to the difference in both the maximum vertical distance and the RMSE. However, this change is dataset dependent, and there is no significant quality difference between the two methods when fewer edges are contracted. Similarly 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.

10.3.3 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 [60], 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. Figure 11 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 9, 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.



(a) Dragons Back Ridge



(b) Moscow Mountain

Fig. 12. The change in the number of critical simplices in two tested dataset when they are simplified with the Geometric-TT method.

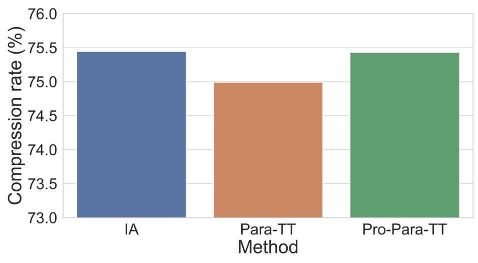
Among six datasets, it is noteworthy that the sliced Wasserstein distance of the *Dragons Back Ridge* dataset is much smaller compared to the others. This happens since the percentage of critical simplices in this dataset is significantly lower, with only 0.05% of simplices being critical, compared to 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 Figure 12, 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 the *Moscow Mountain* dataset, the numbers of minima, saddles, and maxima are reduced by 22%. The changes in other datasets (see figures in Appendix A.4) 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.

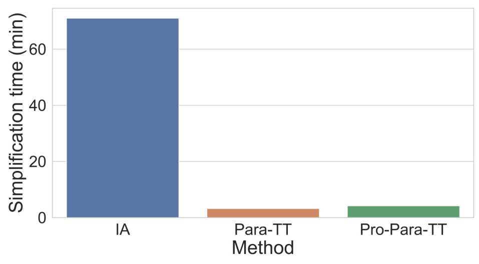
These observations suggest that, when a mesh is simplified with the geometric method, the number of critical simplices in the terrain may either increase or decrease, depending on the terrain topology. In contrast, the number of critical simplices remains constant when a mesh is simplified by a topology-aware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

10.4 Progressive Strategy for Improving Output Mesh Quality

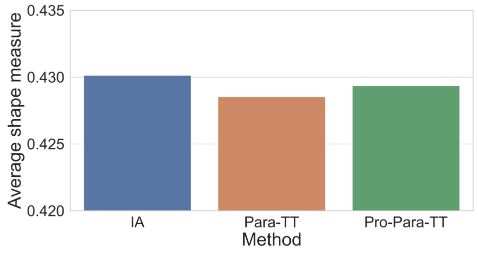
As shown in the above experiments, the simplification on the Terrain trees sometimes produces meshes with higher approximation errors and a lower compression rates compared to the IA-based method, especially when the cost threshold is huge. This difference is mainly caused by the local queues used by 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



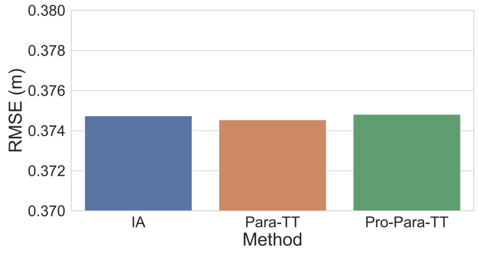
(a) Compression rate



(b) Simplification time



(c) Average shape measure



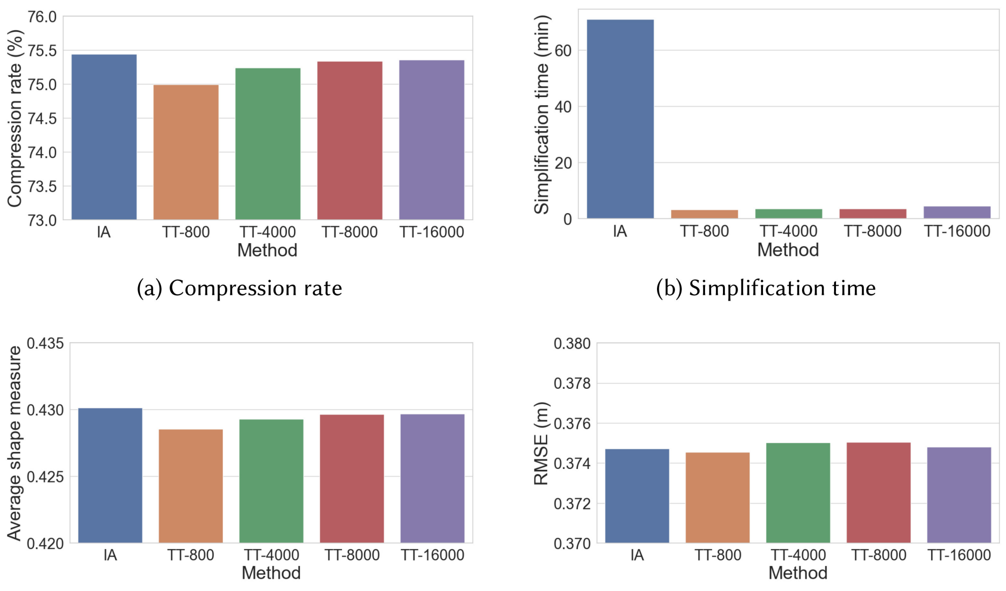
(d) RMSE of vertical distances

Fig. 13. The performance and quality evaluation of the parallel simplification on Terrain trees when a progressive optimization is applied to the *Great Smokey Mountains* dataset.

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.4.

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 1,000 to be sufficiently large to contract most normal edges while avoiding edges with abnormally high costs that could significantly deteriorate the mesh quality. For the IA-based method and the Para-TT method, all contractible edges are contracted. Figure 13 shows the simplification time and quality of the simplified meshes using 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



(a) Compression rate (b) Simplification time

(c) Average shape (d) RMSE of vertical distances

Fig. 14. 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.

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 tradeoff 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.

10.5 Selection of Leaf Capacity on Mesh Quality

In Appendix A.3, 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.4. 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.3 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 quality measures when all contractible edges are contracted.

Figure 14 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 16,000. Similarly, the difference in the average shape measure changes from 0.37% to 0.1% when the capacity increases from 800 to 16,000. 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 4,000, 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 16,000, 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.

11 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 Reference [46], 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 [17]. Comparing the sequential and parallel strategies based on Terrain trees, we observed a further performance increase. The parallel strategy achieved a 12× 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.3 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 10.5, 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 a persistence-based cancellation operator introduced in [46].

The parallel strategy developed here can be easily extended to other topology-aware edge contraction operators, such as the one introduced in Reference [24], since the range of simplices involved in those topology-preserving conditions is the same as for our gradient-aware simplification operator. While the gradient-aware contraction operator is efficient and produces high-quality meshes, 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 the following 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 [4].

Our current parallel strategy is compatible with shared-memory processing based on OpenMP [17]. In the future, we want to check if it is possible to increase its efficiency by using specialized compilers, such as ISPC,[1] and libraries, such as TBB.[2] Last, a natural extension of our simplification method is to support large-scale data processing through the distributed-memory processing strategy based on MPI [13].

A APPENDIX

A.1 Computing the Quadric Error Matrix on Terrain Trees

The QEM [36] 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 P is calculated as $\Delta_P(v) = v^T K_P v$, where K_P 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, then skip t; otherwise, proceed to step (2);
(2) Calculate the fundamental error quadric K_P of the plane on which t lies;
(3) For each vertex v of t, if v is contained in b, then add K_P 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

[1]https://ispc.github.io/
[2]https://software.intel.com/en-us/oneapi/onetbb

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.2 Performing an Edge Contraction

ALGORITHM 3: CONTRACT(e, VT(v_j), ET(e), E, Σ)

```
Input:
        $e = \{v_i, v_j\}$: edge to be contracted to $v_i$
        $VT(v_j)$: the Vertex-Triangle relation of $v_j$
        $ET(e)$: the Edge-Triangle relation of $e$
        $E$: the array of vertex error quadrics
        $\Sigma$: the triangulated terrain
1: for each $t$ in $ET(e)$ do
2:     $\Sigma \leftarrow \Sigma - \{t\}$ // Remove $t$ from $\Sigma$
3: end for
   // For each triangle $t$ incident in $v_j$ but not adjacent to $e$
4: for each $t$ in $(VT(v_j) - ET(e))$ do
5:     $t \leftarrow (t - v_j) \cup v_i$ // Replace $v_j$ with $v_i$ in $t$
6: end for
7: $E[i] \leftarrow E[i] + E[j]$ // Update the error quadric at vertex $v_i$
8: $\Sigma \leftarrow \Sigma - \{v_j\}$ // Remove $v_j$ from $\Sigma$
```

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 v_j (row 2 and row 8). In each remaining triangle in $VT(v_j)$, it replaces v_j with v_i (row 4 to 6). After the contraction, the error quadric of the remaining vertex v_i is updated by adding the quadric of v_j to it (row 7).

A.3 Experiments on Leaf Capacity Selection

In this section, we evaluate the performance of the simplification algorithm with different capacity thresholds 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 1/100,000 and 1/30,000 of the total number of vertices in the data set. This is 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 4 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

Table 4. 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

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	1,200	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	**1,000**	100.1	39.0	70.6	**1,200**	99.8	45.9	80.5	1,500	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	1,200	100.5	39.4	70.6	1,500	129.2	46.0	80.6	1,800	188.7	57.4	77.6
700	29.0	12.7	73.3	700	39.8	17.3	75.0	1,050	68.4	24.9	74.7	1,400	100.3	39.4	70.6	1,800	126.7	45.9	80.6	2,100	172.0	57.5	77.6
800	29.5	12.7	73.4	**800**	39.1	17.2	75.0	1,200	66.5	24.6	74.7	1,600	100.9	39.3	70.6	2,100	127.9	46.1	80.6	**2,400**	170.9	57.4	77.7
900	29.1	12.7	73.4	900	39.4	17.3	75.0	1,350	66.7	24.8	74.7	1,800	102.2	39.2	70.6	2,400	131.7	45.7	80.6	2,700	176.9	57.3	77.7
1,000	29.2	12.7	73.4	1,000	39.4	17.3	75.0	1,500	73.2	24.8	74.8	2,000	106.7	38.9	70.7	2,700	131.8	46.1	80.6	3,000	176.3	56.9	77.7
1,100	29.2	12.8	73.4	1,100	39.6	17.4	75.0	1,650	75.5	24.7	74.8	2,200	111.4	39.3	70.7	3,000	129.8	46.1	80.6	3,300	177.2	57.3	77.7
1,200	29.8	12.7	73.4	1,200	39.5	17.3	75.1	1,800	73.3	24.6	74.8	2,400	106.8	39.2	70.7	3,300	129.6	46.0	80.6	3,600	173.9	57.2	77.7

The capacity value in bold is the capacity value used to generate the Terrain tree of each dataset.

still performs well. For the experiments presented in the article, 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 4 in bold.

A.4 Additional Experiment Results

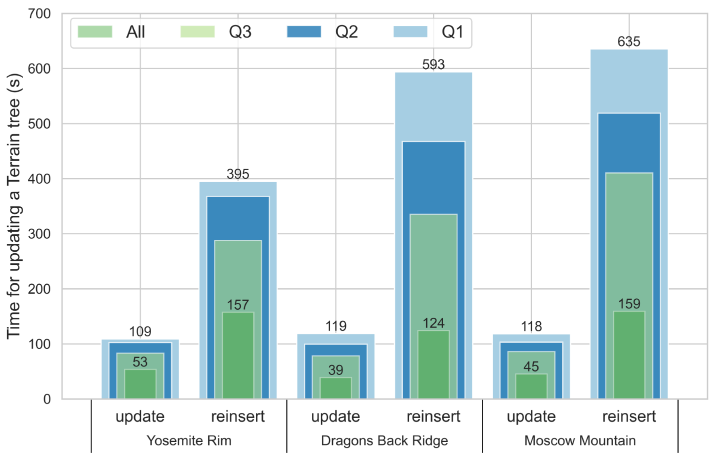


Fig. 15. The time for updating Terrain trees when using two different methods: (1) new *update* algorithm and (2) original Q1, Q2, and Q3 represent the result when the cost threshold is set to the first, second, and third quartile edge costs of each dataset, respectively. *All* represents the result 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.

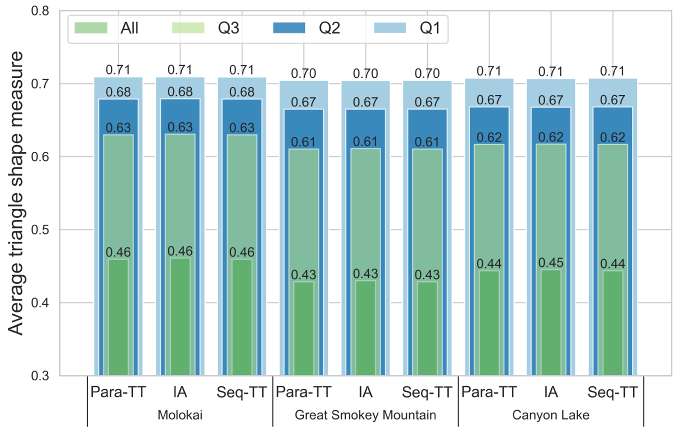


Fig. 16. The average triangle shape measure of meshes obtained from three simplification methods IA, Seq-TT, and Para-TT when the cost threshold is set to three edge cost quartiles (Q1, Q2, and Q3) and when all contractible edges are contracted (*All*).

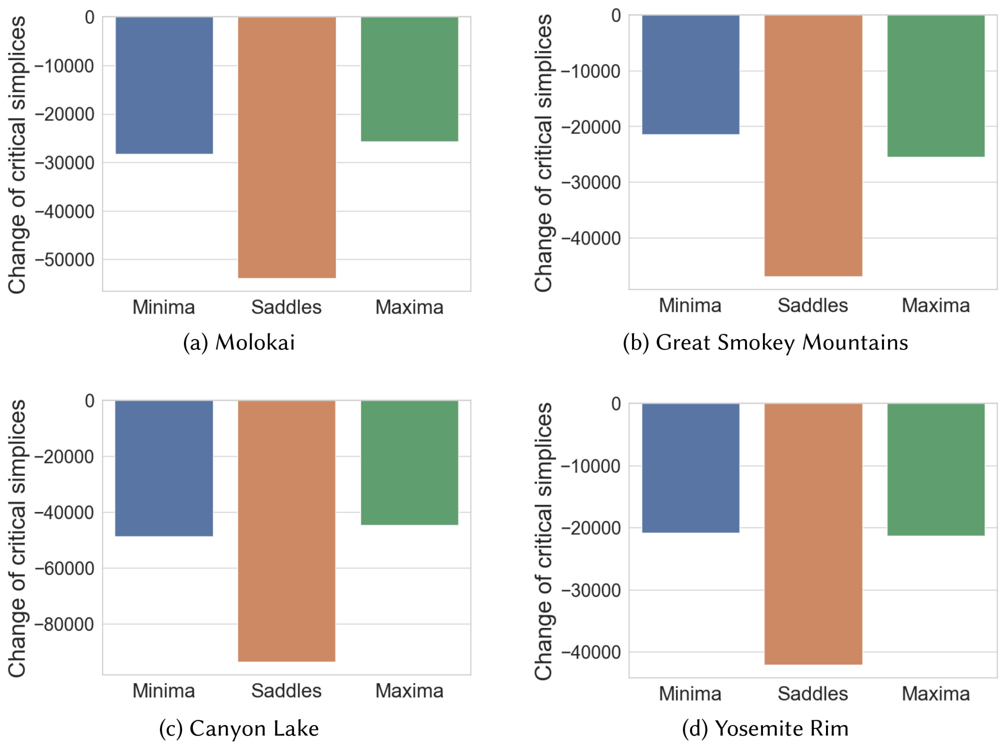
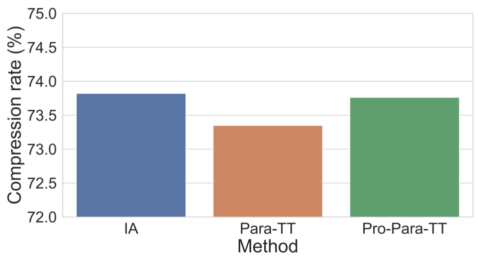
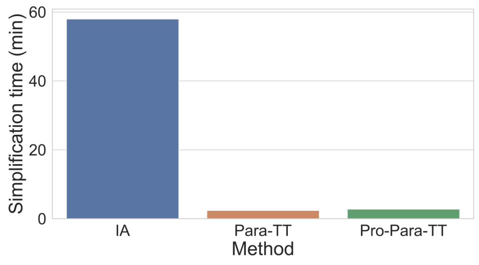


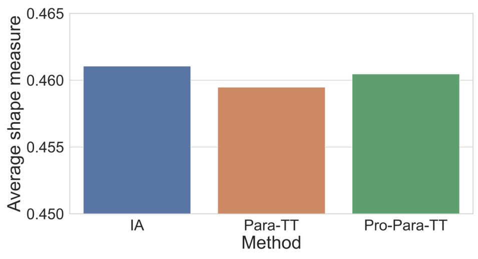
Fig. 17. The change in the number of critical simplices in each dataset when it is simplified with the Geometric-TT method.



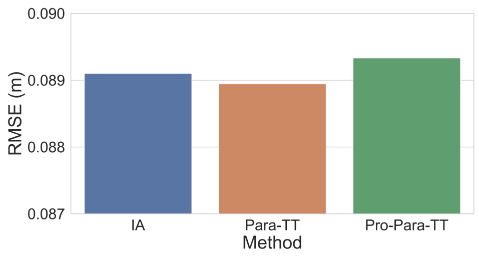
(a) Compression rate



(b) Simplification time

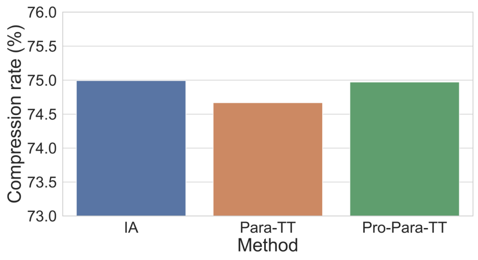


(c) Average shape measure

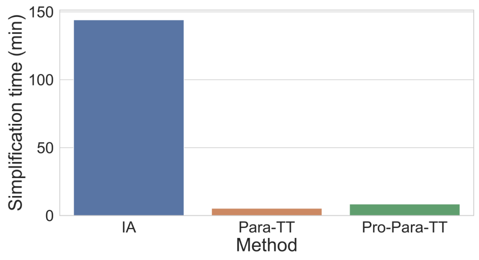


(d) RMSE of vertical distances

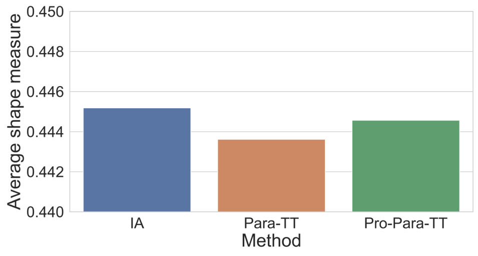
Fig. 18. The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Molokai* dataset.



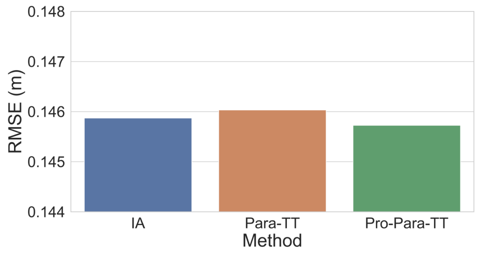
(a) Compression rate



(b) Simplification time

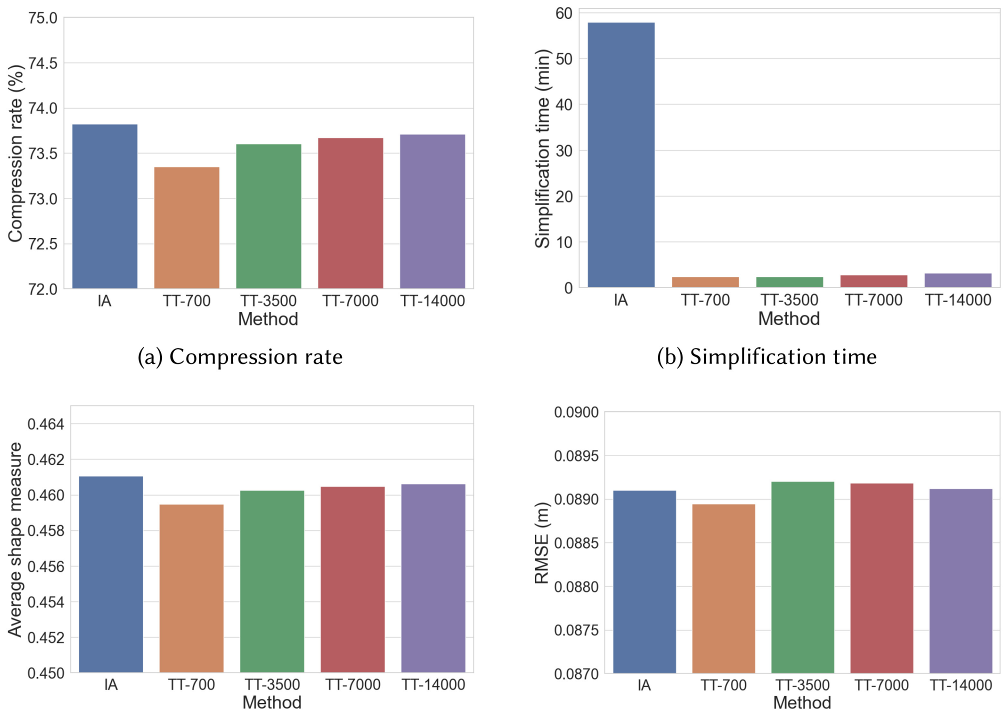


(c) Average shape measure



(d) RMSE of vertical distances

Fig. 19. The performance and quality evaluation of the parallel simplification when a progressive optimization is applied to the *Canyon Lake* dataset.



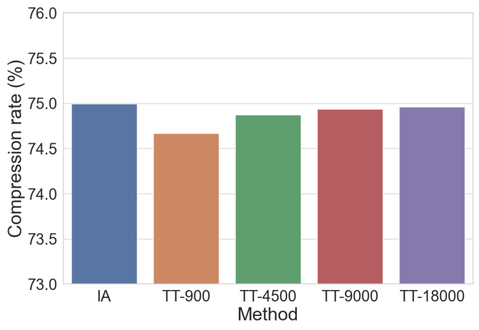
(a) Compression rate

(b) Simplification time

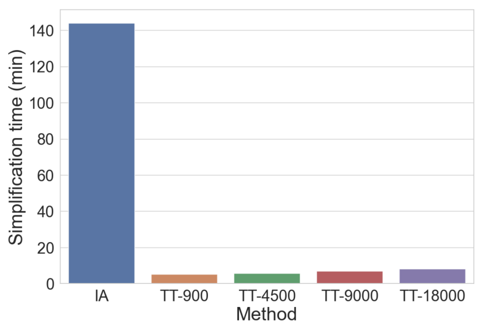
(c) Average shape measure

(d) RMSE of vertical distances

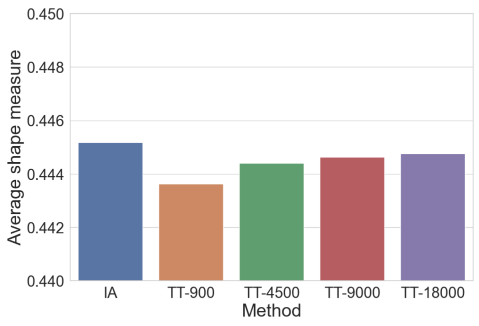
Fig. 20. The performance and quality evaluation of the simplification of the *Molokai* dataset when it is simplified by the parallel simplification on Terrain trees with different leaf capacity values and by the simplification on the IA data structure. The *TT-* refers to the Terrain trees method, and the number after it corresponds leaf capacity value used during the Terrain tree generation.



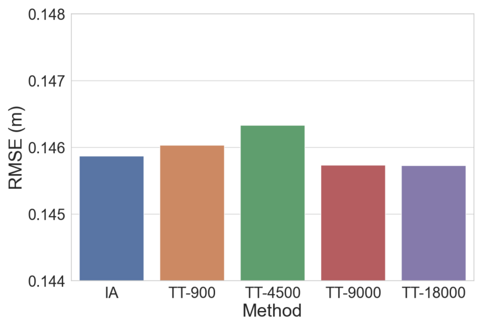
(a) Compression rate



(b) Simplification time



(c) Average shape measure



(d) RMSE of vertical distances

Fig. 21. 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.

ACKNOWLEDGMENTS

The Molokai point cloud is kindly provided by NOAA National Centers for Environmental Information.

REFERENCES

[1] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. 2002. MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '02)*, Vol. 1. 705–708. DOI: https://doi.org/10.1109/ICME.2002.1035879

[2] Chandrajit L. Bajaj and Daniel R. Schikore. 1998. Topology preserving data simplification with error bounds. *Comput. Graph.* 22, 1 (1998), 3–12. DOI: https://doi.org/10.1016/S0097-8493(97)00079-4

[3] Thomas F. Banchoff. 1970. Critical points and curvature for embedded polyhedral surfaces. *Am. Math. Monthly* 77, 5 (1970), 475–485.

[4] Matthew V. Bilskie, Scott C. Hagen, and Stephen C. Medeiros. 2020. Unstructured finite element mesh decimation for real-time Hurricane storm surge forecasting. *Coast. Eng.* 156 (2020), 103622. DOI: https://doi.org/10.1016/j.coastaleng.2019.103622

[5] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. 2015. Sliced and radon wasserstein barycenters of measures. *J. Math. Imag. Vis.* 51, 1 (2015), 22–45.

[6] Peer-Timo Bremer, Valerio Pascucci, and Bernd Hamann. 2009. Maximizing adaptivity in hierarchical topological models using cancellation trees. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Springer, Berlin, 1–18. DOI: https://doi.org/10.1007/b106657_1

[7] Daniela Cabiddu and Marco Attene. 2015. Large mesh simplification for distributed environments. *Comput. Graph.* 51, 1 (2015), 81–89. DOI: https://doi.org/10.1016/j.cag.2015.05.015

[8] CGAL 2021. Computational Geometry Algorithms Library (CGAL). Retrieved April 2021 from https://www.cgal.org/
[9] Patrick Ciarlet and Françoise Lamour. 1996. Does contraction preserve triangular meshes? *Numer. Algor.* 13, 2 (1996), 201–223.
[10] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. 2003. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Comput. Graph. Forum* 22, 3 (2003), 505–514.
[11] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998. A comparison of mesh simplification algorithms. *Comput. Graph.* 22, 1 (1998), 37–54. DOI: https://doi.org/10.1016/S0097-8493(97)00082-4
[12] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. 1998. Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum* 17, 2 (1998), 167–174.
[13] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. 1994. The MPI message passing interface standard. In *Programming Environments for Massively Parallel Distributed Systems*. Birkhäuser Basel, Basel, 213–218.
[14] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. 2005. Stability of persistence diagrams. In *Proceedings of the 21st Annual Symposium on Computational Geometry (SCG '05)*. Association for Computing Machinery, New York, NY, 263–271. DOI: https://doi.org/10.1145/1064092.1064133
[15] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. 2010. Lipschitz functions have L p-stable persistence. *Found. Comput. Math.* 10, 2 (2010), 127–139.
[16] Lidija Čomić, Leila De Floriani, and Federico Iuricich. 2012. Dimension-independent multi-resolution morse complexes. *Comput. Graph.* 36, 5 (Aug. 2012), 541–547. DOI: https://doi.org/10.1016/j.cag.2012.03.010
[17] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An industry standard API for shared-memory programming. *Comput. Sci. Eng.* 5, 1 (1998), 46–55.
[18] Emanuele Danovaro, Leila De Floriani, Paola Magillo, Mohammed Mostefa Mesmoudi, and Enrico Puppo. 2003. Morphology-driven simplification and multiresolution modeling of terrains. In *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (GIS '03)*. Association for Computing Machinery, New York, NY, 63–70. DOI: https://doi.org/10.1145/956676.956685
[19] Leila De Floriani, Ulderico Fugacci, Federico Iuricich, and Paola Magillo. 2015. Morse complexes for shape segmentation and homological analysis: Discrete models and algorithms. *Comput. Graph. Forum* 34, 2 (2015), 761–785. DOI: https://doi.org/10.1111/cgf.12596
[20] Leila De Floriani and Annie Hui. 2005. Data structures for simplicial complexes: An analysis and a comparison. In *Proceedings of the 3rd Eurographics Symposium on Geometry Processing (SGP '05)*. Eurographics Association, Eindhoven, The Netherlands, 119–128.
[21] Christopher DeCoro and Natalya Tatarchuk. 2007. Real-time mesh simplification using the GPU. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*. 161–166.
[22] Frank Dehne, Chiristian Langis, and Gerhard Roth. 2000. Mesh simplification in parallel. In *Algorithms and Architectures for Parallel Processing (ICA3PP '00)*. World Scientific, Singapore, 281–290.
[23] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. 1998. Topology preserving edge contraction. *Publ. l'Inst. Math.* 66, 80 (1998), 23–45.
[24] Tamal K. Dey and Ryan Slechta. 2018. Edge contraction in persistence-generated discrete Morse vector fields. *Comput. Graph.* 74, 1 (2018), 33–43. DOI: https://doi.org/10.1016/j.cag.2018.05.002
[25] Tamal K. Dey, Jiayuan Wang, and Yusu Wang. 2017. Improved road network reconstruction using discrete morse theory. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 1–4. DOI: https://doi.org/10.1145/3139958.3140031
[26] Herbert Edelsbrunner and John Harer. 2008. Persistent homology-a survey. *Contemp. Math.* 453, 26 (2008), 257–282.
[27] Riccardo Fellegara, Federico Iuricich, Yunting Song, and Leila De Floriani. 2023. Terrain trees: A framework for representing, analyzing and visualizing triangulated terrains. *GeoInformatica* 27, 3 (2023), 525–564. DOI: https://doi.org/10.1007/s10707-022-00472-3
[28] Riccardo Fellegara and Yunting Song. 2022. Terrain Analysis on the IA Data Structure. Retrieved February 2024 from https://github.com/UMDGeoVis/Terrain_Analysis_on_IA [Online; accessed February-2024].
[29] Riccardo Fellegara and Yunting Song. 2023. Terrain Trees Library. https://zenodo.org/records/10714553
[30] Riccardo Fellegara, Kenneth Weiss, and Leila De Floriani. 2021. The Stellar decomposition: A compact representation for simplicial complexes and beyond. *Comput. Graph.* 98, 1 (2021), 322–343. DOI: https://doi.org/10.1016/j.cag.2021.05.002
[31] Robin Forman. 1998. Morse theory for cell complexes. *Adv. Math.* 134, 1 (1998), 90–145.
[32] Martin Franc and Václav Skala. 2000. Parallel triangular mesh reduction. In *Proceedings of Scientific Computing (ALGORITMY '00)*. 357–367.
[33] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. 1980. On visible surface generation by a priori tree structures. *SIGGRAPH Comput. Graph.* 14, 3 (July 1980), 124–133. DOI: https://doi.org/10.1145/965105.807481

[34] Ulderico Fugacci, Michael Kerber, and Hugo Manet. 2020. Topology-preserving terrain simplification. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20)*. ACM, New York, NY, 36–47. DOI: https://doi.org/10.1145/3397536.3422237

[35] Ulderico Fugacci, Claudia Landi, and Hanife Varlı. 2020. Critical sets of PL and discrete morse theory: A correspondence. *Comput. Graph.* 90, 1 (2020), 43–50. DOI: https://doi.org/10.1016/j.cag.2020.05.020

[36] Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley, 209–216. DOI: https://doi.org/10.1145/258734.258849

[37] Nico Grund, Evgenij Derzapf, and Michael Guthe. 2011. Instant level-of-detail. In *Vision, Modeling, and Visualization (2011)*. Peter Eisert, Joachim Hornegger, and Konrad Polthier (Eds.). The Eurographics Association, 293–299. DOI: https://doi.org/10.2312/PE/VMV/VMV11/293-299

[38] André Guéziec. 1999. Locally toleranced surface simplification. *IEEE Trans. Vis. Comput. Graph.* 5, 2 (1999), 168–189.

[39] Leonidas Guibas and Jorge Stolfi. 1985. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.* 4, 2 (Apri. 1985), 74–123. DOI: https://doi.org/10.1145/282918.282923

[40] Topraj Gurung and Jarek Rossignac. 2009. SOT: A compact representation for tetrahedral meshes. In *Proceedings SIAM/ACM Geometric and Physical Modeling (SPM '09)*. ACM, New York, 79–88. DOI: https://doi.org/10.1145/1629255.1629266

[41] Attila Gyulassy, Mark Duchaineau, Vijay Natarajan, Valerio Pascucci, Eduardo Bringa, Andrew Higginbotham, and Bernd Hamann. 2007. Topologically clean distance fields. *IEEE Trans. Vis. Comput. Graph.* 13, 6 (Nov. 2007), 1432–1439. DOI: https://doi.org/10.1109/TVCG.2007.70603

[42] Paul S. Heckbert and Michael Garland. 1997. *Survey of Polygonal Surface Simplification Algorithms*. Carnegie Mellon University Technical Report. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Pittsburgh, PA.

[43] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*. ACM, New York, NY, 19–26. DOI: https://doi.org/10.1145/166117.166119

[44] Veysi İşler, Rynson W. H. Lau, and Mark Green. 1996. Real-time multi-resolution modeling for complex virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '96)*. Association for Computing Machinery, New York, NY, 11–19. DOI: https://doi.org/10.1145/3304181.3304186

[45] Federico Iuricich. 2021. Persistence cycles for visual exploration of persistent homology. *IEEE Trans. Vis. Comput. Graph.* 28, 12 (2021), 4966–4979.

[46] Federico Iuricich and Leila De Floriani. 2017. Hierarchical forman triangulation: A multiscale model for scalar field analysis. *Comput. Graph.* 66, 1 (2017), 113–123. DOI: https://doi.org/10.1016/j.cag.2017.05.015

[47] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. 1996. Mesh reduction with error control. In *Proceedings of the 7th Conference on Visualization '96 (VIS '96)*. IEEE Computer Society Press, Washington, DC, 311–318.

[48] Guillaume Lavoue and Massimiliano Corsini. 2010. A comparison of perceptually-based metrics for objective evaluation of geometry processing. *IEEE Trans. Multimedia* 12, 7 (2010), 636–649. DOI: https://doi.org/10.1109/TMM.2010.2060475

[49] Hyunho Lee and Min-Ho Kyung. 2016. Parallel mesh simplification using embedded tree collapsing. *Vis. Comput.* 32, 6 (2016), 967–976.

[50] Peter Lindstrom. 2000. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. 259–262.

[51] Peter Lindstrom and Valerio Pascucci. 2002. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Trans. Vis. Comput. Graph.* 8, 3 (2002), 239–254.

[52] Jonas Lukasczyk, Christoph Garth, Ross Maciejewski, and Julien Tierny. 2021. Localized topological simplification of scalar data. *IEEE Trans. Vis. Comput. Graph.* 27, 2 (Feb. 2021), 572–582. DOI: https://doi.org/10.1109/TVCG.2020.3030353

[53] Yukio Matsumoto. 2002. *An Introduction to Morse Theory*. Vol. 208. American Mathematical Soc.

[54] Mohamed H. Mousa and Mohamed K. Hussein. 2021. High-performance simplification of triangular surfaces using a GPU. *PLos One* 16, 8 (2021), e0255832.

[55] OCM Partners. 2021. 2013 USACE NCMP Topobathy Lidar: Molokai (HI). (2021). NOAA National Centers for Environmental Information. Retrieved April 2021 from https://www.fisheries.noaa.gov/inport/item/49753

[56] Małgorzata Olejniczak, André Severo Pereira Gomes, and Julien Tierny. 2020. A topological data analysis perspective on noncovalent interactions in relativistic calculations. *Int. J. Quant. Chem.* 120, 8 (Apr. 2020), e26133. DOI: https://doi.org/10.1002/qua.26133

[57] OpenTopography 2020. OpenTopography—High-Resolution Topography Data and Tools. Retrieved Janauary 2020 from http://www.opentopography.org/

[58] Alberto Paoluzzi, Fausto Bernardini, Carlo Cattani, and Vincenzo Ferrucci. 1993. Dimension-independent modeling with simplicial complexes. *ACM Trans. Graph.* 12, 1 (1993), 56–102.
[59] Alexandros Papageorgiou and Nikos Platis. 2015. Triangular mesh simplification on the GPU. *Vis. Comput.* 31, 2 (2015), 235–244. DOI: https://doi.org/10.1007/s00371-014-1039-x
[60] Persim 2022. Persim 0.3.1 Documentation. Retrieved June 2022 from https://persim.scikit-tda.org/en/latest/index.html
[61] Vanessa Robins, Peter John Wood, and Adrian P. Sheppard. 2011. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 8 (2011), 1646–1658. DOI: https://doi.org/10.1109/TPAMI.2011.95
[62] Jarek Rossignac and Paul Borrel. 1993. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*. Springer, Berlin, 455–465.
[63] Jarek Rossignac, Alla Safonova, and Andrzej Szymczak. 2003. Edgebreaker on a corner table: A simple technique for representing and compressing triangulated surfaces. In *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer, 41–50.
[64] Hanan Samet. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.
[65] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. 1992. Decimation of triangle meshes. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '92)*. ACM, New York, NY, 65–70. DOI: https://doi.org/10.1145/133994.134010
[66] Andriani Skopeliti, Leda Stamou, Lysandros Tsoulos, and Shachak Pe'eri. 2020. Generalization of soundings across scales: From DTM to harbour and approach nautical charts. *ISPRS Int. J. Geo-Inf.* 9, 11 (2020), 693.
[67] Yunting Song and Riccardo Fellegara. 2023. Topology-aware Simplification on Terrain Trees. https://zenodo.org/records/10723706
[68] Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. 2021. Efficient topology-aware simplification of large triangulated terrains. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21)*. 576–587. DOI: https://doi.org/10.1145/3474717.3484261
[69] Akash Anil Valsangkar, Joy Merwin Monteiro, Vidya Narayanan, Ingrid Hotz, and Vijay Natarajan. 2019. An exploratory framework for cyclone identification and tracking. *IEEE Trans. Vis. Comput. Graph.* 25, 3 (Mar. 2019), 1460–1473. DOI: https://doi.org/10.1109/TVCG.2018.2810068
[70] Kenneth Weiss, Federico Iuricich, Riccardo Fellegara, and Leila De Floriani. 2013. A primal/dual representation for discrete Morse complexes on tetrahedral meshes. *Comput. Graph. Forum* 32, 3 (2013), 361–370. DOI: https://doi.org/10.1111/cgf.12123
[71] Julie C. Xia and Amitabh Varshney. 1996. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the IEEE Visualization Conference*, 327–334. DOI: https://doi.org/10.1109/visual.1996.568126
[72] Xin Xu, Federico Iuricich, and Leila De Floriani. 2020. A persistence-based approach for individual tree mapping. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*. ACM, 191–194. DOI: https://doi.org/10.1145/3397536.3422231
[73] Bisheng Yang, Wenzhong Shi, and Qingquan Li. 2005. A dynamic method for generating multi-resolution TIN models. *Photogram. Eng. Remote Sens.* 71, 8 (2005), 917–926.
[74] Xianwei Zheng, Hanjiang Xiong, Jianya Gong, and Linwei Yue. 2017. A morphologically preserved multi-resolution TIN surface modeling and visualization method for virtual globes. *ISPRS J. Photogram. Remote Sens.* 129 (2017), 41–54. DOI: https://doi.org/10.1016/j.isprsjprs.2017.04.013

Received 17 May 2023; revised 3 March 2024; accepted 8 March 2024