Efficient topology-aware simplification of large triangulated terrains

Yuting Song  
University of Maryland  
College Park, Maryland, USA  
ytsong@umd.edu

Riccardo Fellegara  
German Aerospace Center (DLR)  
Braunschweig, Germany  
riccardo.fellegara@dlr.de

Federico Iuricich  
Clemson University  
Clemson, South Carolina, USA  
fiurici@clemson.edu

Leila De Floriani  
University of Maryland  
College Park, Maryland, USA  
deflo@umd.edu

ABSTRACT
A common first step in the terrain processing pipeline of large Triangulated Irregular Networks (TINs) is simplifying the TIN to make it manageable for further processing. The major problem with TIN simplification algorithms is that they create or remove critical points in an uncontrolled way. Topology-aware operators have been defined to solve this issue by coarsening a TIN without affecting the topology of its underlying terrain, i.e., without modifying critical simplices describing pits, saddles, peaks, and their connectivity. While effective, existing algorithms are sequential in nature and are not scalable enough to perform well with large terrains on multicore systems. Here, we consider the problem of topology-aware simplification of very large meshes. We define a topology-aware simplification algorithm on a compact and distributed data structure for triangle meshes, namely the Terrain trees. Terrain trees reduce both the memory and time requirements of the simplification procedure by adopting a batched processing strategy of the mesh elements. Furthermore, we define a new parallel topology-aware simplification algorithm that takes advantage of the spatial domain decomposition at the basis of Terrain trees. Scalability and efficiency are experimentally demonstrated on real-world TINs originated from topographic and bathymetric LiDAR data. Our experiments show that topology-aware simplification on Terrain trees uses 40\% less memory and half the time than the same approach implemented on the most compact and efficient connectivity-based data structure for TINs. Beyond that, our parallel algorithm on the Terrain trees reaches a 12x speedup when using 20 threads.

KEYWORDS
terrain simplification, edge contraction, spatial indexes, topological methods, shared memory processing

1 INTRODUCTION

Morse theory is a powerful mathematical framework that allows for segmenting a scalar field according to the regions of influence of its critical points. This general task has revealed fundamental in many application domains, including material science [30], chemistry [41], environmental science [50], forest monitoring [52], and urban analysis [16], to mention a few. In terrain analysis, in particular, the segmentation of a terrain according to its critical points (i.e., peaks and pits) provides information regarding terrain morphology, which are fundamental for assessing the risk of landslides or floods. 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 limited by their large storage costs compared to DEMs. At the same time, the increasing availability of large point clouds [42] intensifies the need for scalable data representations for TINs.

Spurious critical points, naturally affecting noisy data, can severely affect the analysis of a terrain. For this reason, several simplification approaches capable of removing spurious features while maintaining important critical points have been defined in the literature [2, 8, 36, 38]. These approaches reduce the morphological complexity of the dataset, leaving the underlying digital terrain model unchanged. This represents still an issue when working with large terrain datasets since the complexity of extracting, representing, and visualizing topological features and structures is directly related to the resolution of a terrain model. At the same time, reducing the terrain model’s resolution may affect its topology in an uncontrolled way.

In [33], we recently addressed this problem by defining a local simplification operator, called gradient-aware edge contraction, 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, the user is enabled 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 is memory-demanding, and, thus, there is the need to use efficient representations to reduce memory requirements. Second, performing a large number of simplifications sequentially is time-consuming [33]. This latter directly affects 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 [18], which has been shown to be the most compact representation for triangulated terrains (see Section 6). Moreover, the distributed nature of Terrain trees is used to define a new parallel simplification algorithm (see Section 7). Our approach reduces the geometric complexity of a large triangulated terrain without affecting its morphology and without incurring into limitations due to processing time or space constraints. In Section 8, we experimentally evaluate both the sequential and the parallel topology-aware simplification on Terrain trees.

2 BACKGROUND

In this section, we review some fundamental elements of discrete Morse theory, which is the basis for 2D scalar field topology, but just restricting to triangle meshes. The interested reader is referred to [11, 22] for a complete view of the theory, and its application in shape analysis and visualization.

Morse theory [37] is a mathematical tool studying the relationships between the topology of a manifold shape and the critical points of a smooth scalar function $f$ defined over $M$. Based on Morse theory, we can define segmentations of the shape based on the regions of influence and the connectivity of its critical points. Discrete Morse Theory (DMT) [22] is a combinatorial counterpart of 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 [18, 30, 45, 51].

Given a triangle mesh $\Sigma$ and an elevation function $f : \Sigma \to \mathbb{R}$ defined on the vertices of $\Sigma$, a (discrete) vector is defined as a pair of cells of the complex $(\sigma, \tau)$ such that $\sigma$ is a face of $\tau$. A gradient pair can be viewed as an arrow formed by a head ($k$-simplex) and a tail ($k - 1$-simplex). Recall that a $k$-simplex is the convex hull of $k + 1$ points that are affinely independent in $\mathbb{R}$, and $k$ is called the dimension of the simplex. In a triangle mesh, a vertex is a 0-simplex, an edge a 1-simplex, and a triangle a 2-simplex, and we have arrows formed by a triangle and an edge (triangle-edge pair), and by an edge and a vertex (edge-vertex pair). A simplex that is not a head or a tail of any arrow is a critical simplex. In a triangle mesh, there are three types of critical simplices: critical triangles indicating maxima, critical edges indicating saddles, and critical vertices indicating minima. It has been proved that critical simplices appear in correspondence of critical points of the terrain [25].

A discrete vector field $V$ is any collection of vectors such that each cell of $V$ is a component of at most one vector in $V$. A $V$-path is a sequence of vectors $(\sigma_i, \tau_i)$ belonging to $V$, for $i = 0, \ldots, r$, such that, for all indexes $0 \leq i \leq r - 1$, $\sigma_{i+1}$ is a facet of $\tau_i$ and $\sigma_i \neq \sigma_{i+1}$. A $V$-path is said to be closed if $\sigma_0 = \tau_r$, and trivial if $r = 0$. Discrete vector field $V$ is a Forman gradient if all of its closed $V$-paths are trivial. Figure 1(a) shows an example of a discrete gradient computed on a triangle mesh. Red, green, and blue dots indicate critical triangles, edges, and vertices, respectively. Arrows indicate gradient vectors.

For terrain analysis, the Forman gradient can be seen as the combinatorial counterpart of the gradient of the elevation function $f$ [45, 51] and directly allows the computation of different topological structures [11]. Figure 1(b) shows the gradient paths connecting pairs of critical cells (i.e., 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)).

3 RELATED WORK

Topology-aware simplification combines the need for reducing the size of a mesh with the need to maintain its topological properties. The task of mesh simplification has been extensively studied in the literature, and, thus, we refer the reader to comprehensive surveys on the topic [6, 31, 49].

Popular techniques for mesh simplifications include edge collapse [32], vertex decimation [48], and vertex clustering [46]. Edge collapse, also called edge contraction, consists of the contraction of one edge to a single vertex. When the edge is contracted to one of its endpoints, this operation is called half-edge collapse. In this paper, we adopt the half-edge collapse operator as it is the most commonly used one, and does not introduce a new vertex when contracting an edge.

After selecting a simplification operator, one should define the order in which simplifications are performed. When considering edge contraction, several metrics have been defined for optimizing the quality of output meshes, such as minimizing an energy function [32], setting a threshold on the Hausdorff distance between the original and the simplified models [34], minimizing the error quadrics [26], or setting a threshold on the error volume [28]. The
Quadric Error Metric (QEM) [26] is one of the few approaches keeping a good balance between the computational cost and the quality of the mesh produced.

Besides the quality of the output mesh, another issue addressed in the literature is to efficiently simplify large meshes. A common and well-established method is to perform a simplification on the mesh in parallel or on a cluster. One approach to achieve that is to define heuristics to avoid contracting adjacent edges concurrently [27, 35, 40]. A different strategy is partitioning the mesh into submeshes that can be then processed independently [3, 13, 23, 44].

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. In general, simplification operators are not topology-preserving, which means they can modify the number of critical points of terrain, or their connectivity, in an uncontrolled way.

This problem has been first addressed in [1] by introducing a simplification operator which 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 [10], the first efficient topology-preserving operator based on edge contraction has been introduced. The method preserves the critical points connectivity by checking the separatix lines incident at the endpoints of a contracted edge. However, while the operator prevents 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 is introduced in [33]. The 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 (see Section 5 for details), the contraction is guaranteed to preserve the terrain topology. On top of that, a multiresolution model, called a Hierarchical Forman Triangulation (HFT), is introduced. This model combines geometric and topological operators to enable the mesh simplification or refinement by varying the resolution of both topology and geometry on-demand.

Starting from the gradient-aware simplification operators, the original criteria have been relaxed in [15] to allow the removal of critical simplices. While this operator does not lead to a multiresolution model like HFT, it increases the number of admissible edge contractions, and thus, the compression factor. Recently, a new approach based on vertex removal has been proposed in [24]. The simplification operator is similar to the one introduced in [1], but in this case the topology is preserved by checking a descriptor which definition is rooted in algebraic topology, i.e., persistent homology [17]. A vertex removal is valid only if the link of the removed vertex can be re-triangulated preserving persistent homology.

In this work, we use the gradient-aware edge contraction from [33]. Differently from the vertex removal [24], this operator comes with several metrics for controlling and optimizing the quality of the output meshes [26, 28, 32, 34]. Also, any simplification sequence performed with this operator supports the construction of a HFT multiresolution model which is fundamental if we want to enable a user to explore a dataset interactively in real time.

4 TERRAIN TREES

In this section, we briefly introduce the data structure used in our work. We refer the reader to the original paper for more details [18]. 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 [12]. Within this class of data structures we can find the Indexed data structure with Adjacencies (IA data structure) [43], the Corner Table (CoT) [47] and the Sorted Opposite Table (SOT) [29]. Recently, a new compact family of spatial data structures designed for triangulated terrain meshes, called Terrain Trees, has been developed [18]. Based on Terrain trees, we have developed the Terrain Tree library (TTL), 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 [18]. In our experiments, the PR Terrain tree has been shown to be slightly more compact and efficient than the other two. So, we will use here the PR Terrain tree, that we will just call Terrain tree, for the sake of simplicity.

A Terrain tree on a triangle mesh $\Sigma$ consists of: (1) a global vertex array $\Sigma_V$, encoding, for each vertex, its coordinates and elevation, (2) a global triangle array $\Sigma_T$, encoding, for each triangle, a triplet of vertex indices in the global vertex array, (3) a bucketed 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 $D$, in which each leaf block $b$ contains the vertices of the mesh that fall in $b$ plus the triangles that intersect $b$.

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 [21]. 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, with about 36% less storage than the most compact state-of-the-art mesh data structure (the IA data structure), while maintaining good performances in extracting connectivity relations. Moreover, 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. 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 [32]. Given an edge $e = \{v_1, v_2\}$ in a triangle mesh $\Sigma$, it contracts $e$ to one of its endpoints, and removes from $\Sigma$ edge $e$, one of its endpoints, and the triangles incident in $e$. An edge contraction operator can modify the shape of the TIN creating non-valid meshes. To prevent this, we verify that contracting an edge $e$
 satisfies two fundamental validity conditions, namely the link and the fold conditions.

The link condition [14] 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 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 2 shows an example of an edge contraction that does not satisfy the link condition. If edge \( e \) is contracted, the resulting mesh is invalid since more than two triangles would be incident in the same edge (i.e., edge \( \{v_f, v_g\} \)).

The fold condition [5, 26] ensures that, for every edge \( e' \in Lk(v_2) \), \( v_1 \) and \( v_2 \) lie on the same side of the line \( l \) extended from \( e' \). If this condition is not verified, then \( \Sigma \) will have at least a triangle folding over itself after contracting \( e \) to \( v_1 \).

Our purpose is to preserve the topological features of the scalar field defined on \( \Sigma \), while simplifying the underlying mesh. This translates into maintaining the Forman gradient, and thus the critical simplices. To this aim, we apply the gradient-aware condition [33]. Given a mesh \( \Sigma \) 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 \( (v_2, e, \text{and two triangles incident in } e) \) are not critical; and (2) either \( v_1 \) or \( v_2 \) is paired with \( v \) in \( V \).

A gradient-aware edge contraction requires, in addition to the modification of the mesh, also the update of the 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 \( v_1 \) to \( v_2 \). We denote the other two triangles adjacent to \( t_1 \) as \( t_3 \) and \( t_4 \) and the vertex opposite to \( e \) in \( t_3 \) as \( v_3 \). The updates on the left part need to ensure that vertices \( v_1, v_2, e \) \( \{v_1, v_2\} \), and triangles \( t_3 \) and \( t_4 \) are still paired after simplification. We know that edge \( e \) is paired with either \( v_1 \) or \( v_2 \). Thus, if \( e \) is paired with \( v_2 \), after the contraction, the pairing of \( v_1 \) does not change; otherwise, \( v_1 \) will be paired with the simplex previously incident to \( v_2 \).

Now 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 3(a)), or with another triangle, \( t_5 \) (see Figure 3(b)). In both cases, \( t_3 \) and \( t_5 \) are paired with the same simplexes 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 3(a)), or with another triangle \( t_5 \) (see Figure 3(b)). The same reasoning applies when \( t_1 \) was paired with \( \{v_2, v_3\} \). The same update strategy is applied to the simplices on the right of the edge \( e \) to maintain the topology of the discrete gradient [33].

6 TOPOLOGY-AWARE SIMPLIFICATION ON TERRAIN TREES

In this section, we describe a new topology-aware simplification algorithm we developed on a Terrain tree \( T \) to simplify a triangle mesh \( \Sigma \). To preserve the topology of the scalar field (elevation for terrains), the algorithm uses a Forman gradient \( V \) computed on \( \Sigma \) inside the Terrain tree and encoded as a bit vector using the same indexing of \( \Sigma_T \), resulting in a cost of one byte per triangle [31]. As an error metric for edge contraction we use the Quadratic Error Metric (QEM) [26]. For brevity, we describe in Appendix A.1 how to compute the initial error quadrics associated with each vertex \( v \), which represent a set of planes incident in \( v \).

All leaf blocks in Terrain tree \( T \) are visited through a depth-first traversal. Algorithm 1 provides a pseudo-code description of the simplification procedure within a leaf block \( b \). 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 edge \( e \) 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 \( \omega \). Edge \( e \) is an internal edge for \( b \) if also the other vertex of \( e \) is in \( b \), otherwise \( e \) is a cross edge.

For each leaf block \( b \), the algorithm performs the following steps:

1. Extract the Vertex-Triangle (VT) relations for the vertices in \( b \) (row 1): the Vertex-Triangle (VT) relation for a vertex \( v \) in \( b \) is defined as the set of triangles incident in \( v \).
2. Build a priority queue \( Q \) of candidate edges (row 3): the edges in the queue are ordered by their cost.
Algorithm 1 LEAF_SIMPLIFICATION(b, Σ, V, E, ω, C, bR)

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
       bR: root block of the hierarchy

// Extract the local VT relations for the vertices in b
1: local_vts ← local_vt(b, Σ)
// Create an array for encoding the updated edges costs
2: updated_edges ← []
// Create a priority queue of candidate edges
3: Q ← CANDIDATE_EDGES(b, Σ, E, ω)

while Q ≠ ∅ do
5: e ← DEQUEUE(Q)  // e = {v1, v2}
   // Check if e has been updated and if its cost is updated
6: if e ∈ updated_edges and not SAME_COST(e, updated_edges) then
7:     skip e  // If its cost is not updated, then skip this edge
9: end if
10: VT(v1) ← get_vt(v1, local_vts, bR, Σ)
11: VT(v2) ← get_vt(v2, local_vts, bR, Σ)
12: ET(e) ← GET_ET(e, VT(v1))
   // Check three conditions introduced in Section 5 for e
13: if LINK_CONDITION(e, VT(v1), VT(v2), ET(e))
14:   and FOLD_CONDITION(e, VT(v2), ET(e))
15:   and GRADIENT_CONDITION(e, VT(v2), ET(e), V)
16: then
17:   CONTRACT(e, VT(v1), VT(v2), E, Σ)
18:   UPDATE_GRADIENT(e, VT(v1), VT(v2), ET(e), V)
19:   UPDATE_INDEX(e, VT(v2), b, bg)
   // Update the VT relation of v1
20: VT(v1) ← VT(v1) ∪ VT(v2) - ET(e)
   // Update the cost of edges, and add these edges to Q
21: updated_edges ← UPDATE_COSTS(v1, VT(v1), E, Q)
22: end if
23: end while
24: C ← C ∪ local_vts  // Add local_vts to the LRU-cache

(3) Simplify candidate edges (rows 4-23): for each candidate edge e, the three validity conditions (introduced in Section 5) are checked. If these conditions are satisfied, edge e is contracted, and the Forman gradient updated together with the Terrain tree. This step is described in details below.

The link, fold and gradient-aware conditions are checked for each edge e = {v1, v2} extracted from Q (row 5). To check these conditions we need the VT relations for v1 and v2 and the Edge-Triangle (ET) relation of e (rows 10-12). ET(e) consists of the two triangles sharing edge e. If e is an internal edge, VT(v1) and VT(v2) in function get_vt are encoded in array local_vts. Conversely, if e is a cross edge, and v1 is contained by another leaf block b1, get_vt must extract the VT relations of b1. 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 v1 is in block b1, get_vt looks first if the VT relations of block b1 are in C. If such relations are not in C, then we extract them and save them in C.

The ET relation of a candidate edge e is extracted by traversing VT(v1) and finding triangles incident to e (GET_ET procedure at row 12). To check the link condition (row 13), the set of vertices adjacent to v1 or v2 are extracted on-the-fly in LINK_CONDITION by traversing the VT relations of v1 and v2.

If e satisfies all three conditions, then it is contracted to its optimized position (i.e., v1) by function CONTRACT (row 17). This procedure takes as input edge e, the VT relation of v2, the ET relation of e, the array of vertex error quadrics E, and TIN Σ. It removes vertex v2 as well as the two triangles adjacent to e. In each remaining triangle in VT(v2), it replaces v2 with v1. After the contraction, the error quadric of v1 is updated by adding the quadric of v2 to it. The pseudo-code of function CONTRACT is in Algorithm 2 (see Appendix A.2).

After the contraction, both the Forman gradient V and the Terrain tree T are updated (rows 18-20). The update of gradient V (UPDATE_GRADIENT procedure at row 18) follows the method introduced in Section 5. This involves up to four triangles adjacent to triangles in ET(e) (see Figure 3 for an example). Such triangles are retrieved by using the corresponding VT and ET relations.

The update of T is performed by function UPDATE_INDEX (row 19). If e is an internal edge, the current leaf block b is updated by removing the index of v2 and the indexes of the triangles incident in e. If e is a cross edge, and v1 is indexed in leaf block b1, then both b and b1 are updated in a similar way. In this latter case, the indexes of those triangles that were incident in v2 but not encoded in b1 are also added to b1. The VT relation of vertex v1 is updated by adding the triangles in the VT relation of v2 and by removing the triangles adjacent to e (row 20).

Since the error quadric of v1 is updated after the contraction of e, the costs of all edges currently incident in v1 need to be updated accordingly (row 21). A local auxiliary array updated_edges which is initialized in row 4, is used to keep track of the updated edge costs. All updated edges are added to Q again. Note that we do not update the costs of edges in Q directly, and, therefore, each time we process an edge e from Q, we check if e has been updated in previous contractions (row 6). If e has been updated and the cost stored with e is not the one stored in updated_edges then we discard e and process the next edge in Q.

Finally, after the simplification of leaf block b, the local_vts array is inserted to C (row 24).

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. 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. Thus, the key idea behind our parallel simplification strategy is to assign each leaf block to a single thread from a set of available threads. The main challenge here is to prevent conflicts that may occur if two threads modify the same vertex, or the same triangle concurrently.
We need to ensure that: (1) the check on the vertex being removed is in the 2-neighborhood of \( v \) and (2) the update of \( v \) is updated by a thread. A conflict block \( b \) can be modified during the contraction of edge \( e \) only if it is adjacent to a triangle incident in \( e \), i.e., adjacent to a triangle to be removed during the contraction of \( e \).

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 that:

**Proposition 7.2.** The gradient pairing information associated with triangle \( t \) can be modified during the contraction of edge \( e \) only if \( t \) is adjacent to a triangle incident in \( e \), i.e., adjacent to a triangle to be removed during the contraction of \( e \).

The validation of the gradient condition involves only the triangles in \( V T(\psi_2) \). It is straightforward to prove that if another edge contraction operation is modifying the gradient pairing information of a triangle in \( V T(\psi_2) \), then the vertex to be removed by that operation is in the 2-neighborhood of \( \psi_2 \) and, thus, breaks the leaf locking strategy condition.

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 is modified by two threads \( T \psi_1 \) and \( T \psi_2 \) at the same time and edge \( e \) is being removed by \( T \psi_1 \). From Proposition 7.2, we know that \( t_3 \) should be adjacent to two triangles being removed by \( T \psi_1 \) and \( T \psi_2 \), respectively. Without loss of generality, we assume that \( t_3 \) (purple triangle in Figure 5) is a triangle to be removed by \( T \psi_2 \). Then, either \( \{v_4, v_5\} \) or \( \{v_5, v_3\} \) is the edge to be removed by \( T \psi_2 \). In both cases, the vertex to be removed is in the 2-neighborhood of \( \psi_2 \), which violates Proposition 7.1.

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 \( C I(b) \), 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 \( b \) containing \( v \), and add \( b \) to \( C I(b) \).

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 list of conflict blocks which changes due to the ongoing 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
Table 1: Overview of experimental datasets. For each terrain, we list the number of vertices $|V_T|$ and triangles $|T_T|$.

| terrain         | $|V_T|$ | $|T_T|$ |
|-----------------|--------|--------|
| Molokai         | 25M    | 34M    |
| Great Smokey Mts| 50M    | 68M    |
| Canyon Lake     | 49M    | 98M    |
| Yosemite Rim Fire| 78M  | 155M   |
| Dragons Back Ridge | 91M | 182M   |
| Moscow Mountain | 113M   | 226M   |

Great Smokey Mountains, Canyon Lake, Yosemite Rim Fire, Dragons Back Ridge, and Moscow Mountain, are topographic LiDAR point clouds from the OpenTopography repository [42].

The generation of the Terrain tree we use in this paper relies on a single parameter which defines the maximum number of vertices allowed in each leaf block of decomposition, i.e. the block capacity. To select the block capacity for each dataset in connection with the mesh simplification task, we establish an initial range for capacity values between 1/10000 and 1/30000 of the total number of vertices in the data set, this is in order to have coarser hierarchical subdivisions, usually beneficial for tasks requiring intense navigation of the hierarchy. Within this range, we have selected ten different capacity values for each dataset, and we have compared the performance in sequentially simplifying the meshes encoded by resulting Terrain tree. Our comparisons have shown that the memory footprint and the compression rate, defined as the ratio between the number of vertices in the simplified mesh and in the original one, 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. Further details of this experimental evaluation can be found in Appendix A.4. In the following, for each dataset, we use the capacity value showing the best trade-off between simplification time and memory requirements.

8 Experimental Results

In this section, we evaluate the performances of both the sequential and parallel topology-aware terrain mesh simplification algorithms on the Terrain tree. In subsection 8.1, we compare the performance of the sequential topology-aware simplification on the Terrain trees against our implementation of the most compact triangle-based data structure for meshes, the Indexed data structure with Adjacencies (IA data structure) [43]. In subsection 8.2, we compare the sequential and parallel simplification strategies implemented on the Terrain trees. The source code of the simplification algorithm based on Terrain trees is available at [19].

All the experiments are performed on a dual Intel Xeon E5-2630 v4 @2.20Ghz CPU (20 cores in total), and 64GB of RAM. A total of six TINs, generated from raw point clouds using the CGAL library [4], 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 [39].
the two approaches analyzing the size of the output TIN when varying the quality control parameter. Given a user-defined threshold \( \omega \), we simplify all contractible edges with cost lower than \( \omega \). Based on the initial error quadrics, we compute the costs for all edges in \( \Sigma \), and use the quartile values to set three different thresholds.

Table 2 shows the results obtained. Using a global queue leads to TINs that are about 1% smaller than the one obtained by the Terrain tree. On the other hand, the simplification on the Terrain tree is always faster. When using a larger \( \omega \), the Terrain tree is at least twice as fast as the IA data structure. Also, when \( \omega \) increases, the memory requirements on the IA data structure also increase, while on the Terrain tree they remain stable. While using local queues show a slightly lower compression rate with respect to using a global queue, timings and memory requirements are dramatically in favor of the Terrain tree approach. This 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. Results include timings required for computing initial error quadrics and performing the topology-aware simplification, the memory footprint required by the simplification, and the compression rate. On average, Terrain trees simplify 0.5% less edges than the IA data structure, while they use from 45% to 56% less time than the IA data structure. As shown in Table 3, 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 experiment datasets can be simplified using the IA data structure.

### 8.2 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 when using from 1 to 64 threads.

Figure 6 shows the speedup achieved by the parallel simplification algorithm when increasing the number of threads. The approach scales well as long as the number of threads is lower than the number of physical cores (20 in our case). The speedup still slightly increases when using more than 20 threads, but it decreases with more than 40.

The efficiency of the parallel algorithm can be computed as \( E = T_1 / N T_N \), where \( T_N \) is the time for the parallel algorithm using \( N \) threads, \( T_1 \) is the parallel algorithm using a single thread. Figure 7 shows the results. Note that efficiency decreases with the increasing of the number of threads. This is common for parallel algorithms due to possible load imbalance and overheads during the computation. When using 20 threads, the efficiency of the parallel simplification is 67% on all experimental datasets. With more than 20 threads the efficiency decreases faster. Considering these results, we observe that the best trade-off is achieved when the thread number is equal to the number of available cores.

Finally, we compare the parallel and sequential mesh simplification algorithms using the same Terrain tree, and 20 threads (see Table 3). The parallel simplification strategy provides a 12x speedup compared to the sequential strategy, while still reaching the same compression rate. Also, even if the parallel strategy processes multiple leaf blocks at the same time, its memory footprint remains stable since, on average, it uses only 1% more memory than that the sequential algorithm. These results show the scalability and efficiency of the Terrain tree representation also when using shared-memory processing techniques.

### 9 CONCLUDING REMARKS

We have presented a new method for simplifying very large triangle meshes representing terrains on a compact data structure, the Terrain tree. A Terrain tree [18] has been shown to be the most compact data structure for triangulated terrains, which combines a minimal connectivity-based encoding of the triangle mesh with a spatial index as a clustering mechanism that enables an implicit encoding of other connectivity relations.

The simplification method we presented extends the strategy defined in [33] on a global topological data structure, which is
Table 2: Time T (in minutes), peak memory usage M (in Gigabytes), and reduction rate R (in %) of topology-aware mesh simplification on the IA data structure and the Terrain tree (TT) when using different cost threshold $\omega$. Q1, Q2, and Q3 represent the first, the second, and the third quartile edge costs of each dataset, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Molokai</th>
<th>Great Smokey Mountain</th>
<th>Canyon Lake</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega$</td>
<td>IA TT</td>
<td>IA TT</td>
<td>IA TT</td>
</tr>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
</tr>
<tr>
<td>$</td>
<td>\Sigma</td>
<td>_T$</td>
<td>21.5</td>
</tr>
<tr>
<td></td>
<td>42.9</td>
<td>42.9</td>
<td>35.4</td>
</tr>
<tr>
<td>$M$</td>
<td>15.6</td>
<td>10.5</td>
<td>21.6</td>
</tr>
<tr>
<td></td>
<td>17.5</td>
<td>12.7</td>
<td>18.7</td>
</tr>
<tr>
<td>$R$</td>
<td>80.5</td>
<td>80.5</td>
<td>99.8</td>
</tr>
</tbody>
</table>

Table 3: Time T (in minutes), peak memory usage M (in Gigabytes), and reduction rate R (in %) of topology-aware mesh simplification on the IA data structure and on sequential (seq.) and parallel (para.) version on Terrain trees.

<table>
<thead>
<tr>
<th></th>
<th>Molokai</th>
<th>Great Smokey Mountain</th>
<th>Canyon Lake</th>
<th>Yosemite Rim</th>
<th>Dragons Back Ridge</th>
<th>Moscow Mountain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IA TT</td>
<td>IA TT</td>
<td>IA TT</td>
<td>IA TT</td>
<td>IA TT</td>
<td>IA TT</td>
</tr>
<tr>
<td>$T$</td>
<td>58.0</td>
<td>29.0</td>
<td>23.8</td>
<td>71.0</td>
<td>39.1</td>
<td>3.31</td>
</tr>
<tr>
<td>$M$</td>
<td>21.3</td>
<td>12.7</td>
<td>12.8</td>
<td>29.0</td>
<td>17.2</td>
<td>17.4</td>
</tr>
<tr>
<td>$R$</td>
<td>73.8</td>
<td>73.3</td>
<td>73.4</td>
<td>75.4</td>
<td>75.0</td>
<td>75.0</td>
</tr>
</tbody>
</table>

based on a local simplification operator, called gradient-aware edge contraction, capable of reducing the resolution of a TIN while preserving the topology of the underlying terrain. This operator, paired with a topological simplification operator, reduces the resolution of both the topology and the geometry of a terrain in a completely controlled way. Also, thanks to the distributed nature of Terrain trees, we defined a parallel version of this simplification method. The parallel strategy is based on a leaf locking strategy, that prevents conflicts occurring when multiple threads try to update the same vertex or the same triangle concurrently.

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, and the most compact at the state of the art, Terrain trees require half the time and 40% of the memory while still reaching similar simplification levels. These results prove the scalability and efficiency of our method at processing large-scale triangle meshes. Also, when comparing the sequential and parallel strategies based on Terrain trees, we noticed a further performance increase. Thanks to the local and distributed nature of Terrain trees, the parallel strategy achieves a 12x speedup when using 20 threads while having similar memory requirements.

The parallel strategy developed here can be easily extended to other topology-aware edge contraction operators, such as the one introduced in [15], since the range of simplices involved in those topology-preserving conditions is the same as for our gradient-aware simplification operator. Although the gradient-aware contraction operator is efficient and produces good-quality meshes, some applications require maintaining the Delaunay property while simplifying the mesh. We plan to investigate how to preserve Delaunay properties in the simplification process, also in connection to coastal ocean modeling for storm surge and tide simulation.

Our current parallel strategy is compatible with a shared-memory processing based on OpenMP [9]. To increase its efficiency, we plan to use specialized compilers, like ISPC1, and libraries, like TBB2. We also plan to extend the simplification algorithm to support a distributed-memory processing strategy based on MPI [7].

ACKNOWLEDGMENTS

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-2467209. 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. The Molokai point cloud is kindly provided by NOAA National Centers for Environmental Information.

REFERENCES


---

1https://ispc.github.io/

A Appendix

A.1 Computing the quadric error matrix on Terrain trees

In the Quadric Error Metric (QEM) [26], the error at one vertex \( v \) of a triangle mesh \( \Sigma \) is defined as the sum of the squared distances to the planes of the triangles incident in \( v \). 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 \( 4 \times 4 \) matrix called fundamental error quadric. The overall error at \( v \) can be represented as \( \Delta(v) = v^T Q v \). \( Q_v \) is called initial error quadric \( v \), and 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 \( \Delta(e) = v^T (Q_1 + Q_2) v \), 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 a traversal of its triangle list. For each triangle \( t \) in \( b \):

1. check if at least one vertex of \( t \) contained in \( b \). If not, skip \( t \), otherwise perform 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 \( b \), 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 \( \Sigma_T \) of \( \Sigma \) and calculate the corresponding fundamental error quadrics. On the other hand, the computation of the initial independent error quadrics at the vertices in different leaf blocks are completely independent and fully local to each leaf block. This is optimal for computing the quadric error matrices of \( \Sigma \) in parallel.

A.2 Performing an edge contraction

In this section, we describe how an edge contraction is performed. Algorithm 2 depicts the edge contraction operation at row 17 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 Correctness proof of the parallel simplification algorithm

In this section, we provide the correctness proof of Proposition 7.1 and an explanation of why the possible edge cost conflicts cannot affect the parallel simplification.

We recall Proposition 7.1:

**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.

---

Algorithm 2 Contract\((e, VT(v_j), ET(e), E, \Sigma)\)

**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. \( \Sigma \leftarrow \Sigma \setminus \{t\} \) \quad // Remove \( t \) from \( \Sigma \)

3. **end for**

4. **// For each triangle** \( t \) **incident in** \( v_j \) **but not adjacent to** \( e \)
5. **for each** \( i \) in \( (VT(v_j) \setminus ET(e)) \) **do**
6. \( t \leftarrow (t \setminus v_j) \cup v_i \) \quad // Replace \( v_j \) with \( v_i \) in \( t \)
7. **end for**

8. \( E[i] \leftarrow E[i] + E[j] \) \quad // Update the error quadric at vertex \( v_i \)
9. \( \Sigma \leftarrow \Sigma \setminus \{v_j\} \) \quad // Remove \( v_j \) from \( \Sigma \)

---

**Figure 8:** An example of a vertex \( v \) and two vertices in its 2-neighborhood.

Consider a vertex \( v \) in leaf block \( b_1 \) 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 8) or has a sharing adjacent vertex with \( v \) (e.g., \( v_2 \) in Figure 8). We first consider the case that a vertex \( v_2 \) shares an adjacent vertex \( v_1 \) with \( v \). There are 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 of them are internal edges, (2) one of them is an internal edge, the other one is a cross edge, (3) both of them 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 \( v_1 \), while in case (3), \( v_2 \) belongs to a block \( b_2 \) which shares a conflict block \( b_1 \) with \( v_1 \) as shown in Figure 8. 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 it is not active.

Similarly, when \( v' \) is adjacent to \( v \), \( v' \) is either in \( b \) or in a conflict block of \( b \). In both cases, \( v_2 \) cannot be considered in an edge contraction operation.

In section 7, we 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 \( v_2 \)
Table 4: Time (in minutes) (denoted as T), peak memory usage (in Gigabytes) (denoted as M), and reduction rate (in %) (denoted as R) of simplification when using different capacity values (denoted as C) for the subdivision of Terrain tree.

<table>
<thead>
<tr>
<th>Molokai</th>
<th>Great Smokey Mountains</th>
<th>Canyon Lake</th>
<th>Yosemite Rim Fire</th>
<th>Dragons Back Ridge</th>
<th>Moscow Mountain</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>T</td>
<td>M</td>
<td>R</td>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>300</td>
<td>36.3</td>
<td>12.8</td>
<td>73.1</td>
<td>300</td>
<td>44.8</td>
</tr>
<tr>
<td>400</td>
<td>35.1</td>
<td>12.8</td>
<td>73.2</td>
<td>400</td>
<td>44.8</td>
</tr>
<tr>
<td>500</td>
<td>29.5</td>
<td>12.7</td>
<td>73.3</td>
<td>500</td>
<td>39.4</td>
</tr>
<tr>
<td>600</td>
<td>29.0</td>
<td>12.8</td>
<td>73.3</td>
<td>600</td>
<td>41.9</td>
</tr>
<tr>
<td>700</td>
<td>29.0</td>
<td>12.7</td>
<td>73.3</td>
<td>700</td>
<td>39.8</td>
</tr>
<tr>
<td>800</td>
<td>29.5</td>
<td>12.7</td>
<td>73.4</td>
<td>800</td>
<td>39.1</td>
</tr>
<tr>
<td>900</td>
<td>29.1</td>
<td>12.7</td>
<td>73.4</td>
<td>900</td>
<td>39.4</td>
</tr>
<tr>
<td>1000</td>
<td>29.2</td>
<td>12.7</td>
<td>73.4</td>
<td>1000</td>
<td>39.4</td>
</tr>
<tr>
<td>1100</td>
<td>29.2</td>
<td>12.8</td>
<td>73.4</td>
<td>1100</td>
<td>39.6</td>
</tr>
<tr>
<td>1200</td>
<td>29.8</td>
<td>12.7</td>
<td>73.4</td>
<td>1200</td>
<td>39.5</td>
</tr>
</tbody>
</table>

Figure 9: (a) shows an 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) shows a case that blocks encoding \( v_1 \) and \( v_4 \) can be active at the same time under the leaf locking strategy, and (c) shows an invalid case in which \( v_1 \) and \( v_4 \) cannot be removed at the same time.

and another edge \( e_2 = \{v_3, v_4\} \) being contracted to \( v_3 \) on \( \Sigma \). If \( v_2 \) and \( v_3 \) are connected by an edge \( e_0 \), 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 9(b)). Otherwise, like the example in Figure 9(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 these blocks. Therefore although it is possible that the cost of an edge is updated by different threads, such edge is not a candidate edge of current active blocks and will not affect the simplification of those blocks.

A.4 Experiments on leaf capacity selection

In this section we present the results in which we evaluate the performance of simplification algorithm when using different capacity thresholds on Terrain trees. Recall that a capacity defines the maximum number of vertices that each leaf block can contain.

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 suboptimal capacity for generating a Terrain Tree, performances are not severely affected and the algorithm still performs well. In the paper, we keep only one capacity value for each dataset to use in the experiments. In general, we use the capacity value that leads to the shortest simplification time, but when the variation in time is small (less than 1%), we consider also the memory cost and the reduction rate. The capacity value selected for each dataset is denoted in Table 4 in bold face.