Noname manuscript No.
(will be inserted by the editor)

Terrain trees: a framework for representing, analyzing and visualizing triangulated terrains

Riccardo Fellegara · **Federico Iuricich** ·
Yunting Song · **Leila De Floriani**

the date of receipt and acceptance should be inserted later

Abstract We propose a family of spatial data structures for the representation and processing of Triangulated Irregular Networks (TINs). We call such data structures *Terrain trees*. A Terrain tree combines a minimal encoding of the connectivity of the TIN with a hierarchical spatial index. Connectivity relations are extracted locally at run-time, within each leaf block of the hierarchy, based on specific application needs. Spatial queries are performed by exploring the hierarchical data structure. We present a new framework for terrain analysis based on Terrain trees. The framework, implemented in the Terrain trees library (TTL), contains algorithms for morphological features extraction, such as roughness and curvature, and for topology-based analysis of terrains. Moreover, it includes a technique for multivariate visualization, which enables the analysis of multiple scalar fields defined on the same terrain. To prove the effectiveness and scalability of such framework, we have compared the different Terrain trees against each other and also against the most compact state-of-the-art data structure for TINs. Comparisons are performed on storage and generation costs and on the efficiency in performing terrain analysis operations.

Keywords terrain modeling, Triangulated Irregular Networks (TINs), spatial indexes, terrain analysis, multivariate visualization

1 Introduction

Thanks to recent development of remote sensing technologies such as Light Detection and Ranging (LiDAR), the amount of available spatial data, represented as raw massive point clouds, has been increasing exponentially. LiDAR data is used in a variety of different fields,

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

Federico Iuricich
Clemson University, Clemson, SC, USA E-mail: fiurici@clemson.edu

Yunting Song
University of Maryland, College Park, MD, USA E-mail: ytsong@umd.edu

Leila De Floriani
University of Maryland, College Park, MD, USA E-mail: deflo@umiacs.umd.edu

including urban modeling [69], climate study [3], earthquake analysis [47], disaster management [75,43], flood risk mapping [71], forest analysis [37,12,74], and coastal morphology analysis [73,65]. Surface models based on LiDAR data enable the extraction of features relevant for several applications. As an example, the slope of a terrain is used to map seafloor habitats [42], while pairing slope, roughness, and curvature of a terrain is used to model coral distribution [17]. Also, the segmentation of a terrain according to its critical points (i.e., peaks and pits) provides information about terrain morphology, which are fundamental for assessing the risk of landslides or floods. Research in these fields has been greatly enhanced by the increasing availability of open data repositories.

A raster representation is often used for modeling a terrain or a surface from airborne LiDAR data, mainly because of the availability of a large number of software tools for processing raster data in Geographic Information Systems (GISs) and remote sensing. While it is always possible to transform a point cloud into a raster representation, this is a computationally intensive operation, which can account for 70-80% of the time required by the total analysis pipeline [6,1]. Also, artifacts might be created due to the resolution of the raster grid, especially if the original point cloud contains noise due to acquisition errors, or if it contains missing data, due to the presence of occlusions.

Adaptive alternatives to raster representations are *Triangulated Irregular Networks (TINs)*, which are used for encoding irregular distributed data at the cost of a higher memory consumption with respect to raster-based elevation models (usually called *Digital Elevation Models (DEMs)*). But compact and widely-used data structures for encoding TINs suffer from scalability issues. For instance, using the most compact state-of-art data structure for triangle meshes, we can process meshes up to 150 million of vertices (300 million of triangles) on our workstation with an Intel Xeon E5-2630 v4 CPU at 2.20Ghz and 64GB of RAM, which is much lower than the size of currently available data sets (see, for instance, the *OpenTopography* repository[1]).

We propose here a new data structure for in-memory processing of TINs, the *Terrain tree*, which encodes the topology of a triangulated terrain combined with a spatial index built on the triangle mesh. We present, discuss and compare three spatial indexes which together form the Terrain tree family. By encoding the vertices incident in each triangle as well as the field values associated with each vertex, we provide the minimum amount of information required for extracting the full mesh connectivity and for processing such fields locally. On the other hand, the spatial index provides the ability to navigate the triangle mesh at a global scale. These two components enable the efficient extraction of connectivity information and guarantee compactness and scalability. Our work builds on a short paper [23] in which we have introduced a single spatial index for triangle meshes, using point-based decomposition of the domain, and we discuss the extraction of some basic morphological terrain features, as slope, curvature and critical points identification. In this work, we have developed a new distributed approach for extracting such morphological information on all Terrain trees. Moreover, we have developed distributed algorithms for computing and analyzing the topology of a terrain by using discrete Morse theory [30]. Such algorithms require an efficient navigation of a triangle mesh, which is a challenging task in hierarchical and modular mesh data structures like the Terrain trees.

Morse theory [52] has been used for computing segmentations of the graph of 2D and 3D scalar fields based on the critical points of the field, like the critical net, which consists of the critical points and of the separatrix lines connecting them, or Morse decompositions, defined by the regions of influence of the critical points. Due to the discrete nature of the

[1] `https://opentopography.org/`

problem, recent research has focused on discrete Morse theory [29] which is a combinatorial counterpart of smooth Morse theory [52]. Discrete Morse theory is based on the definition of a discrete vector field, also called a *Forman gradient*, which emulates the gradient of the original function. By means of the Forman gradient, the connectivity of the critical points and the Morse decomposition can be extracted, thus providing an efficient and compact representation of the terrain topology [15,38].

It is common in applications to study a terrain in combination with additional fields defined on it, and we can find such data in several domains, like forest management, weather predictions and geological sciences [40]. On geological data, such fields usually represent gravity and intensity of the magnetic field. In the case of LiDAR point clouds, these fields can be either: (i) generated by the instrument used for creating the point cloud (e.g., the elevation of each point), (ii) directly derived from the raw data (e.g., the slope or the airborne laser scanning cover), or (iii) computed by a domain expert at run-time (e.g., estimating the curvature and roughness values from the elevation of each point). These types of data sets are referred to as *multifield datasets*, and they are formally represented by triangle meshes and collections of scalar values at the vertices of the mesh. To analyze such multifield datasets, we have extended the strategy defined by Nagaraj et al. [53] for computing a new scalar function capturing the relationships among the multifield data.

The remainder of the paper is organized as follows. In Section 2, we review some background notions on triangle meshes and on discrete Morse theory. In Section 3, we discuss related work on connectivity-based data structures for triangle meshes, on spatial indexes for triangle meshes and maps, and on techniques from topological data analysis relevant to our work. In Section 4, we present the Terrain trees, describing the different subdivision rules and how they are generated, and, in Section 5, their encoding structure and how to execute basic spatial and topological queries on them. In Section 6, we describe how to extract classical morphological terrain features, namely *triangle* and *edge slope*, *curvature* and *roughness* in the Terrain trees framework. In Section 7, we define how to perform topology-based terrain analysis on Terrain trees by using a discrete Morse gradient. First, we introduce a distributed algorithm for extracting a discrete gradient field, and, then, a distributed procedure for extracting the critical net on the critical points of the gradient. In Section 8, we depict how multivariate visualization is performed in Terrain trees. In Section 9, we provide an experimental evaluation of the Terrain trees by comparing the performances of the different spatial indexes also against a state-of-the-art compact data structure for meshes. Finally, in Section 10, we draw some concluding remarks and discuss directions for future work.

2 Background Notions

In this Section, we review some background notions on triangle meshes, which are at the basis of triangulated terrains, and on Morse and discrete Morse theories, which are at the basis of the terrain analysis developed in this work.

2.1 Triangulated Irregular Networks (TINs)

A *Triangulated Irregular Network (TIN)* is a digital terrain model defined by a finite set of irregularly distributed points in the plane, each of which has an elevation value associated. A *TIN* consists of a triangle mesh connecting the points in the plane and of a piecewise linear interpolating function defined on the triangles of such mesh.

To define a triangle mesh, we need to introduce the concept of a simplex. A *k-simplex* σ is the convex hull of $k+1$ independent points in the Euclidean space $\mathbb{E}^n$ (with $k \geq 0$). k is the *dimension* of σ. A 0-simplex is a vertex, a 1-simplex is an edge and a 2-simplex is a triangle. An *h-facet* σ' of a k-simplex σ is an h-simplex ($0 \leq h < k$) generated by $h+1$ vertices of σ. For instance, a triangle has three 0-facets, its vertices, and three 1-facets, its edges. The set of all the facets of a simplex defines its *boundary*. Conversely, the *star* of a simplex σ is the set of simplices that have σ as a facet. For instance, the star of a vertex is the set of triangles and edges incident in it. The *link* of a simplex σ is the set of all the facets of simplices in the star of σ that are not incident in σ. A *triangle mesh* Σ is a collection of vertices (0-simplices), edges (1-simplices), and triangles (2-simplices) such that, given any two triangles in Σ, either they have an empty intersection or they intersect at a common simplex (edge or vertex).

The incidences and adjacencies among the simplices of a triangle mesh are captured by *connectivity relations* [16]. We distinguish among *boundary relations*, which relate a simplex to its facets, *co-boundary relations*, which relate a simplex to the simplices for which it is a facet, and *adjacency relations*, which relate simplices sharing a facet. For instance, the *Triangle-Vertex* (TV) relation is a boundary relation that associates with a triangle t its three vertices. The *Vertex-Triangle* (VT) relation is a co-boundary relation that associates with a given vertex the triangles in its star. The *Triangle-Triangle* (TT) relation is an adjacency relation that associates with a triangle t the three triangles sharing an edge with t.

2.2 Morse theory

Let us consider a domain $D \subseteq \mathbb{R}^2$ and a smooth (i.e., C^∞) scalar function f defined over D [70]. A point $p \in D$ is called a *critical point* of f if and only if the gradient of f vanishes at p. The determinant of the Hessian matrix $Hess_p(f)$ of the second order partial derivatives of f, evaluated in p, provides additional information about the critical points of f. The number of negative eigenvalues of $Hess_p(f)$, called the *index* of p, defines the type of the critical point. A critical point of index 0 is a *minimum*, a critical point of index 1 is a *saddle*, and a critical point of index 2 is a *maximum*. For each critical point p, the eigenvectors of $Hess_p(f)$ define the directions in which function f decreases. The lines everywhere tangent to the gradient of function f are called *integral lines*. An integral line connecting two critical points of consecutive index is called a *separatrix line*.

Morse theory [52] has been developed for smooth functions f such that all the critical points of f are non-degenerate, i.e., $Hess_p(f) \neq 0$. In such cases f is said to be a *Morse function*. The set of critical points and integral lines define a decomposition of D based on the regions of influence of the critical points. If we consider a critical point p of index k, the integral lines converging at p form a k-cell, called the *descending manifold* of p. The descending manifold of a maximum is a region, the descending manifold of a saddle is a line and that of a minimum, a point. The collection of all descending manifolds form the *descending Morse complex*. Dually, integral lines originating at p form a $(2-k)$-cell, called the *ascending manifold* of p. Here, the ascending manifold of a maximum is a point, the ascending manifold of a saddle is a line and that of a minimum a region. The collection of the ascending manifolds forms the *ascending Morse complex*.

Figure 1(a) shows the critical points of f, namely minima (▼), saddles (◓), or maxima (▲). Lines indicate integral lines, while bold lines indicate separatrix lines connecting minima to saddles and saddles to maxima. In Figure 1(b), we depict in red the descending manifold defined by the integral lines having destination in the maximum. In Figure 1(c),

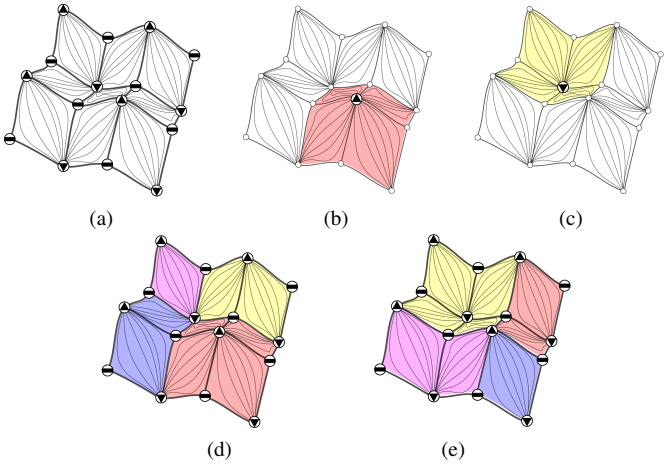


Fig. 1 (a) Minima (▽), saddles (⊖), and maxima (△) and the integral lines connecting them. (b) Descending manifold corresponding to a maximum. (c) Ascending manifold corresponding to a minimum. (d) Descending Morse complex and (e) ascending Morse complex.

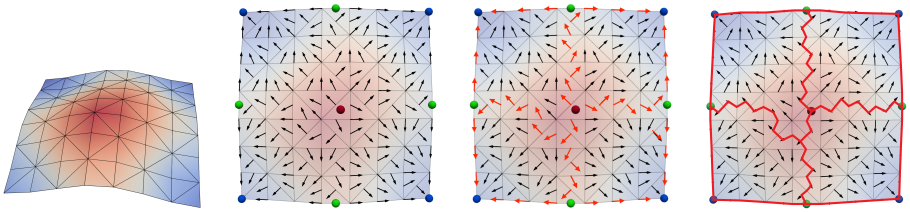
we depict in yellow the ascending manifold defined by the integral lines having their origin at the minimum. The collection of all descending and ascending manifolds defines the descending and ascending Morse complexes, respectively, depicted in Figure 1(d) and Figure 1(e). In this work, we are specifically interested in the *critical net* that compactly describes the terrain morphology. The *critical net* is a network having its nodes at the critical points, and as arcs the separatrix lines connecting critical points of consecutive indexes. In Figure 1(a) the critical net is depicted as a set of lines in bold connecting critical points.

2.3 Discrete Morse theory

In the applications we deal with scalar fields sampled at discrete locations within a domain. To this aim, the results of Morse theory, defined in the smooth case, have been extended by its discrete counterpart, called *Discrete Morse Theory* [29]. By assuming a scalar function F, defined at the vertices of a triangle mesh Σ, discrete Morse theory allows for the computation of a combinatorial gradient approximating the gradient of F, also called *Forman gradient*. The Forman gradient is defined by a collection of simplex pairs such that a k-simplex of Σ is paired with a $(k-1)$-simplex or a $(k+1)$-simplex, and each simplex of Σ is in at most one pair. A k-simplex involved in no pairs is called a *critical* simplex of index k.

A gradient pair can be viewed as an *arrow* formed by a head (k-simplex) and a tail ($(k-1)$-simplex). 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*). In a triangle mesh, unpaired simplices can be: critical triangles indicating maxima, critical edges indicating saddles, and critical vertices indicating minima. Figure 2(b) shows the Forman gradient computed on the triangle mesh shown in Figure 2(a). Black arrows indicate gradient pairs. Red points indicate critical triangles, green points indicate critical edges, and blue points indicate critical vertices.

In the same way, critical simplices are the discrete counterpart of critical points, and sequences of gradient pairs are the discrete counterpart of the integral lines. We call a V-



(a) (b) (c)

Fig. 2 (a) A TIN with elevation depicted according to a diverging blue-red colormap. (b) Forman gradient, (d) separatrix V-paths, and (c) the corresponding critical net.

path a sequence of simplices $[\sigma_0, \tau_0, ..., \sigma_i, \tau_i, ..., \sigma_q, \tau_q]$ such that σ_i and σ_{i+1} are on the boundary of τ_i and (σ_i, τ_i) are paired simplices, where $i = 0, ..., q$. A *separatrix V-path* is a triple (τ, ρ, σ), where τ and σ are two critical simplices having consecutive indexes and ρ is a V-path connecting τ to σ. In a triangle mesh Σ, we have separatrix V_1-paths connecting a critical edge to a critical vertex and separatrix V_2-paths connecting a critical triangle to a critical edge. In Figure 2(c) separatrix V-paths are depicted in red.

V-paths and separatrix V-paths are used to extract features from the Forman gradient, including the critical net. Specifically, within the framework of Forman theory, the vertices of the critical net are the critical simplices of V. Arcs of the critical net are the separatrix V-paths connecting them. A geometrical interpretation of the critical net is given by connecting tails and heads of all the arrows in the separatrix V-paths. Figure 2(d) shows in red the critical net computed by following the separatrix V-paths.

3 Related Work

In this Section, we review some related work on connectivity-based data structures for triangle meshes, on hierarchical spatial indexes for maps and meshes, and on techniques for topological data analysis.

3.1 Connectivity-based data structures

Connectivity-based data structures extend graph-based data structures for supporting the efficient extraction of connectivity relations (see [16] for a survey). A variety of connectivity-based data structures have been developed in the literature for triangle meshes [16]. The most widely used data structures are *triangle-based* ones, which encode the vertices and the triangles of the mesh but not the edges. The most compact triangle-based representation is the *indexed data structure*, which encodes the vertices and the triangles of the mesh, and, for each triangle, the references to its vertices. The *Indexed data structure with Adjacencies* (*IA* data structure) [59,55] extends the indexed data structure by explicitly encoding the adjacencies between triangles through the *Triangle-Triangle* relation and, for each vertex, the index of a triangle incident into it.

Other triangle-based data structures are the *Corner Table (CoT)* [61] and the *Sorted Opposite Table (SOT)* [32] data structures, in which the connectivity of the triangles is encoded through the concept of *corner*. Given a triangle t, a corner is a reference to an edge-adjacent triangle of t associated with one the vertices of t. Each triangle is thus identified by three

corners. The storage requirement of the CoT data structure is the same that of the IA data structure, as shown in [16]. The SOT data structure introduces a compact version of the CoT and IA data structures, by encoding only the triangles of the mesh and the *Triangle-Triangle* relation, thus requiring about 50% of the storage of the other two representations. However, the SOT data structure is only suited for static meshes and static applications, since modifications to the mesh require the global reconstruction of the *Triangle-Vertex* relation. *Edge-based* data structures are also used for triangle meshes, but they have been shown to be more verbose than triangle-based ones [16], while providing the same computational performances.

3.2 Hierarchical spatial indexes

Hierarchical spatial indexes use a recursive subdivision of the space on which the objects of interest are embedded according to different refinement rules. The main classification is between *regular* refinement and *bisection* refinement [5]. Regular refinement on rectangular blocks generates *quadtrees* in 2D space, and *octrees* in 3D space, while the bisection refinement of axis-aligned hyper-rectangles bisected by axis-aligned hyperplanes generates *kD-trees*. These decompositions have been originally defined for indexing point sets. They subdivide the space either into blocks of equal size, generating *Point-Region (PR)* quadtrees and kD-trees [57] or by using the positions of the points, generating *point quadtrees* and *point kD-trees* [28]. In the following, we review spatial indexes dealing with connected entities and maps (see [63] for an in-depth treatment of the subject).

The class of *Polygonal Map* (PM)-quadtrees [64] extends the PR-quadtree to represent polygonal maps in 2D space, considered as collections of segments intersecting only at most at their extreme vertices. There are three variants of a PM-quadtree, namely the *PM_1-quadtree*, the *PM_2-quadtree* and the *PM_3-quadtree*. These differ in their subdivision rule, but they all maintain a list of edges in their leaf blocks.

The *Randomized Polygonal Map (PMR)-quadtree* [54,35] is an index for collection of line segments in the plane (not necessarily forming a polygonal map). In a PMR-quadtree, if the insertion of an edge causes the number of edges in a leaf block to exceed a given threshold, the block is split, but only once, thus generating an order-dependent quadtree subdivision. In [45] it has been proven that the number of blocks in a PR-quadtree is proportional to the number of line segments and is independent of the depth of the tree.

A first attempt to extend the PM_2-quadtree to index triangle meshes is the *PM_2-Triangle quadtree (PM2T-quadtree)* [13], in which triangles, in place of edges, guide the partition into blocks. However, the PM2T-quadtree has two fundamental limitations. The first one is that each block indexes just one vertex, leading to a very deep hierarchy. The second limitation is that the spatial index is stored on top of the IA data structure, leading to a verbose data structure, which greatly limits its scalability.

Spatial indexes have been widely used for terrain rendering [10,31], providing efficient ways to generate adaptive meshes in in-core and out-of-core environments. Such representations are optimized for rendering, but not for geometric processing, as required in terrain analysis. We refer an interested reader to [58] for a description of such approaches.

3.3 Methods from topological data analysis

Morse theory [52] has been the basis for extracting topological structures, like Morse and Morse-Smale complexes [33]. Morse theory is defined for smooth functions, but recently two discrete counterparts have been developed, *piecewise linear Morse theory* [4] and *Discrete Morse Theory* [30]. In our work we are focusing on this latter, and we refer the reader to [15] for a complete analysis of these methods.

Among the many algorithms defined for computing a Forman gradient [15], the most efficient ones are those computing the gradient from a function sampled at the vertices of a triangle or tetrahedral mesh, or of a regular grid. The algorithm described in [41] is based on a divide-and-conquer approach and has the main drawback of introducing many spurious critical simplices. In [67], a similar approach, based on a weighted discrete function, has been defined for computing a Forman gradient on 2D regular grids. The algorithm is well suited for parallelization and significantly reduces the number of spurious critical cells. In [60], an algorithm is proposed for 3D regular grids that processes the *lower star* of each vertex independently. The *lower star* of a vertex v is the set of grid cells in the star of v, on which the function values at the vertices different from v is lower than the function value at v. This algorithm does not generate spurious critical cells. The algorithm has been extended to triangle [24] and tetrahedral meshes [72] by using a new implicit encoding of the discrete gradient. It has been shown [44] that for triangle meshes, the discrete Forman gradient finds a critical vertex for each piecewise linear minimum (i.e., a minimum found by applying piecewise linear Morse theory), while piecewise linear saddles and maxima are on the boundary of critical edges and critical triangles.

Recently, a great interest arose in the visualization community in the analysis of datasets having several scalar values per sampled point, called *multifield data*. Critical features are extracted from such data to highlight information about the scalar fields therein defined. Examples of such structures are the *Reeb space* [20], the *Joint Contour Nets (JCNs)* [7], *fiber surfaces* [8,68], the *Jacobi set* [18] and the *Pareto sets* [36]. Based on discrete Morse theory, a few approaches appeared extending to the multivariate case the extraction of a discrete gradient [2]. The first algorithm capable of dealing with real data of reasonable size is discussed in [39]. In the case of triangulated terrains, capturing the relationships among the different scalar fields in an "aggregate" value results particularly effective as it reduces the problem to a single scalar field visualization problem. A recent technique proposed by Nagaraj et al. [53] aims at computing an aggregate value for the multiple fields, indicating the presence of Jacobi sets [18]. This technique will be discussed in Section 8 together with our extension to triangle meshes for multifield analysis and visualization on terrains.

4 Terrain Trees

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

1. a triangle mesh Σ connecting the projections on the plane of the points in S;
2. a quadtree describing the nested subdivision of the domain D into square blocks in such a way that the vertices and triangles of Σ are associated with the leaf blocks of the quadtree subdivision.

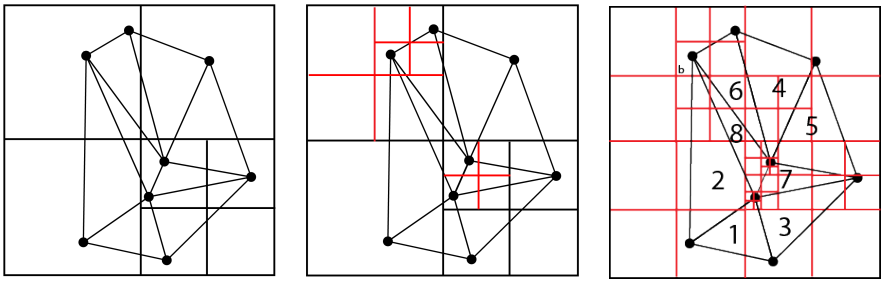


Fig. 3 Given the triangle mesh of Figure 4, a vertex threshold $k_V = 2$ and a triangle threshold $k_T = 2$, the Figure shows the spatial subdivision obtained with a PR-T Tree (a), a PM-T Tree (b) and a PMR-T Tree (c). In black are highlighted the spatial decomposition caused by the vertices threshold, while in red those caused by the triangles one. For the PMR-T tree we also highlight the triangle insertion order that drives the spatial decomposition (note that k_T is not a bucketing threshold for the PMR-T tree).

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

Similarly to spatial indexes for 2D maps, we have defined different criteria for domain subdivision, based only on the TIN vertices, on the TIN triangles and on both vertices and triangles. Thus, the Terrain trees family consists of three spatial data structures, namely the *PR-Terrain tree (PR-T tree)*, the *PM-Terrain tree (PM-T tree)* and the *PMR-Terrain tree (PMR-T tree)*, which are bucketed versions of the PM_3-quadtree, of the PM_2-quadtree, and of the PMR-quadtree for maps, respectively [64,54,55]. As we demonstrate in Section 9, bucketing is a crucial aspect for our spatial indexes, since it allows the indexing of much larger datasets, compared to existing representations in the literature.

A *PR-T tree* subdivides the domain D based on the vertices of Σ. Its subdivision rule uses a threshold k_V on the number of vertices contained in a leaf block b. If b contains more than k_V vertices, then b is recursively split into four blocks until this condition is met. An example of a PR-T tree is shown in Figure 3(a). The generation of a PR-T tree is entirely guided by the vertices of Σ, and, thus, the first step in the generation process is exactly the same as for the PR-quadtree [62]. Then, each triangle t of Σ is added to all the leaf blocks intersecting t, without affecting the spatial decomposition.

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

(1) a block b containing up to k_T triangles is a leaf block;
(2) a block b that contains more than k_T triangles is a leaf block if and only if all triangles intersecting b are incident in the same vertex v, which can be either inside or outside b;
(3) otherwise, the block is recursively split until either condition (a) or (b) is met.

An example of the subdivision obtained with the PM-T tree is shown in Figure 3(b). Note that a *PM-T tree* extends the PM_2 quadtree defined for maps, by adding bucketing thresholds k_V and k_T for both vertices and triangles. The generation of the initial hierarchical decomposition in a PM-T tree is entirely guided by the vertices, like in PR-T trees. Then, each triangle t in Σ is added to a leaf block b intersecting t, if and only if b contains less than k_T triangles. Otherwise, b is split until either condition (1) or (2) is met.

A PM-T tree is a sort of *bucketed* version of the PM2T-quadtree with some fundamental differences. The PM-T tree has a bucketing threshold for both vertices and triangles. The lack of a bucketing threshold in the PM2T-quadtree produces much deeper decompositions with a number of leaf blocks that is at least four times the number of vertices in Σ (see [14] for details). The PM-T tree uses a simple indexed representation encoding only *Triangle-Vertex* relations while the PM2T-quadtree encodes the triangle mesh through the IA data structure (see Section 5).

The subdivision strategy for the *PMR-T tree* is driven only by the triangles of Σ. It extends to triangle meshes the subdivision approach defined for sets of segments in the plane in [54]. A leaf block b is split if b intersects more than k_T triangles, where k_T is a user-defined threshold, but b is split only once, not recursively. The decomposition and, thus, the shape of the tree depends on the insertion order of the triangles. In [54] it has been proven that in a PMR quadtree built on the edges of a map, the number of edges intersected by a leaf block cannot exceed the sum of the splitting threshold and of the depth of the leaf block. For a PMR-T tree this result still holds and, thus, the number of triangles in a leaf block of a PMR-T tree can be at most equal to $d_b + k_T$, where d_b is the depth of the leaf block and k_T is the splitting threshold. An example of PMR-T tree is shown in Figure 3(c). For instance, in leaf block b there are three triangles indexed by it (i.e., triangles 2, 6 and 8), but no split operation is triggered, as the space is decomposed once when inserting triangle 8. Also condition $d_b + k_T$ is verified as b is at depth 3. Notice that the split condition may trigger unnecessary splits. For example, in Figure 3(c) the insertion of triangles 7 and 8 causes unnecessary split operations on the vertices sharing these two triangles.

A *PMR-T tree* is generated as follows. For each triangle t, and for each leaf block b intersecting t, t is added to b if and only if b contains less than k_T triangles. Otherwise, b is split and its triangles are distributed to the newly generated leaf blocks (i.e., the children of b). Triangle t is also added to the children of b intersecting it.

5 Implementation of Terrain trees

In this section, we describe the implementation of the Terrain trees in the *Terrain trees library* (TTL). The kernel of the tool contains the implementation of the three Terrain trees, the *PR-Terrain tree (PR-T tree)*, the *PM-Terrain tree (PM-T tree)* and the *PMR-Terrain tree (PMR-T tree)*, plus their generation algorithms and algorithms for answering basic spatial and connectivity-based queries. Other functions currently implemented and discussed in the following sections are the extraction of morphological features (see Section 6), the extraction of topology-based features (see Section 7), and the analysis of multivariate terrain data (see Section 8).

The encoding of the triangle mesh Σ in a Terrain tree consists of two arrays Σ_V and Σ_T, storing the vertices and the triangles of Σ, respectively. Each vertex v and triangle t is represented by a unique index, i_v and i_t within arrays Σ_V and Σ_T, respectively. Σ_V encodes the geometry of the terrain Σ by storing the longitude, latitude, and elevation and the other

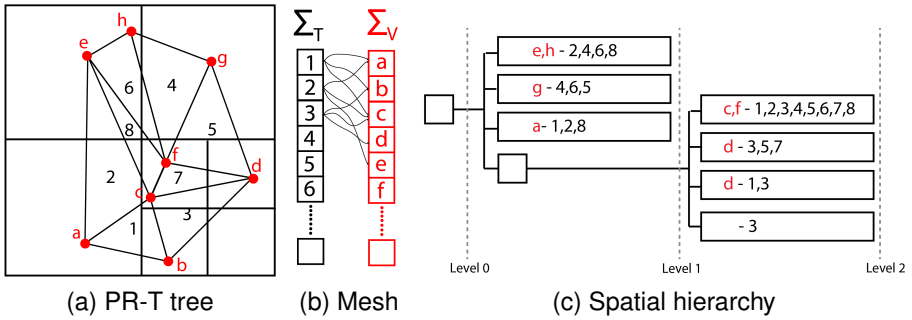


Fig. 4 (a) Triangle mesh and PR-Tree as encoded in a Terrain tree. (b) The mesh topology is organized by encoding, for each triangle, the boundary relation with its vertices. (c) The PR-T represents a hierarchy where each leaf block encodes the list of vertices contained in the block and the list of triangles intersecting the block.

field value(s) associated with each vertex v in Σ. Σ_T encodes the connectivity of each triangle by storing its three vertex indexes (see Figure 4(b)).

We use a pointer-based representation for the hierarchy describing the nested subdivision of a Terrain tree (see Figure 4(c)). Each internal block of a Terrain tree contains a reference to its parent block and a reference to its children. Each leaf block contains a reference to its parent block plus the information about the vertices and triangles, or only the triangles in the case of a PMR-T trees.

To encode the information associated with the leaf blocks, we use the compact encoding proposed in [27] for an arbitrary-dimensional connectivity-based data structure for simplicial complexes based on a vertex clustering. Such encoding uses the *sequential range encoding (SRE)*, a variant of the *run-length encoding* [34], that represents a run of consecutive indexes using two integers. The first (negative) index encodes the starting index of the run, while the second encodes the number of remaining elements of the run. The effectiveness of this compression increases with longer runs. This is exploited by representing all vertices inside a leaf block with a single run. Once we obtain the spatial decomposition, a single tree traversal is sufficient to reindex the vertices. The vertices indexed into the same leaf block get a contiguous range of indexes in the reindexed vertices array Σ_V. Within each leaf block, this range is represented as a pair of integers. Exploiting the spatial coherence for the triangles is more involved. The reindexing and compression of triangles is performed in such a way that, at the end, triangles indexed by the same set of leaves have contiguous indices in Σ_T. To obtain this representation, we traverse first the tree to extract, for each triangle t, the tuple of leaf blocks indexing t. Then, we extract the dual relation, i.e., we associate the list of triangles with each tuple of leaf blocks. Given this inverted relation, we extract a coherent ordering for the triangles of Σ_T, where triangles indexed by the same leaf tuple have contiguous indexes. Once we have this spatial ordering on the triangles, we apply it to the triangle list of each leaf block, and we compress this list by using the *SRE* compression. Finally, we update Σ_T to be consistent with this spatial ordering.

5.1 Spatial and connectivity-based queries in a Terrain tree

We have developed algorithms on Terrain trees for answering two fundamental spatial queries, namely a *point location* and a *window* query, and for extracting connectivity-based relations (described in Section 2), which enable the local traversal and processing of the underlying mesh. The point location query consists of finding the triangle (or triangles) containing a given query vertex, while the window query consists of finding all the triangles which intersect an axis-aligned rectangular window.

We have implemented algorithms for connectivity-based queries which extract the connectivity relations discussed in Section 2. Since the indexed TIN representation underlying a Terrain tree encodes the TIN vertices and triangles, extracting the vertices and edges of a triangle (these latter being expressed as vertex pairs) does not require the use of the tree structure unless we combine the extraction of such information with a window query when focusing on portions of the TIN. Extracting co-boundary relations at the vertices of the mesh efficiently and massively, as our experiments show, are fundamental for computing morphological features, the discrete Forman gradient estimators, and the critical net (see Sections 6 and 7). We describe here how we extract some fundamental co-boundary relation, namely the *Vertex-Triangle (VT)*, the *Vertex-Vertex (VV)*, and the *Vertex-Edge (VE)* relations, and the *Edge-Triangle (ET)* relation, as well as the *Edge-Vertex (EV)* relation inside a leaf block of the Terrain tree. These are used in the terrain analysis algorithms presented in the following sections.

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

Extracting the *Edge-Triangle (ET)* relations in a block b is slightly more involved, since the edges are not explicitly encoded in a Terrain tree. The algorithm iterates over the triangles in a block b and extracts the edges on their boundary provided by the *TE* relation. An edge e belonging to the boundary of a triangle t is considered *internal* to b if it has at least one vertex indexed by b. Each internal edge e is encoded in a local associative array having as key e and as value a pair containing the index of the two triangles in the co-boundary of e. These two triangles are identified during a single iteration on the triangles in b. The strategy for extracting the *Edge-Vertex (EV)* relation in a block b is similar to the one for extracting the ET relation. The algorithm iterates over the triangles of b and extracts their edges. Since each edge is encoded as a pair of vertex indices, we add those edges having at least one vertex indexed by b to the output.

6 Morphological Terrain Features

We have developed and implemented in the *Terrain trees library* algorithms which extract classical morphological terrain features, namely *triangle* and *edge slope*, *curvature* and *roughness*. An experimental comparison among the various Terrain trees and the IA data structure in computing such features is presented in Section 9.

The *slope* of an edge or a triangle in a TIN represents the steepness and the direction of its extent, where the steepness is the absolute value of the slope. A zero value of the slope indicates horizontality. The direction is increasing if the slope is positive and decreasing if the slope is negative. Specifically, the *edge slope* of an edge e is the angle between e and its projection on the horizontal plane defined by the z-coordinate of its lowest endpoint. In other words, given $v_i = \{v_{i_x}, v_{i_y}, v_{i_z}\}$ and $v_j = \{v_{j_x}, v_{j_y}, v_{j_z}\}$ the endpoints of e. Let v_i the endpoint of e with the minimum z-value, we consider the projection v'_j of vertex v_j on the plane $z = v_{i_z}$. The slope of edge e is the angle $\widehat{v_j v_i v'_j}$. For computing edge slopes, we need to extract the edges as pair of vertices, by using the *TV* relation, since the edges are not explicitly encoded. Extracting the edges requires an auxiliary data structure. The decomposition of the mesh defined by the blocks becomes computationally relevant, since the auxiliary data structure is created in each block independently and discarded after processing the block. This makes it possible to compute slopes in a region of interest without iterating through the entire domain.

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

We approximate the curvature at the vertices of a TIN by using a discrete approach. In our previous work [49,50], we have developed three discrete curvature approximations of Gaussian and mean curvature, and compared and evaluated them in [48] for curvature estimation on a TIN. Our results showed that all curvature estimators provide similar results, also when used as the basis for TIN segmentation, and that *concentrated curvature* is the least sensitive to noise.

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

The computation of curvature requires extracting, for each vertex v of the TIN, the set of triangles incident at v, i.e., extracting the *Vertex-Triangle* relation, which is performed as discussed in Section 5.1. As the internal vertices and boundary vertices have different equations for estimating concentrated curvature, a preprocessing step to identify boundary vertices is performed. For each vertex v, we check the number of edges $|e|$ and the number of triangles $|t|$ in the boundary of the star of v. A vertex is on the boundary when $2|e| \neq 3|t|$.

There are several ways to define surface roughness, and the most commonly used is the standard deviation of local elevation at each vertex, evaluated based on the neighbors of the vertex itself [66]. We have extended the definition that is given for raster grids to TINs by considering the vertices adjacent to a vertex v, which are the ones sharing an edge with v. Based on that definition, the *roughness* at a vertex v in a TIN is computed as:

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

where m is the number of vertices adjacent to v plus v itself, z_1,z_2,...,z_m are the elevations at such vertices, and $\bar{z}$ is the average of those elevations. From Equation 1, the roughness computation requires the extraction of the VV relation of v (see Section 5.1 for details).

7 Topology-based Terrain Segmentation

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

In Subsection 7.1, we present the general strategy for computing a discrete Morse (Forman) gradient and a distributed approach based on Terrain trees. In Subsection 7.2, we discuss how to compute the critical net and present an algorithm for Terrain trees.

7.1 Forman gradient computation

We consider a triangle mesh Σ and an elevation function $f : \Sigma_V \longrightarrow \mathbb{R}$ defined on the vertices of Σ. The algorithm for computing the Forman gradient is based on the extension to TINs of the algorithm proposed for regular grids [60]. It consists of three major steps, which are described below and illustrated by referring to Figure 5.

Step 1 (Indexing). The first step requires computing a total order I on the vertices of Σ. The total order will serve as guiding schema for the subdivision of the triangles, edges and vertices of Σ in independent sets. This is done by *Simulation of Simplicity* [21], i.e., by sorting the vertices of Σ in ascending order and by assigning a unique index to each of them. In Figure 5(a) we indicate the index in I of each vertex of a triangle mesh. On Terrain trees, *Step 1* is executed by sorting the vertices stored in the global vertex array.

Step 2 (Partition). Σ is then subdivided by associating each vertex v with the set of edges and triangles having the same value of I as v. I is extended to the edges and triangles of Σ via $I(\sigma) := \max_{v \in \sigma} I(v)$, where σ is either an edge or a triangle of Σ and v is a vertex on the boundary of σ. For each edge or triangle σ, we denote as v the vertex of σ with maximum value of I. For this reason, this set of triangles and edges associated with v is called the *lower star* of v according to I and denoted $L_I(v)$. It can be proved that each triangle or edge in Σ belongs to exactly one lower star. Then, the lower stars associated with the vertices form a partition of Σ and thus they can be processed in parallel. In Figure 5(a) the lower star of vertex 6 is depicted with bold lines.

On Terrain trees, *Step 2* is implemented through a tree traversal where each leaf block b is processed once. In a leaf block b, the algorithm extracts the lower star $L_I(v)$ for each vertex v in b by retrieving the set of triangles incident in v (*Vertex-Triangle (VT)* relation) and the set of edges *(Vertex-Edge (VE)* relation), and by computing their values of I at runtime.

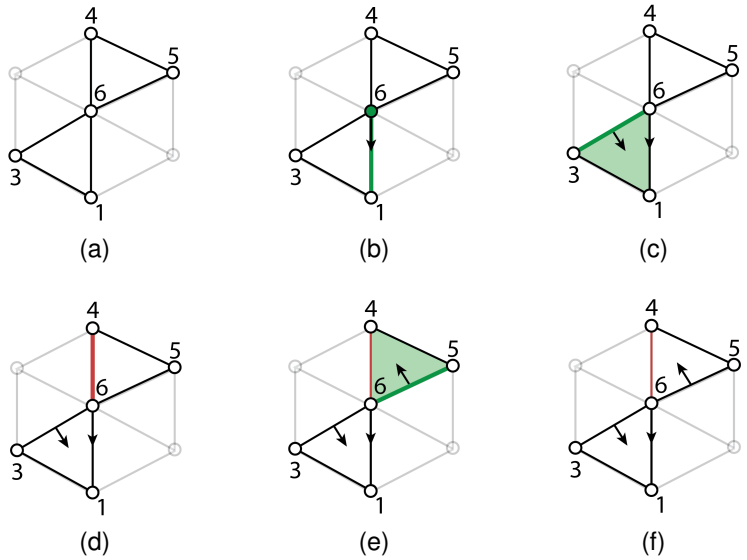


Fig. 5 Homotopy expansion [60] computed on the simplices belonging to the lower star of vertex 6.

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

We initialize a set LS with $L_I(v)$. If $LS = \{v\}$, v is declared as a critical vertex. Otherwise the pair (v,e) is created by pairing v with the edge e in LS having $\min_{v \in e} I(v)$. In Figure 5(b) edge $(6,1)$ is selected to be paired with vertex 6.

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

- if t has no unpaired edges on its boundary, then it is classified as critical,
- if t has exactly one unpaired edge e on its boundary, then it is paired with e

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

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

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

7.2 Extracting the critical net

Computing the critical net means visiting all the separatrix V_1-paths connecting critical vertices and edges, and all the separatrix V_2-paths connecting critical edges and triangles (see

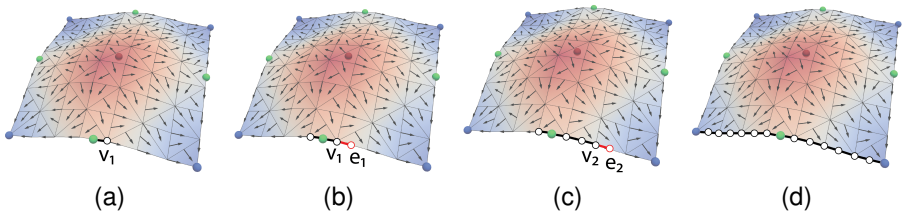


Fig. 6 Reconstruction of a portion of the critical net connecting a critical saddle with two critical minima. (a) Starting from a critical edge, the boundary vertices are connected with the edge. (b) The edge e_1 paired with v_1 is added to the stack. (c) The edge e_1 is extracted from the stack, connected with its paired vertex v_1. The other vertex on the boundary of e_1 (i.e., v_2) is retrieved and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until both paths reach critical vertices (minima).

Section 2.3). The geometrical representation of the critical net is computed by connecting the barycenters of the triangles and edges visited in the separatrix V-paths. For the vertices of the triangle mesh, we consider the points themselves.

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

Figure 6 shows the extraction of the critical net limited to a critical edge and two critical vertices. Starting from the critical edge (Figure 6(a)) the two vertices in its boundary are extracted and connected with the critical edge. Edges paired with such vertices are inserted in the stack Q. In Figure 6(b), edge e_1 is extracted from the stack and connected with its boundary vertex v_1. In Figure 6(c), the other vertex on the boundary of e_1 (i.e., v_2) is connected to e_1 and its paired edge e_2 is added to the stack Q. The depth-first traversal continues in the same manner until two critical vertices are encountered (Figure 6(d)).

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

Figure 7 shows the extraction of the critical net limited to a critical edge and a critical triangle. Starting from a critical edge (Figure 7(a)) the triangle t_1 in its coboundary is retrieved. The edge e_1 paired with t_1 is inserted in the stack Q. In Figure 7(b), the edge e_1 is extracted from the stack and connected with the triangle paired with it (i.e., triangle t_1). In Figure 7(c), the next triangle on the coboundary of e_1 (i.e., t_2) is connected to e_1 and its paired edge e_2 is added to the stack Q. The depth first traversal continues in the same manner until a critical triangle is encountered (Figure 6(d)).

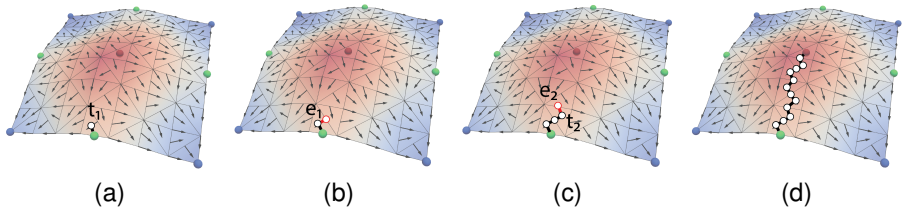


Fig. 7 Reconstruction of a portion of the critical net connecting a critical saddle with a critical maxima. (a) Starting from a critical edge, the coboundary triangle (i.e., t_1) is visited. (b) The traversal starts by adding the edge e_1, paired with t_1, to the stack. (c) The edge e_1 is extracted from the stack and connected with its paired triangle t_1. The other triangle in the coboundary of e_1 (i.e., t_2) is extracted and its paired edge e_2 is added to the stack. (d) The portion of the critical is reconstructed repeating the same steps until a critical triangle (maximum) is reached.

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

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

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

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

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

- the new V-paths starting from the critical edges indexed by b are visited. Notice that an edge could be shared by two blocks. To process each critical edge once, we use the following convention: a critical edge is indexed by b if and only if b indexes the vertex with the higher label value in the vertex array Σ;
- the *dangling paths* that are entering b are then expended. Each time a dangling path is expanded, the corresponding entry in the auxiliary data structure storing dangling paths is removed. In this way, the storage requirement of this structure is kept negligible during the extraction process.

Notice that the visit of a V-path can be interrupted several times, as it can cross multiple leaf blocks. Once the visit of the V-paths in b is terminated, the connectivity relations computed are saved into the cache.

8 Multifield Visualization

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

Extracting and visualizing descriptive information for multifield data is a major challenge. The technique implemented in this work relates to the notion of *Jacobi set* [19]. The Jacobi set of a collection of real-valued Morse functions defined on a common manifold is the set of all points where the function gradients are linearly dependent, which is directly related to the rank of the Jacobian matrix. This definition inspired numerical techniques aimed at rendering a single function built out of the multiple scalar fields in a way compatible with the relationships among scalar fields. In [53], a comparative measure is defined as measure for the evaluation of the local coherence among different scalar fields based on the gradient of the fields.

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

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

The multifield comparison measure in [53] is defined as the norm of such matrix $\eta^F(v) = ||dF(v)||$. To speed up the computation, the estimation of $\eta^F(v)$ is reduced to the root of the maximum eigenvalue of the matrix $(dF(v))^T(dF(v))$. To compute $\eta^F(v)$, when v is a vertex of a TIN, we need to estimate partial derivatives at v. In [46] several methods have been analyzed, and the best method has been shown to be the *Average Gradient on Star (AGS)* method [51], which is both accurate and efficient. In AGS, the gradient at a vertex v is approximated by taking the average of the gradients estimated at the triangles incident in v.

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

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

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

To compute the gradient at vertex v, we need to compute the so-called *mixed area* [51]. Let t be a triangle and p_i, p_j, p_k the vectors of coordinates of its three vertices. If t is non-obtuse, the contribution of t to the mixed area is $\frac{1}{8}(|p_i p_k|^2 cot\angle p_j + |p_i q_j|^2 cot\angle p_i p_k)$. If t is obtuse, there are two cases: if the angle at v is obtuse, the mixed area will be half of the triangle area, while if the angle at v is not obtuse, then the area will be a quarter of the triangle area. Then, the gradient at v is the weighted average of gradients computed at each triangle incident in v weighted by the corresponding mixed area.

Since the computation of the gradient at vertex v relies on the gradients at all the triangles intersecting at p, the computation of such gradient in the Terrain trees requires extracting the *Vertex-Triangle (VT)* relation, as discussed in Section 5.1.

Table 1 Overview of experimental datasets. For each terrain, we list the number of vertices $|\Sigma_V|$ and triangles $|\Sigma_T|$.

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

9 Experimental Results

In this section, we study the performances of the Terrain trees family. TINs used are computed from LiDAR point clouds by means of a Delaunay triangulation algorithm from the *CGAL* library [9].

The characteristics of each TIN are reported in Table 1. We have used a total of seven TINs with a number of vertices ranging from 34 million to 193 million. GREAT SMOKEY MOUNTAIN, CANYON LAKE GORGE and BIG CREEK are three datasets provided by the OpenTopography repository[56]. For each of them we have computed a single TIN. The original SONOMA COUNTY dataset includes more than 60 billion points. We created four different datasets by subsampling the original point cloud at four different resolution levels. For each point cloud obtained we have computed a TIN that is used in our experiments.

Our experimental evaluation addresses five aspects: (i) calibration of the thresholds guiding the construction of Terrain trees, (ii) evaluation of the requirements for initializing Terrain trees, (iii) extraction of connectivity-based relations, (iv) computation of morphological and topology-based features, and (v) analysis and visualization of multifield data. The hardware configuration used for these experiments is a dual Intel Xeon E5-2630 v4 CPU at 2.20Ghz, and 64GB of RAM. The source code of the Terrain trees library implementing the Terrain trees is available at [26]. The source code of the LibTri library implementing the IA data structure is available at [25].

9.1 Selection of thresholds for Terrain trees generation

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

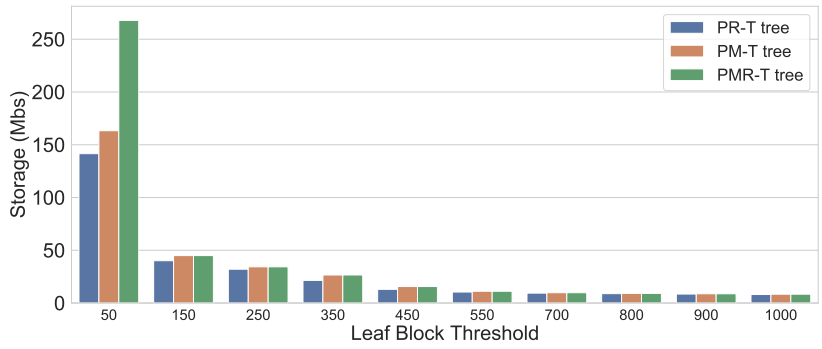


Fig. 8 The Storage costs for storing the hierarchical index of the Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.

(iii) time requirements for answering the most common connectivity-based query, i.e., the extraction of *Vertex-Triangle* relation.

Since all Terrain trees encode the TIN with the same indexed representation, we focus only on the storage costs of the hierarchical index without considering the storage costs for encoding information on the vertices and on the triangle connectivity. Figure 8 shows these results. The storage costs show a sharp decrease with smaller thresholds, while they remain nearly constant with larger ones. The differences in storage costs among the three indices are more noticeable when the thresholds are small, while the costs become nearly identical for larger ones. The storage cost of the hierarchical index is closely related to the number of nodes in a Terrain tree. When using larger thresholds (i.e., k_V greater than 550), Terrain trees have the same number of nodes, and this means that the spatial index is losing its effectiveness at decomposing the embedding space. This latter result highlights that the threshold on the triangles is the one guiding the spatial decomposition when both thresholds become larger.

Figures 9(a) shows the time required for generating the spatial decomposition of a Terrain tree on GREAT SMOKEY MOUNTAIN dataset. Generation times decrease for all types of Terrain trees when using larger thresholds since each leaf block can contain more vertices and triangles, and the spatial decomposition is obtained by executing fewer split operations. Also, since the subdivision rule for the PMR-T tree is not recursive, its construction is always faster than the other two rules, that, on average, have similar generation times.

We evaluate now how different thresholds affect the extraction of the *Vertex-Triangle (VT)* relation. This relation is key to most of the applications defined in our framework. Figure 9(b) summarizes the results we have obtained. We notice that the larger the threshold used, the faster the extraction of the *VT* relation is. The larger time drop is noticeable on smaller thresholds, while for larger thresholds the time difference becomes smaller. Since PR-T and PM-T trees explicitly encode the range of vertices, they show similar performances, and are always faster than PMR-T trees, using from 30% to 70% less time, as the latter has to compute such vertex ranges at run-time.

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

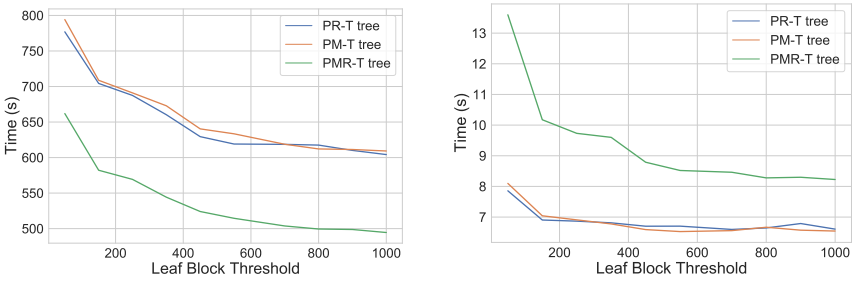


Fig. 9 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on GREAT SMOKEY MOUNTAIN dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

Table 2 Overview of Terrain trees. For each Terrain tree, we list the thresholds k_V and k_T and the number of blocks in the index ($|N|$).

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

representation. For such thresholds, leaf blocks encode more entities, leading to a reduced speedup gain and to higher storage requirements for encoding application-specific auxiliary data structures. Based on the results and evaluations above, we choose thresholds that generate trees having a number of nodes between 300K and 500K, as we have noticed that such spatial indices are neither too coarse nor too deep, with overall good performances at generating and encoding Terrain trees and answering connectivity queries.

9.2 Terrain trees evaluation

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

We generate a PR-T tree, a PM-T tree, and a PMR-T tree for each TIN. A single value for k_v and k_t is selected for each dataset according to the results discussed in Section 9.1. Table 2 shows the thresholds selected and the total number of internal and leaf nodes of each Terrain tree. Notice that PR-T trees and PMR-T trees use only one threshold value

Table 3 Comparison of storage, expressed in megabytes (*MB*) and gigabytes (*GB*), for the underlying TIN, Terrain trees and IA data structure. O.O.M. stands for Out Of Memory.

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

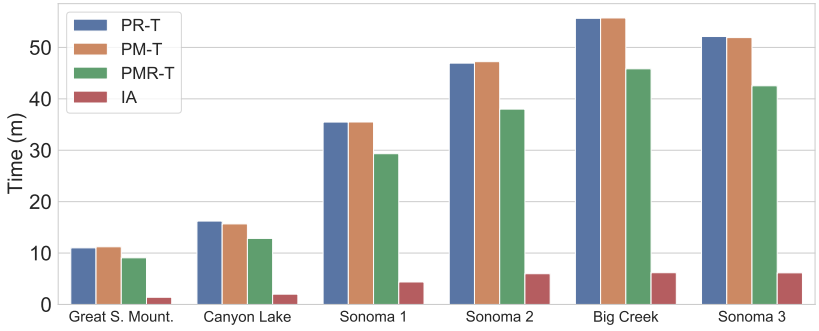


Fig. 10 Comparison of total timings, expressed in minutes (m), for generating a Terrain tree or the IA data structure.

(i.e., k_v and k_t, respectively), while PM-T trees use both. As shown in Table 2, the number of nodes in a PM-T tree and a PMR-T tree of the same dataset is always similar, which leads to comparable storage costs. The number of nodes in a PR-T tree is always smaller than in the other two trees. These results match the ones of Section 9.1, which show that a PR-T tree always has a lower storage cost compared to the other two Terrain trees.

Table 3 shows the storage costs of the Terrain trees and of the IA data structure. Since both Terrain trees and IA data structure encode an indexed representation of the TIN, we represent this storage requirement separately, and thus, the storage costs shown in Table 3 consider only the requirements for encoding the spatial index in a Terrain tree, and for encoding adjacency and coboundary relations (i.e., *Triangle-Triangle* and partial *Vertex-Triangle* relation) in the IA data structure. The total storage requirements can be easily computed by adding the storage cost of the TIN to the corresponding Terrain trees or IA data structure overhead cost. Note that the overhead cost of Terrain Trees is between 1% and 3% of the overhead cost of the IA data structure. When considering also the cost of the underlying indexed TIN representation, a Terrain tree can encode the same dataset using, on average, 36% less storage than IA data structure. The differences between the three Terrain trees are minimal. The PR-T tree is the most compact since it generates fewer leaf blocks compared to the other two.

Figure 10 shows the time requirements for generating a Terrain tree or the IA data structure. The generation time do not consider the time needed to load the TIN from file, but only the time for generating the corresponding data structure. The generation times for the IA data structure are about 10% of those of Terrain Trees. This is expected since the IA computes only the adjacencies between triangles and a partial *VT* relation. Terrain Trees,

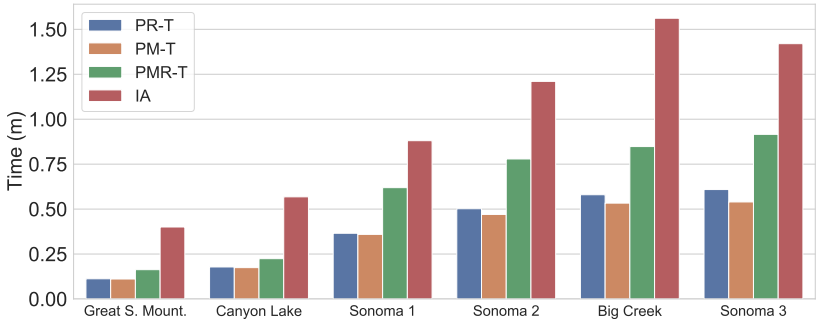


Fig. 11 Comparison of total timings, expressed in minutes (m), for extracting the VT relations.

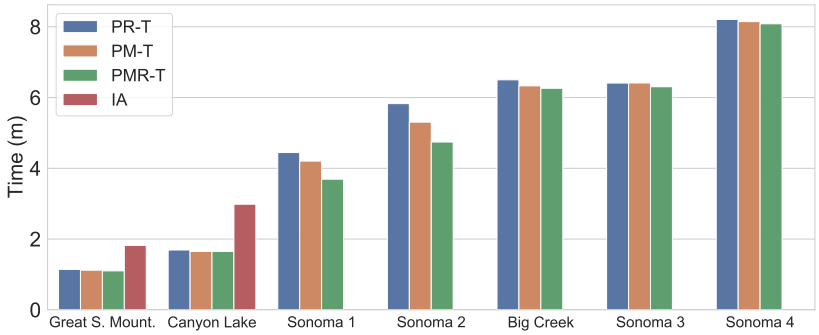


Fig. 12 Comparison of total timings, expressed in minutes (m), for extracting edge slopes.

instead, are created by first computing the spatial decomposition and then compressing its representation, as described in Section 5. Also in this case, the differences between the three Terrain trees are minimal, and the generation of a PMR-T tree is always 20% faster than that of the other two, which is consistent with the findings in Section 9.1.

As shown in Figure 11, Terrain trees can always extract *VT* relations faster than the IA data structure. PR-T and PM-T trees use from 57% to 72% less time than the IA data structure. PMR-T trees use, on average, from 30% to 70% more time compared to the PR-T and PM-T trees, still saving at least 30% time compared to IA data structure. The differences among Terrain Trees are due to the encoding of the vertex ranges. In PR-T and PM-T trees, such ranges are explicitly encoded, while in PMR-T trees, they are computed at run-time on a block-by-block basis.

9.3 Computing morphological features

In this section, we evaluate the performances for computing morphological features, as described in Section 6. Results are shown in Figures 12, 13, and 14.

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

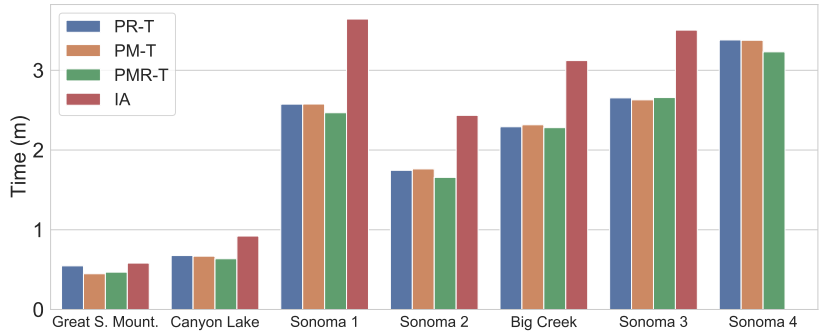


Fig. 13 Comparison of total timings, expressed in minutes (m), for extracting concentrated curvatures.

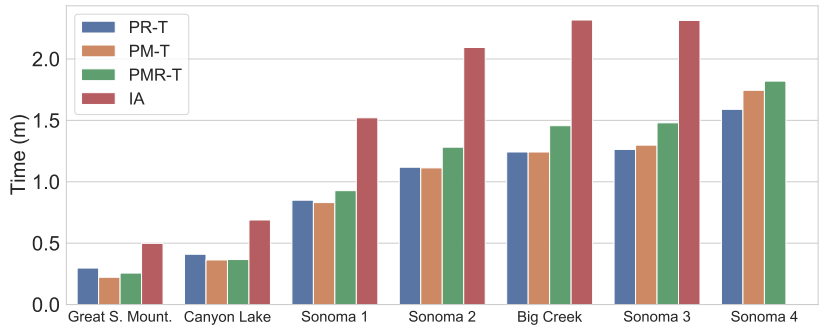


Fig. 14 Comparison of total timings, expressed in minutes (m), for extracting roughness values.

As edges are not explicitly encoded neither in Terrain trees nor in the IA data structure, we have to use an auxiliary lookup table in both implementations for encoding the slope values without duplicates. Terrain trees enable the usage of a local data structure within each leaf block, and this reduces both the cost of encoding and accessing the lookup table. As shown in Figure 12, the slope estimation benefits by the use of a spatial index and a modular structure. We notice that computing edge slopes on Terrain trees requires from 37% to 45% less time and less memory than by using the IA data structure (considering also that the IA data structure goes out of memory five times). The difference among the three Terrain Trees is limited, since the extraction time for the *VT* relation accounts for a small portion of the overall slope computation time.

As discussed in Section 6, estimating the *curvature* requires visiting the star of each vertex (i.e., extract the *Vertex-Triangle* relation). As in Figure 13, the implementation based on Terrain trees is always faster than the one based on the IA data structure, as it requires from 25% to 30% less time. Both Terrain trees and the IA data structure require the same amount of space for encoding curvature values, while the size of the auxiliary data structures is negligible. The performances of the three Terrain trees are similar. This shows that extracting the vertices in a block at run-time does not affect significantly the performances of the spatial index.

Estimating *roughness* requires computing the *Vertex-Vertex* relation. In Terrain trees, the roughness computation is pretty efficient (see Figure 14), being from 36% to 55% faster than the corresponding procedure on the IA data structure. On larger datasets, PMR-T trees are

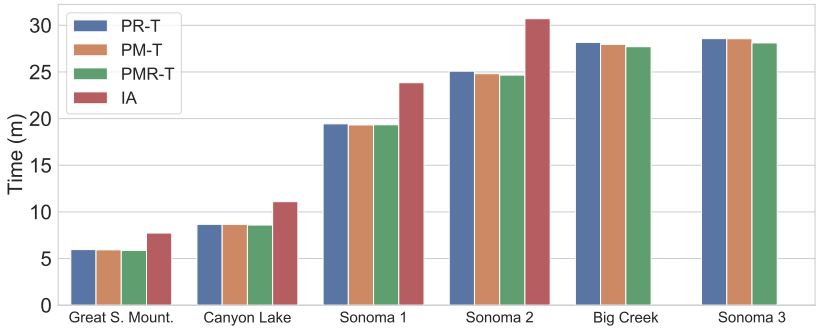


Fig. 15 Comparison of total timings, expressed in minutes (m), for computing the Forman gradient vector.

always slower than PR-T and PM-T trees, since they are less efficient at extracting the *VV* relation.

9.4 Computing topology-based segmentations

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

Forman gradient computation

As discussed in Section 7, computing the discrete gradient on a Terrain tree requires extracting the star of each vertex of the TIN, i.e., the *Vertex-Triangle* relation. As shown in Figure 15, Terrain trees are always faster than the IA data structure requiring about 20% less time. PMR-T trees are usually slightly faster than PR-T and PM-T trees, but overall the time difference between the three Terrain trees is small.

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

Critical net extraction

In a Terrain tree, the extraction of the critical net requires an intense navigation of the spatial index, and the extraction at run-time of connectivity relations that, in the IA data structure, are either explicitly encoded (*Triangle-Triangle* relation) or efficiently extracted (*Vertex-Vertex* relation). This application represents an interesting worst-case scenario for Terrain trees.

Both Terrain trees and the IA data structure use auxiliary data structures for extracting the critical net. The IA data structure uses a global stack to perform the TIN traversal. Conversely, a Terrain tree uses a cache of leaf blocks with expanded connectivity information as well as a list of dangling paths, plus a local stack within each leaf block (see Section 7.2). As shown in Figure 16, thanks to the lower storage requirements of Terrain trees, such auxiliary structures can be effectively encoded in memory, while the IA data structure goes out of memory on the three larger datasets.

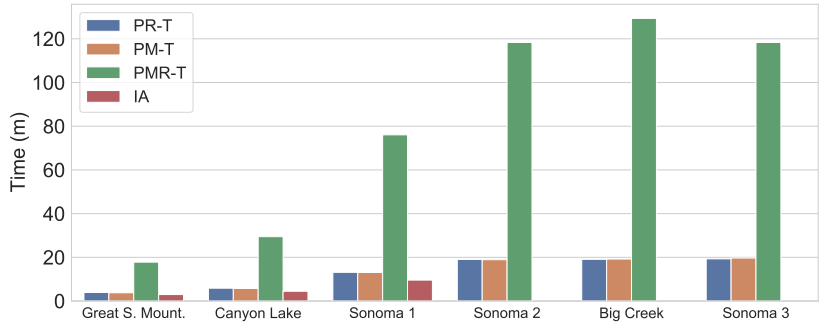


Fig. 16 Comparison of total timings, expressed in minutes (m), for computing the critical net.

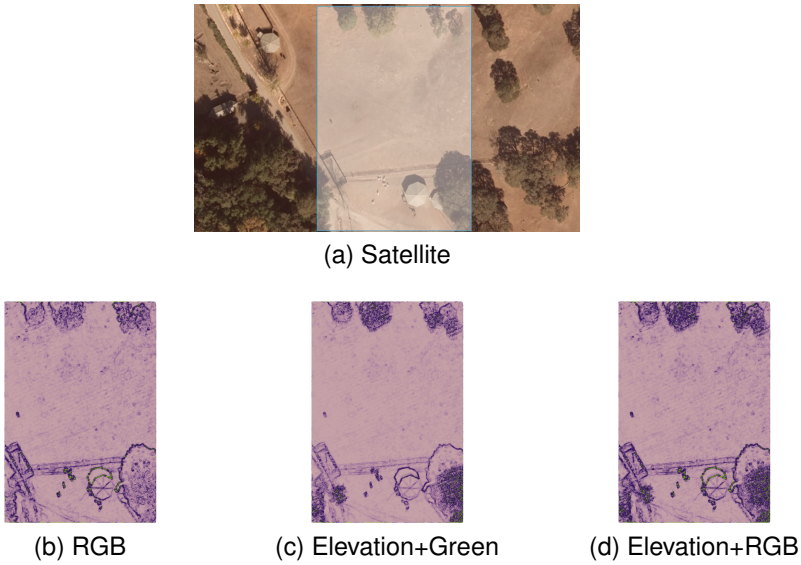


Fig. 17 The satellite map and the visualization of the multifield comparison measure on an area with few trees and some human artifacts. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), based on elevation paired with green values (c), and based on elevation paired with RGB values (d).

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

9.5 Multifield data visualization

We evaluate the performances of Terrain Trees for computing the multifield measure described in Section 8. We test our implementation for a visual analysis on two small datasets

(shown in Figures 17 and 18), and for a performance analysis against the IA data structure on three different areas of the SONOMA COUNTY dataset. Each dataset has a total of five scalar fields: elevation, a color field (encoded as a RGB triple), and roughness.

First, we visually analyze the algorithm performance on an area with few trees and some human artifacts (a fence and a small building). Figure 17 presents the raster image of this area (Figure 17(a)), and three output images obtained with our algorithm in which we used as input scalar fields: (i) the RGB values (Figure 17(b)), (ii) the green band paired with the elevation (Figure 17(c)), and (iii) the RGB values paired with the elevation (Figure 17(d)). Just using the RGB values, the algorithm can identify the boundary of the buildings and their shadows clearly, but it does not highlight precisely the trees. Pairing the green band with the elevation improves the identification of trees, while the human artifacts result smoothed and less clear. Lastly, if we pair the three RGB values and the elevation, the algorithm can correctly highlight both trees and human artifacts.

In order to understand the performance of our strategy in distinguishing forest areas from other land cover types (like rivers or streets), a region with higher tree density has been used (see Figure 18). We compare the visualization results when using just the RGB values (Figure 18(b)), and when pairing them with roughness (Figure 18(c)). In this case, we pair the RGB values with roughness instead of the bare elevation, since roughness has been proven to be a better estimator for identifying surface deformations in a terrain [66]. The visual comparison of the outputs shows that the multifield strategy is more precise in highlighting the different cover types, when also a geometric field is added to the identification procedure. Pairing roughness with the color values enables the identification of both the road crossing the forest and areas of low vegetation (Figure 18(c)), that cannot be clearly identified by just using the color (Figure 18(b)). Also, a narrow band representing the road in the forest can be identified clearly only if we include roughness values in our input fields.

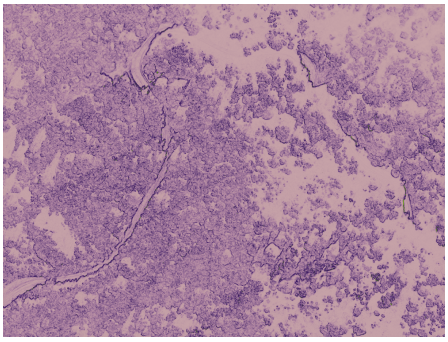
The visual analysis of the results shows that the multifield strategy can be effectively used for highlighting key areas in satellite datasets, and that the identification improves when pairing a geometric attribute with other scalar fields that are not spatial or geometric.

Finally, we compare the performance of Terrain trees and IA data structure at extracting the multifield measure on three datasets based on SONOMA COUNTY. Results are reported in Figure 19. Overall, the timing performances of the Terrain trees and IA data structure are very similar, even if we notice that Terrain trees are about 5% faster than the IA data structure. The performance difference is minimal if we compare the Terrain trees. Thanks to the initial lower storage requirements, Terrain trees can compute the multifield measure on all three datasets, while the IA data structure goes out of memory on the largest one.

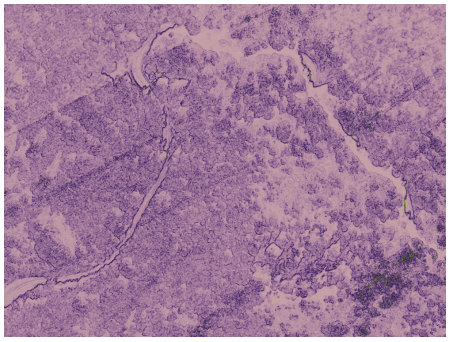
10 Concluding Remarks

We have presented a family of spatial data structures, the *Terrain trees*, for the efficient representation, analysis and visualization of triangulated terrains. A Terrain tree combines a minimal connectivity-based encoding of the underlying triangle mesh with a hierarchical spatial index, thus implicitly encoding other connectivity relations. Terrain trees consist of three spatial indexes that use different bucketed subdivision rules. By borrowing an idea presented in [27] for a distributed data structure for simplicial complexes in arbitrary dimensions, we use spatial coherence to reorder the indexed data, thus achieving the compression of both vertex and triangle information inside the spatial index. This enables high storage reduction and optimized algorithms.

(a) Satellite



(b) RGB



(c) Roughness+RGB

Fig. 18 The satellite map and the visualization of the multifield comparison measure on an area with high tree density. Figures show the satellite image (a), the multifield measure based on red, green and blue values (b), and based on roughness paired with RGB values (c).

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

We have experimentally demonstrated how the bucketing thresholds on the Terrain trees affect generation times, storage requirements, and performances in extracting a basic connectivity relation. This has enabled the optimal identification of the most appropriate threshold ranges. We have then experimentally demonstrated the efficiency of our data structure by comparing it against the most common and compact connectivity-based data structure, the IA data structure. Terrain trees require always less storage, they generally perform better than the IA data structure, and their effectiveness increases with the dataset size. Conversely, the difference in performances among the three Terrain trees is minimal. We have noticed, however, that spatial indexes explicitly encoding the vertices in each leaf block have shown better performances at computing those estimators which require the efficient extraction of

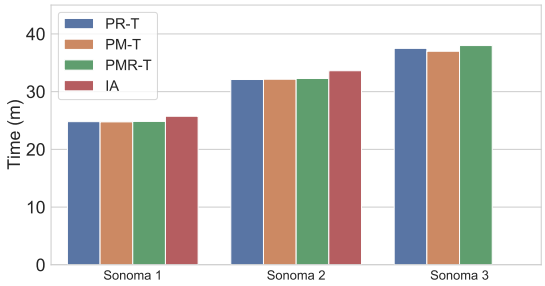


Fig. 19 Comparison of total timings, expressed in minutes (m), for computing the multifield measure.

the triangles incident in a vertex. The source code of Terrain trees library and of the library implementing the IA data structure, called *LibTri*, are available in the public domain [26, 25].

The experimental results showed that encoding the triangle meshes has a relevant impact on the storage cost. As future work, we will consider distributing the global arrays across the leaf block of the Terrain trees. Also, we plan to design an algorithm for computing the TIN at runtime, locally within each leaf block, and discard it when no longer needed. This will further reduce the memory footprint and enable the handling of even larger point clouds.

For topology-based analysis of the terrain, we are currently extending our tool with algorithms for geometric and topology-based simplification. Simplification algorithms have been developed for reducing the size of a TIN based on the edge contraction operator, but the major problem with TIN simplification algorithms is that they can create or remove critical points in an uncontrolled way. Topology-aware operators [38] have been defined to solve this issue by coarsening a TIN without affecting its topology. While effective, existing algorithms are sequential in nature and are not scalable enough to perform well with large terrains. We are currently developing a simplification algorithm in the Terrain Trees using the topology-aware edge contraction operator first introduced in [38]. Thanks to the compact and distributed representation of Terrain trees, this algorithm will improve both the memory and time requirements of the simplification procedure. Furthermore, we plan to investigate a new parallel topology-aware simplification algorithm that takes advantage of the spatial domain decomposition at the basis of Terrain trees.

Algorithms for geometric and topology-based simplification intensively update both the TIN and the Terrain tree. Currently, we are defining an update procedure that keeps the Terrain tree up-to-date after a simplification, in such a way that performances are not largely affected. In the future, we plan to extend this procedure to support also generic updates to the TIN, like adding new points or removing TIN vertices. The generic mechanism to update the Terrain tree is similar to the process performed after a simplification, but it might be more challenging if larger portions of the TIN would need to be re-triangulated.

The morphology of a terrain, represented by the critical net, can also be simplified by modifying the underlying discrete gradient [29]. Simplification strategies for the discrete gradient have been defined for reducing noise and for obtaining clearer and accurate representations of the critical net [22]. We plan to implement a simplification algorithm for the discrete gradient. This will result in more robust descriptions of the terrain morphology.

Lastly, we are currently studying an extension of the Terrain trees for distributed frameworks like Apache Spark [76] or MPI [11]. The distributed environment will increase the scalability of our approach dramatically. The hierarchical representation of the Terrain trees

is well suited to be organized in the distributed framework. The challenge here is defining a distributed algorithm for constructing the TIN in such context.

Acknowledgements This work has been mainly developed while Riccardo Fellegara was with the University of Maryland at College Park, USA. This work has been 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 BIG CREEK, CANYON LAKE GORGE, GREAT SMOKEY MOUNTAIN, and SONOMA COUNTY point clouds are kindly provided by the OpenTopography Facility [56] with support from the National Science Foundation under NSF Award Numbers 1833703, 1833643, and 1833632. We also acknowledge NASA Grant NNX13AP69G for the collection of SONOMA COUNTY dataset, NSF Award number 1043051 for the collection of BIG CREEK, CANYON LAKE GORGE, and GREAT SMOKEY MOUNTAIN datasets.

References

1. Agarwal, P.K., Beutel, A., Mø lhave, T.: TerraNNI: Natural Neighbor Interpolation on 2D and 3D Grids Using a GPU. ACM Transactions on Spatial Algorithms and Systems **2**(2), 1–31 (2016). DOI 10.1145/2786757
2. Allili, M., Kaczynski, T., Landi, C., Masoni, F.: A new matching algorithm for multidimensional persistence. ArXiv, Id:1511.05427 (2015). URL `http://arxiv.org/abs/1511.05427`
3. Baker, W.E., Emmitt, G.D., Robertson, F., Atlas, R.M., Molinari, J.E., Bowdle, D.A., Paegle, J., Hardesty, R.M., Menzies, R.T., Krishnamurti, T., et al.: Lidar-measured winds from space: a key component for weather and climate prediction. Bulletin of the American Meteorological Society **76**(6), 869–888 (1995)
4. Banchoff, T.F.: Critical Points and Curvature for Embedded Polyhedral Surfaces. The American Mathematical Monthly **77**(5), 475–485 (1970). URL `http://www.jstor.org/stable/2317380`
5. Bentley, J.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)
6. Boissonnat, J.D., Cazals, F.: Smooth surface reconstruction via natural neighbour interpolation of distance functions. Computational Geometry **22**(1-3), 185–203 (2002). DOI 10.1016/S0925-7721(01)00048-7
7. Carr, H., Duke, D.: Joint contour nets: Computation and properties. In: Visualization Symposium (PacificVis), 2013 IEEE Pacific, pp. 161–168 (2013). DOI 10.1109/PacificVis.2013.6596141
8. Carr, H., Geng, Z., Tierny, J., Chattopadhyay, A., Knoll, A.: Fiber Surfaces: Generalizing Isosurfaces to Bivariate Data. Computer Graphics Forum **34**(3), 241–250 (2015). DOI 10.1111/cgf.12636. URL `http://onlinelibrary.wiley.com/doi/10.1111/cgf.12636/full`
9. Computational Geometry Algorithms Library (CGAL) (2020). `https://www.cgal.org/` [Online; accessed February-2020]
10. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R.: BDAM – Batched Dynamic Adaptive Meshes for high performance terrain visualization. Computer Graphics Forum **22**(3), 505–514 (2003)
11. Clarke, L., Glendinning, I., Hempel, R.: The MPI Message Passing Interface standard. In: K.M. Decker, R.M. Rehmann (eds.) Programming Environments for Massively Parallel Distributed Systems, pp. 213–218. Birkhäuser Basel, Basel (1994)
12. Dalponte, M., Bruzzone, L., Gianelle, D.: Fusion of hyperspectral and lidar remote sensing data for classification of complex forest areas. IEEE Transactions on Geoscience and Remote Sensing **46**(5), 1416–1427 (2008)
13. De Floriani, L., Dimitri, D., Facinoli, M., Magillo, P.: The PM_2-Triangle quadtree. Tech. rep., Dipartimento di Informatica e Scienze dell'Informazione (DISI), Università degli Studi di Genova (2007)
14. De Floriani, L., Facinoli, M., Magillo, P., Dimitri, D.: A hierarchical spatial index for triangulated surfaces. In: Proceedings of the Third International Conference on Computer Graphics Theory and Applications (GRAPP 2008), pp. 86–91 (2008)
15. De Floriani, L., Fugacci, U., Iuricich, F., Magillo, P.: Morse complexes for shape segmentation and homological analysis: discrete models and algorithms. In: Computer Graphics Forum, vol. 34, pp. 761–785. Blackwell Publishing Ltd (2015). DOI 10.1111/cgf.12596
16. De Floriani, L., Hui, A.: Data structures for simplicial complexes: An analysis and a comparison. In: Proceedings of the third Eurographics symposium on Geometry processing, pp. 119–es. Eurographics Association (2005)

17. Dolan, M.F., Grehan, A.J., Guinan, J.C., Brown, C.: Modelling the local distribution of cold-water corals in relation to bathymetric variables: Adding spatial context to deep-sea video data. Deep Sea Research Part I: Oceanographic Research Papers **55**(11), 1564–1579 (2008)
18. Edelsbrunner, H., Harer, J.: Jacobi sets of multiple Morse functions. In: Foundations of Computational Mathematics, Minneapolis 2002, *London Mathematical Society Lecture Note Series*, vol. 312, pp. 35–57. Cambridge University Press (2004). DOI 10.1017/CBO9781139106962.003. URL http://dx.doi.org/10.1017/CBO9781139106962.003
19. Edelsbrunner, H., Harer, J.: Jacobi Sets of multiple Morse functions. In: Foundations of Computational Mathematics, *London Mathematical Society Lecture Note Series*, vol. 312, pp. 37–57. Cambridge University Press (2004). DOI 10.1017/CBO9781139106962.003
20. Edelsbrunner, H., Harer, J., Patel, A.K.: Reeb spaces of piecewise linear mappings. In: Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry, SoCG '08, pp. 242–250. ACM, New York, NY, USA (2008). DOI 10.1145/1377676.1377720
21. Edelsbrunner, H., Mücke, E.P.: Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. ACM Transactions on Graphics **9**(1), 66–104 (1990). DOI 10.1145/77635.77639
22. Fellegara, R.: Spatial indexes for simplicial and cellular meshes. In: New Trends in Databases and Information Systems, pp. 373–382. Springer International Publishing (2014)
23. Fellegara, R., Iuricich, F., De Floriani, L.: Efficient representation and analysis of triangulated terrains. In: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL'17, pp. 74:1–74:4. ACM, New York, NY, USA (2017). DOI 10.1145/3139958.3140050. URL http://doi.acm.org/10.1145/3139958.3140050
24. Fellegara, R., Iuricich, F., De Floriani, L., Weiss, K.: Efficient computation and simplification of discrete Morse decompositions on triangulated terrains. In: Proceedings of the 22th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM (2014)
25. Fellegara, R., Song, Y.: LibTri code repository. https://github.com/UMDGeoVis/Terrain_Analysis_on_IA (2021)
26. Fellegara, R., Song, Y.: Terrain trees library code repository. https://github.com/UMDGeoVis/Terrain_Trees (2021)
27. Fellegara, R., Weiss, K., De Floriani, L.: The Stellar decomposition: A compact representation for simplicial complexes and beyond. Computers & Graphics **98**, 322–343 (2021). DOI 10.1016/j.cag.2021.05.002
28. Finkel, R., Bentley, J.: Quad trees a data structure for retrieval on composite keys. Acta informatica **4**(1), 1–9 (1974)
29. Forman, R.: Morse theory for cell complexes. Advances in Mathematics **134**, 90–145 (1998)
30. Forman, R.: A user's guide to discrete Morse theory. Sém. Lothar. Combin **48**, B48c (2002)
31. Gobbetti, E., Marton, F., Cignoni, P., Di Benedetto, M., Ganovelli, F.: C-bdam–compressed batched dynamic adaptive meshes for terrain rendering. In: Computer Graphics Forum, vol. 25, pp. 333–342. Wiley Online Library (2006)
32. Gurung, T., Rossignac, J.: SOT: A compact representation for tetrahedral meshes. In: Proceedings SIAM/ACM Geometric and Physical Modeling, SPM '09, pp. 79–88. San Francisco, USA (2009). DOI 10.1145/1629255.1629266
33. Heine, C., Leitte, H., Hlawitschka, M., Iuricich, F., De Floriani, L., Scheuermann, G., Hagen, H., Garth, C.: A Survey of Topology-based Methods in Visualization. Computer Graphics Forum **35**(3), 643–667 (2016). DOI 10.1111/cgf.12933. URL http://doi.wiley.com/10.1111/cgf.12933
34. Held, G., Marshall, T.: Data compression; techniques and applications: Hardware and software considerations. John Wiley & Sons (1991)
35. Hjaltason, G., Samet, H.: Speeding up construction of PMR quadtree-based spatial indexes. The VLDB Journal, The International Journal on Very Large Data Bases **11**(2), 137 (2002)
36. Huettenberger, L., Heine, C., Garth, C.: Decomposition and Simplification of Multivariate Data using Pareto Sets. **20**(12), 2684–93 (2014). DOI 10.1109/TVCG.2014.2346447
37. Hyde, P., Dubayah, R., Peterson, B., Blair, J., Hofton, M., Hunsaker, C., Knox, R., Walker, W.: Mapping forest structure for wildlife habitat analysis using waveform lidar: Validation of montane ecosystems. Remote sensing of environment **96**(3-4), 427–437 (2005)
38. Iuricich, F., De Floriani, L.: Hierarchical forman triangulation: A multiscale model for scalar field analysis. Computers & Graphics (2017). DOI https://doi.org/10.1016/j.cag.2017.05.015
39. Iuricich, F., Scaramuccia, S., Landi, C., De Floriani, L.: A discrete morse-based approach to multivariate data analysis. In: SIGGRAPH ASIA 2016 Symposium on Visualization on - SA '16, SA '16, pp. 1–8. ACM, New York, NY, USA (2016). DOI 10.1145/3002151.3002166. URL http://dl.acm.org/citation.cfm?doid=3002151.3002166
40. J de Smith, M., Goodchild, M., Longley, P.: Geospatial Analysis: A Comprehensive Guide. The Winchelsea Press, S.l. (2018)

41. King, H.C., Knudson, K., Neza, M.: Generating discrete Morse functions from point data. Experimental Mathematics **14**(4), 435–444 (2005). DOI 10.1080/10586458.2005.10128941
42. Lanier, A., Romsos, C., Goldfinger, C.: Seafloor habitat mapping on the oregon continental margin: A spatially nested gis approach to mapping scale, mapping methods, and accuracy quantification. Marine Geodesy **30**(1-2), 51–76 (2007)
43. Lee, S., Har, D., Kum, D.: Drone-assisted disaster management: Finding victims via infrared camera and lidar sensor fusion. In: 2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE), pp. 84–89. IEEE (2016)
44. Lewiner, T.: Critical sets in discrete Morse theories: Relating Forman and piecewise-linear approaches. Computer Aided Geometric Design **30**(6), 609–621 (2013). DOI 10.1016/j.cagd.2012.03.012
45. Lindenbaum, M., Samet, H., Hjaltason, G.R.: A probabilistic analysis of trie-based sorting of large collections of line segments in spatial databases. SIAM Journal on Computing **35**(1), 22–58 (2005)
46. Mancinelli, C., Livesu, M., Puppo, E.: Gradient Field Estimation on Triangle Meshes. In: M. Livesu, G. Pintore, A. Signoroni (eds.) Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference. The Eurographics Association (2018). DOI 10.2312/stag.20181301
47. Meigs, A.: Active tectonics and the lidar revolution. Lithosphere **5**(2), 226–229 (2013)
48. Mesmoudi, M., De Floriani, L., Magillo, P.: Morphological analysis of terrains based on discrete curvature and distortion. In: W. Aref, M. Mokbel, H. Samet, M. Schneider, C. Shahabi, O. Wolfson (eds.) Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems, pp. 415–418. Irvine, CA, USA (2008). DOI 10.1145/1463434.1463498
49. Mesmoudi, M., De Floriani, L., Port, U.: Discrete distortion in triangulated 3-manifolds. Computer Graphics Forum **27**(5), 1333–1340 (2008). DOI 10.1111/j.1467-8659.2008.01272.x
50. Mesmoudi, M.M., De Floriani, L., Magillo, P.: Discrete distortion for surface meshes. In: International Conference on Image Analysis and Processing, pp. 652–661. Springer (2009)
51. Meyer, M., Desbrun, M., Schroder, M., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: H.C. Hege, K. Polthier (eds.) Proceedings VisMath 2002, pp. 35–57 (2003)
52. Milnor, J.: Morse Theory. Princeton University Press, New Jersey (1963)
53. Nagaraj, S., Natarajan, V., Nanjundiah, R.S.: A Gradient-Based Comparison Measure for Visual analysis of Multifield Data **30**(3), 1101–1110 (2011). DOI 10.1111/j.1467-8659.2011.01959.x
54. Nelson, R., Samet, H.: A consistent hierarchical representation for vector data. ACM SIGGRAPH Computer Graphics **20**(4), 197–206 (1986)
55. Nielson, G.M.: Tools for triangulations and tetrahedralizations and constructing functions defined over them. In: G.M. Nielson, H. Hagen, H. Müller (eds.) Scientific Visualization: overviews, Methodologies and Techniques, chap. 20, pp. 429–525. IEEE Computer Society, Silver Spring, MD (1997)
56. Opentopography - high-resolution topography data and tools (2020). `http://www.opentopography.org/` [Online; accessed February-2020]
57. Orenstein, J.A.: Multidimensional tries used for associative searching. Information Processing Letters **14**(4), 150–157 (1982)
58. Pajarola, R., Gobbetti, E.: Survey of semi-regular multiresolution models for interactive terrain rendering. The Visual Computer **23**(8), 583–605 (2007)
59. Paoluzzi, A., Bernardini, F., Cattani, C., Ferrucci, V.: Dimension-independent modeling with simplicial complexes. ACM Transactions on Graphics (TOG) **12**(1), 56–102 (1993)
60. Robins, V., Wood, P., Sheppard, A.: Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(8), 1646–1658 (2011). DOI 10.1109/TPAMI.2011.95
61. Rossignac, J., Safonova, A., Szymczak, A.: 3D compression made simple: Edge-Breaker on a Corner Table. In: Proceedings Shape Modeling International 2001. IEEE Computer Society, Genova, Italy (2001)
62. Samet, H.: The Design and analysis of spatial data structure. Addison Wesley (1990)
63. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann (2006)
64. Samet, H., Webber, R.: Storing a collection of polygons using quadtrees. ACM Transactions on Graphics (TOG) **4**(3), 182–222 (1985)
65. Saye, S., van der Wal, D., Pye, K., Blott, S.: Beachdune morphological relationships and erosion/accretion: An investigation at five sites in england and wales using lidar data. Geomorphology **72**(1), 128–155 (2005). DOI https://doi.org/10.1016/j.geomorph.2005.05.007. URL `https://www.sciencedirect.com/science/article/pii/S0169555X05001698`
66. Shepard, M.K., Campbell, B.A., Bulmer, M.H., Farr, T.G., Gaddis, L.R., Plaut, J.J.: The roughness of natural terrain: A planetary and remote sensing perspective. Journal of Geophysical Research: Planets **106**(E12), 32777–32795 (2001)
67. Shivashankar, N., Senthilnathan, M., Natarajan, V.: Parallel computation of 2D Morse-Smale complexes. IEEE Transactions on Visualization and Computer Graphics **18**(10), 1757–1770 (2012). DOI 10.1109/TVCG.2011.284

68. Tierny, J., Carr, H.: Jacobi Fiber Surfaces for Bivariate Reeb Space Computation **23**(1), 960–969 (2017). DOI 10.1109/TVCG.2016.2599017
69. Wang, R., Peethambaran, J., Chen, D.: Lidar point clouds to 3-d urban models : a review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **11**(2), 606–627 (2018)
70. Warner, F.W.: Foundations of differentiable manifolds and Lie groups, vol. 94. Springer Science & Business Media (1983)
71. Webster, T.L., Forbes, D.L., Dickie, S., Shreenan, R.: Using topographic lidar to map flood risk from storm-surge events for charlottetown, prince edward island, canada. Canadian Journal of Remote Sensing **30**(1), 64–76 (2004)
72. Weiss, K., Iuricich, F., Fellegara, R., De Floriani, L.: A primal/dual representation for discrete Morse complexes on tetrahedral meshes. In: Computer Graphics Forum, vol. 32, pp. 361–370 (2013)
73. White, S.A., Wang, Y.: Utilizing dems derived from lidar data to analyze morphologic change in the north carolina coastline. Remote sensing of environment **85**(1), 39–47 (2003)
74. Xu, X., Iuricich, F., De Floriani, L.: A Persistence-Based Approach for Individual Tree Mapping. In: Proceedings of the 28th International Conference on Advances in Geographic Information Systems, pp. 191–194. ACM (2020). DOI 10.1145/3397536.3422231
75. Yonglin, S., Lixin, W., Zhi, W.: Identification of inclined buildings from aerial lidar data for disaster management. In: 2010 18th International Conference on Geoinformatics, pp. 1–5. IEEE (2010)
76. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. In: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, pp. 10–10. USENIX Association, Berkeley, CA, USA (2010). URL `http://dl.acm.org/citation.cfm?id=1863103.1863113`

A Appendix

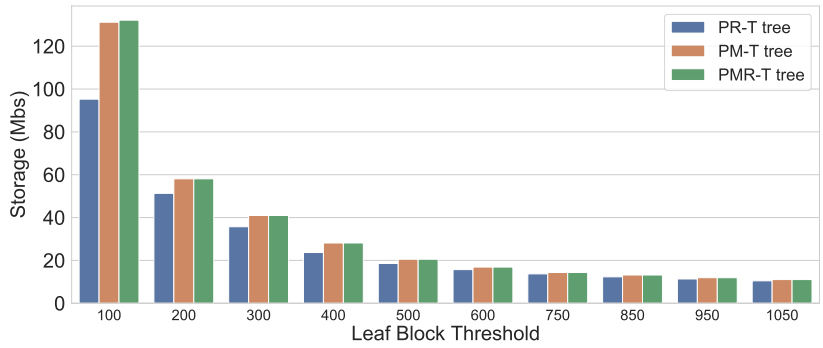


Fig. 20 The storage costs for storing the hierarchical index of the Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.

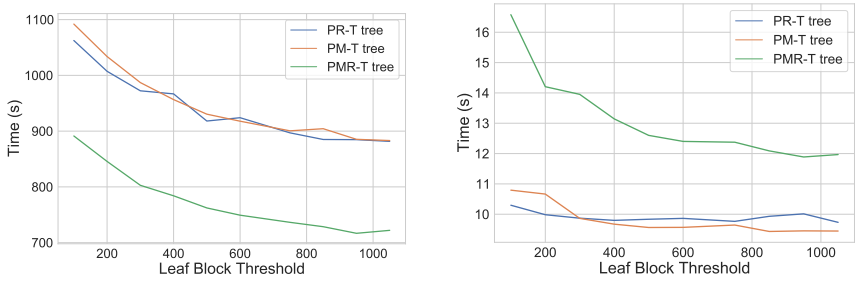


Fig. 21 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on CANYON LAKE GORGE dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

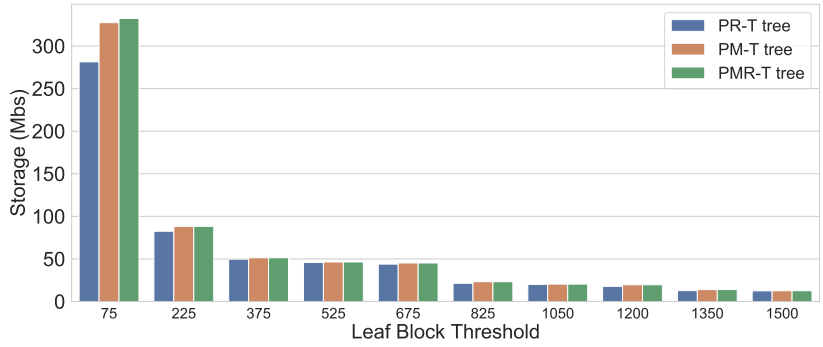


Fig. 22 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.

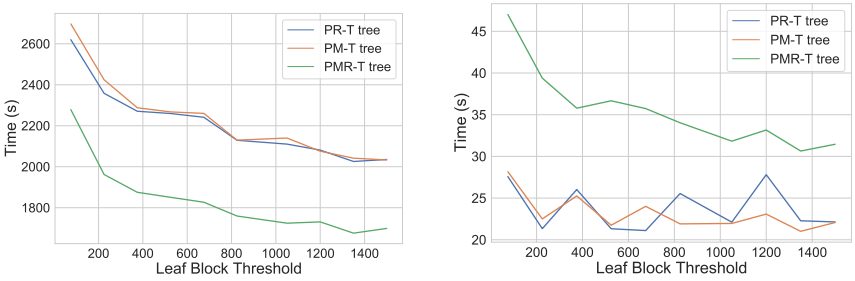


Fig. 23 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 1 dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

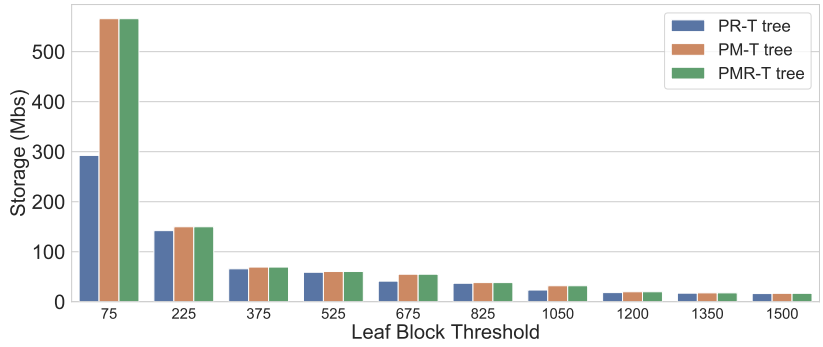
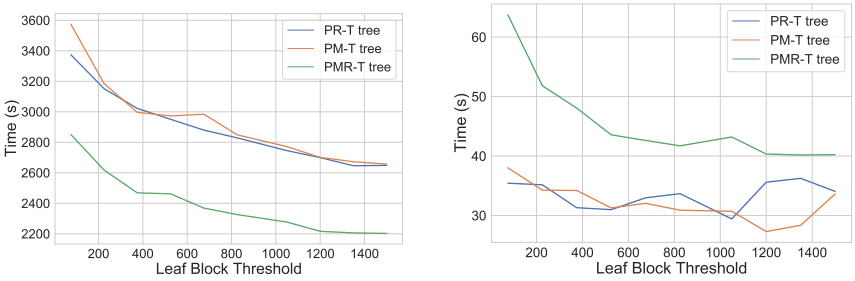


Fig. 24 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.



(a) Generation time (b) VT extraction time

Fig. 25 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 2 dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

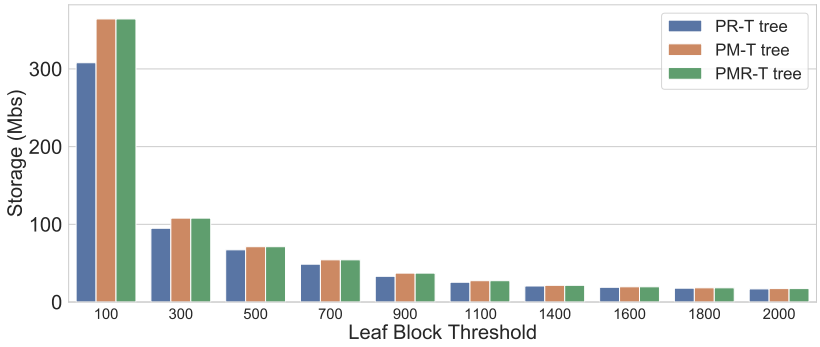
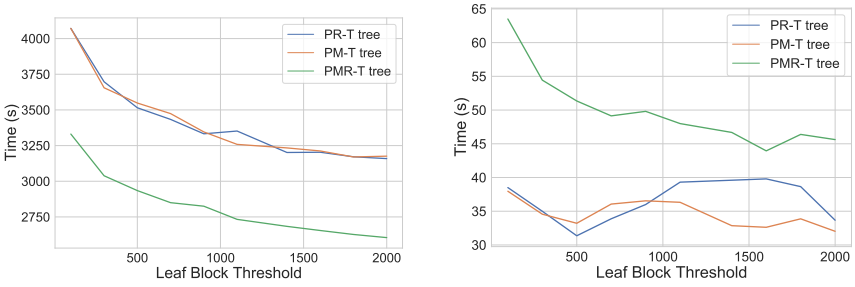


Fig. 26 The storage costs for storing the hierarchical index of the Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.



(a) Generation time (b) VT extraction time

Fig. 27 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on BIG CREEK dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

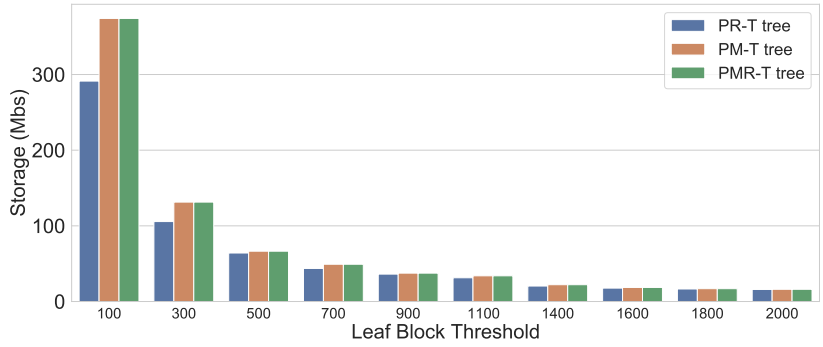


Fig. 28 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.

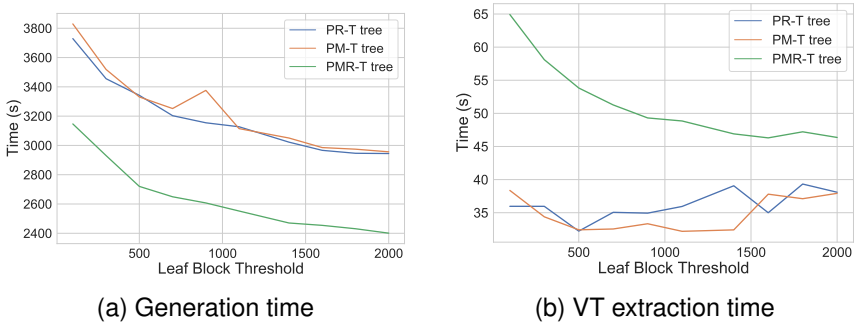


Fig. 29 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 3 dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.

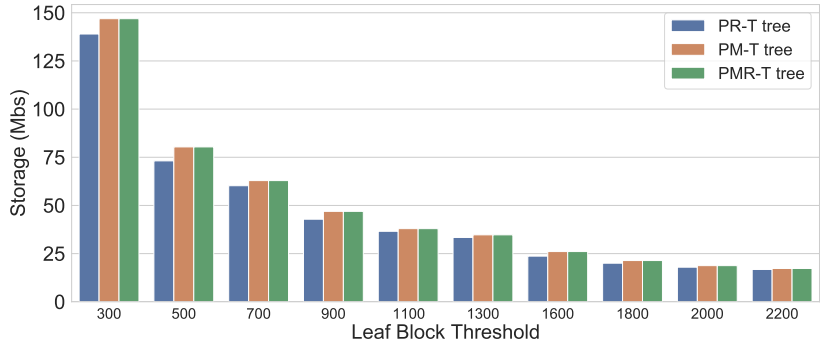
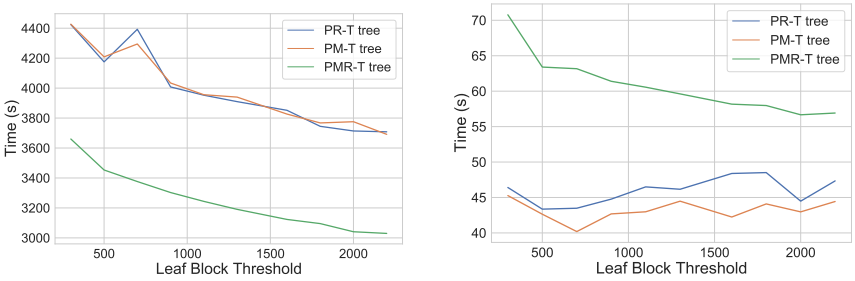


Fig. 30 The storage costs for storing the hierarchical index of the Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the threshold value on the vertices.



(a) Generation time (b) VT extraction time

Fig. 31 Timings for generating the spatial decomposition of Terrain trees and extracting the *VT* relation in Terrain trees on SONOMA COUNTY 4 dataset using different values of k_v and k_t. The x-axis shows the vertex threshold value.