Computers & Graphics 58 (2016) 43–52

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Issue on SMI 2016

Computing a discrete Morse gradient from a watershed decomposition

Lidija Čomić [a], Leila De Floriani [c], Federico Iuricich [b,*], Paola Magillo [c]

[a] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
[b] Department of Computer Science and UMIACS, University of Maryland, College Park, MD, USA
[c] Department of Computer Science, Bioengineering, Robotics, and Systems Engineering, University of Genova, Genova, Italy

ARTICLE INFO

Article history:
Received 14 March 2016
Received in revised form
12 May 2016
Accepted 13 May 2016
Available online 18 May 2016

Keywords:
Discrete Morse theory
Watershed transform
Morse–Smale complexes

ABSTRACT

We consider the problem of segmenting triangle meshes endowed with a discrete scalar function f based on the critical points of f. The watershed transform induces a decomposition of the domain of function f into regions of influence of its minima, called catchment basins. The discrete Morse gradient induced by f allows recovering not only catchment basins but also a complete topological characterization of the function and of the shape on which it is defined through a Morse decomposition. Unfortunately, discrete Morse theory and related algorithms assume that the input scalar function has no flat areas, whereas such areas are common in real data and are easily handled by watershed algorithms. We propose here a new approach for building a discrete Morse gradient on a triangulated 3D shape endowed by a scalar function starting from the decomposition of the shape induced by the watershed transform. This allows for treating flat areas without adding noise to the data. Experimental results show that our approach has significant advantages over existing ones, which eliminate noise through perturbation: it is faster and always precise in extracting the correct number of critical elements.

1. Introduction

Forman's discrete Morse theory [1] is a combinatorial counterpart of Morse theory [2]. Used in shape analysis and understanding, it has been adopted for shape segmentation, and homology and persistent homology computation [3]. In particular, discrete Morse theory is the basis for an efficient and derivative-free computation of a segmentation of a discretized shape endowed with a scalar function f. The discrete Morse gradient fully encodes the topological structure of the function and of its domain and the regions of influence of the critical points. Also, the critical net or the Morse–Smale complex can be efficiently extracted from it, but the whole theory is developed under the assumption that no flat areas are present in the input data. On the other hand, flat areas are very common in real-world datasets. They can be intrinsic to a shape, like in terrains representing lakes, isolines or other real flat features, or they can be due to the limited precision of acquisition devices. Currently, most of the approaches based on discrete Morse theory adopt the idea of Simulation of Simplicity (SoS) [4] for eliminating flat areas (also called plateaus) by introducing noise into them. Both when we are analyzing the morphology of a scalar field, or when we are studying the persistent homology of a shape a fundamental step is computing persistence pairs [5]. Using discrete Morse theory, the computational complexity of this operation depends on the number of critical simplexes present in the data. The pipeline resulting from using SoS is then conceptually misleading. Introducing a number of spurious critical simplexes we increase the computational complexity for computing the persistence pairs and we successively remove the spurious critical simplexes before studying the actual results.

We propose here a new algorithm that creates a discrete Morse gradient starting from a watershed decomposition of a triangulated 2D shape in $\mathbb{R}^3$ endowed with a scalar function. It permits to construct a discrete Morse gradient from real data containing flat areas without perturbing the original function. To do so we first segment our domain using an existing watershed approach, subdividing the vertices of the input triangulated surface Σ into regions of influence of the minima of the scalar function f defined on Σ. Then, our algorithm proceeds in three steps. First, all the vertices in Σ, with the exception of the minima of f, are paired with some incident edge. This pairing is induced by the used watershed algorithm. Second, starting from the boundary of each basin, triangles that are contained in the basin are paired with edges. Third, the gradient obtained so far is modified to introduce critical triangles corresponding to maxima of f and finally, edges and triangles between basins are paired to produce the final discrete Morse gradient. The discrete Morse gradient extends the watershed decomposition in the sense that, by navigating the gradient, we obtain the same regions of influence of the minima as

* Correspondence to: University of Maryland, Department of Computer Science and UMIACS, A.V. Williams Building, College Park (MD) 20740, United States.

http://dx.doi.org/10.1016/j.cag.2016.05.020
0097-8493/

the watershed decomposition. At the same time, it gives a much more powerful topological description of the function and its domain since from a discrete Morse gradient all the Morse features, including the critical net and the Morse–Smale complex, can be computed. The fact that the discrete Morse theory subsumes the watershed approach was experimentally shown in [6] by comparing their results on input data satisfying the requirement of having no flat features. Here, we go further and propose a method to complete the output of a watershed algorithm to a discrete Morse gradient. The main contributions of our work are:

- a new algorithm for computing a discrete Morse gradient on 2D scalar fields with flat areas,
- a constructive proof of the equivalence of the techniques based on discrete Morse theory and watershed,
- experiments showing that our algorithm is faster in computing the Forman gradient compared to the state of the art techniques, while it preserves the flat areas.

The remainder of the paper is organized as follows. In Section 2, we review some background notions on discrete Morse theory and on the watershed transform. In Section 3, we discuss some related work on these subjects. In Sections 4, 5, 6 and 7, we give an overview of our algorithm and we describe and prove the correctness of its three steps. Experimental results are presented in Section 8, and concluding remarks are drawn in Section 9.

2. Background notions

We recall here some basic notions on discrete Morse theory [1] and on the watershed transform [7,8], both in the framework of triangle meshes. A triangle mesh Σ is a set of triangles such that, for any pair of distinct triangles σ_1 and $\sigma_2 \in \Sigma$, either σ_1 and σ_2 are disjoint, or their intersection is a common edge or a common vertex. Vertices and edges of the triangles in Σ are also considered as belonging to Σ. Triangles, edges and vertices are also simplexes of dimensions 2, 1, and 0, respectively. A *k-dimensional simplex*, or *k-simplex*, σ is the convex hull of $k+1$ affinely independent points. The convex hull of any subset of the vertices of σ defines a simplex γ which is called a *face* of σ, while σ is a *coface* of γ. Thus, the faces of a triangle are its three edges and vertices, and the faces of an edge are its two extreme vertices. The *star* of a k-simplex σ in Σ is the collection of all the cofaces of σ in Σ.

2.1. Discrete Morse theory

Discrete Morse theory [1] is defined for simplicial and cell complexes, but here we introduce it for triangle meshes, which are a special case of simplicial complexes. Discrete Morse theory [1] considers functions defined over all the simplexes (vertices, edges, triangles) of a triangle mesh Σ. A function F defined over all the simplexes of Σ is called a *discrete Morse function*, or a *Forman function* if, for any k-simplex σ, all the $(k-1)$-faces of σ have a lower F value than σ, and all the $(k+1)$-cofaces have a higher F value than σ, with at most one exception. A simplex is *critical* if there is no such exception. Fig. 1a shows an example of a discrete Morse function F. Vertex 0 is critical (minimum), since F has a higher value on all edges incident to it. Triangle 9 is critical (maximum), since F has a lower value on all edges incident to it. Edge 8 is critical (saddle), since F has a higher value on the incident triangle 9, and lower values on its extreme vertices.

The unique exception to the above rule, which holds for a non-critical simplex, permits to pair such simplex with either one of its faces, or one of its cofaces. Thus, a discrete Morse function F on a triangle mesh Σ induces a collection of pairs (σ, τ), where σ is an edge and τ is a triangle, or σ is a vertex and τ is an edge, with σ face of τ. Such pair can be depicted as an arrow going from σ (tail) to τ (head), as shown in Fig. 1b. An alternative way to introduce discrete Morse theory is through the notion of Forman gradient.

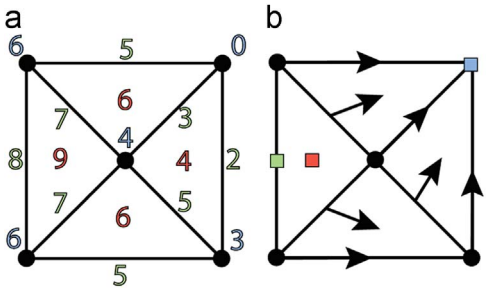

Fig. 1. (a) A discrete Morse function on a triangle mesh and (b) the corresponding discrete Morse gradient. Each simplex is labeled by its function value.

A collection of pairs (σ, τ) defines a *discrete vector field* V when each simplex of Σ is in at most one pair of V [1]. A *V-path* is a sequence $\sigma_1, \tau_1, \sigma_2, \tau_2, \ldots, \sigma_r, \tau_r$ of k-simplexes σ_i and $(k+1)$-simplexes τ_i, $i = 1, \ldots, r$ with $r \geq 1$, such that $(\sigma_i, \tau_i) \in V$, σ_{i+1} is a face of τ_i, and $\sigma_i \neq \sigma_{i+1}$. In a triangle mesh, V-paths involve either vertices and edges, or edges and triangles. A V-path with $r > 1$ is *closed* if σ_1 is a face of τ_r different from σ_{r-1}. A discrete vector field V is called a *discrete Morse gradient vector field* (or a *Forman gradient*) if and only if there are no closed V-paths in V. A *critical simplex* of V of index k is a k-simplex γ which does not appear in any pair of V.

There is a correspondence between discrete Morse functions and discrete Morse gradients [9]. Namely, for each such function F, a discrete Morse gradient V_F can be constructed. Conversely, for each discrete Morse gradient V there exists a (non-unique) Forman function F such that the gradient of F is V. The (negative) discrete Morse gradient V_F of F at a vertex (edge) σ indicates the direction of a unique coface edge (triangle) τ of σ, in which F is decreasing. Fig. 1b shows the discrete Morse gradient V_F corresponding to the Forman function F in Fig. 1a.

As noted in [1,10], Forman functions are hard to find, and are unintuitive to work with. Thus, almost all the algorithms available in the literature focus on computing a discrete Morse gradient.

2.2. The watershed transform

The watershed transform has been first defined for grey-scale images [7,11–14], and has subsequently been extended to triangle meshes [15]. It is defined on an undirected labeled graph $H = (N_H, A_H, f)$, where the nodes in N_H, labeled by function f, represent the pixels in an image, or the vertices of a triangle mesh, and the arcs in A_H represent the edge-adjacencies between pairs of pixels or the edges in a triangle mesh. The notion of a *discrete topographic distance* is given in terms of minimum-cost paths in H [7]. The *lower slope* $LS(p)$ at a node p is the maximal slope linking p to any of its neighbors of lower function value:

$$LS(p) = \max\left\{\frac{f(p)-f(q)}{dist(p,q)} \mid (p,q) \in A_H, f(q) < f(p)\right\}$$

where distance $dist(p,q)$ is computed in domain space (i.e., on the 2D plane in case of an image). If no neighbor q exists with $f(q) < f(p)$, then $LS(p) = 0$. A cost is associated with the arcs in A_H defined in terms of the lower slope. The *π-topographic distance* between nodes p and q is the sum of costs of all directed arcs in path π connecting p and q. The *topographic distance* $T(p,q)$ between p and q is the minimum of the π-topographic distances along all such paths π. The *catchment basin* of a minimum m of f is the set of nodes closer to m than to any other minimum of f. Note

that the topographic distance is actually a pseudo-distance, because it is equal to zero on distinct nodes in the same flat area.

3. Related work

We provide here an overview of algorithms for extracting discrete Morse gradients, and catchment basins based on the watershed transform.

3.1. Algorithms for discrete Morse gradient

Algorithms based on discrete Morse theory are purely combinatorial, dimension-independent and independent of the type of the discretization of the underlying shape (domain of f). Algorithms [16–19] are easily parallelizable or have been specifically developed for distributed computation. Starting from a discrete scalar function f defined over the vertices of a complex Σ, they aim at constructing a discrete Morse gradient that best fits function f. They focus on extracting a minimum number of critical simplexes [16,20], or they perform a posteriori simplifications to reduce their number [21,22]. The typical applications for such algorithms are data analysis and visualization, since a discrete Morse gradient provides a computationally efficient way for extracting the regions of influence of the critical points.

The algorithm presented in [16] has been recognized as one of the most powerful. It has been adapted to triangle and tetrahedral meshes in [19,23] and an optimized version for computing persistent homology on volumetric images has been developed in [24]. A function value is defined for each simplex by listing the field values on its vertices in lexicographic order. The algorithm processes the lower star of each vertex v in Σ independently, where the *lower star* of a simplex σ is the subset of the star of σ containing only simplexes with a lower function value than σ. Simplexes are considered in ascending order of function value and of dimension in such a way that each simplex is considered after its faces. The simplexes in the lower star of the current simplex are paired via homotopy expansion. Two simplexes, k-simplex σ and $(k+1)$-simplex τ, are paired when σ has no unpaired coface and τ has only one unpaired face (i.e., σ). In [16] it has been proven that up to the 3D case, the critical cells identified are in one-to-one correspondence with the topological changes in the sub-level sets of the scalar function defined over the complex.

Based on the latter, a new algorithm has been defined in [25] for computing a discrete Morse gradient based on an input segmentation. Similar to our proposal, the input segmentation is used for guiding the gradient computation. However, in [25] the segmentation has the purpose of limiting the number of gradient paths between adjacent regions, while in our case, as we describe in Section 4, the segmentation helps us to reconstruct the correct number of critical simplexes.

3.2. Watershed algorithms

Several algorithms have been proposed in the literature for performing the watershed transform, which are based on the *discrete topographic distance* [7], on *simulated immersion* [11,12], or on *rain falling simulation* [15,14].

The approach based on topographic distance directly applies the definition of catchment basins. The *image integration* algorithm by Meyer [26,7] is a variation of the Dijkstra–Moore algorithm [27] for computing the shortest path, in terms of topographic distance, from a source node to every other node in a graph. The *hill climbing* algorithm [7] is a simplified version of the image integration approach, which applies to regular grids, since the distance between two adjacent nodes p and q in domain space is assumed to be constant.

The intuitive idea behind the *simulated immersion* approach [11,12] is that of letting water raise from minima. When applied to the graph H describing the image or the triangle mesh, the algorithm expands catchment basins by processing the nodes of H by increasing function values. When a certain level h is reached, all catchment basins of minima with value $h' < h$ have been started and contain just nodes with function values lower than h. Processing level h will add new nodes to existing basins, and will start new basins from minima having a function value equal to h. The expansion of existing basins treats flat areas in a transparent way.

The watershed approaches discussed so far have in common the idea of growing catchment basins upwards from the minima of f. The *rain falling* paradigm [15,14] uses the idea of letting water fall down from each vertex until it reaches a minimum. The algorithm descends from each node to its lowest adjacent node until a minimum is reached. The algorithm in [15] is for triangle meshes, while the one in [14] is for regular grids. Both of them allow for the treatment of flat areas. In [15], all such areas are found in a preliminary step and each of them is treated as a single node. In [14], flat areas are found as they are encountered during the descent. An implementation of the rain falling simulation for triangle meshes has been used in [28].

In the output of watershed algorithms, not all nodes belong to a catchment basin. In the approaches based on topographic distance and on simulated immersion, nodes that have the same distance from different minima, or are reached by water from different minima, are labeled as belonging to watershed lines (watershed nodes).

3.3. Removal of flat areas

Simulation of Simplicity (SoS) [4] is the most widely used method to resolve differentiability in topological analysis in the discrete case. Given a complex with flat areas, a strictly increasing ordering of vertices is defined. The new indexes preserve the old order (induced by function values) for distinct value data and impose a new ordering for vertices having the same value. The main drawback of SoS is that it introduces spurious critical points that are not present in the original data. To mitigate this problem an improved version of the SoS approach has been introduced in [22]. Two sorted queues are used for imposing a breadth first visit of the flat areas. In this manner, no spurious minima are created but saddles and maxima are still introduced without control. The algorithm presented in [28] is the only one, to the best of our knowledge, that address the problem of perturbing data in a such a way that the modified scalar field does not have missing or extra critical points. This algorithm works on triangle meshes only. It iteratively addresses vertices on the boundary of a flat area and slightly raises or lowers them (according to specific rules), in order to progressively "erode" each plateau without creating new maxima or minima, or deleting existing ones. The method has to use a priority queue of candidate vertices for the various rules, as each modified vertex creates new candidates. Moreover, in order to avoid machine precision errors, it applies a symbolic modification of height values followed by a final rescaling of all the vertices. For such reasons, it is rather complicated and intrinsically slower than using perturbation.

4. Computing the discrete Morse gradient

In this section, we describe our approach, that combines watershed transform and discrete Morse theory, for computing a Forman gradient from a scalar field. The input of our algorithm is a

triangle mesh Σ endowed with a scalar function f at its vertices, together with a partition of the vertices of Σ, produced by a watershed algorithm (see Section 3). In our implementation we have used simulated immersion, but any other watershed algorithm can be used as well.

We apply the watershed algorithm to the graph H formed by the vertices and edges of Σ, where each node of H, which is a vertex of Σ, is labeled with the corresponding value of f. Recall that the watershed algorithm applied to H labels the nodes of H as belonging to catchment basins or to watershed lines. We remove watershed nodes by assigning them to a basin in the following way. A queue of watershed nodes having a lower adjacent node assigned to some basin is built. At each step, a watershed node v is extracted from the queue, labeled as belonging to the same basin as its neighbor with lowest function value, and watershed nodes adjacent to v are inserted into the queue. The process continues until the queue is emptied. The result of this preprocessing is a partition of the vertices of Σ into catchments basins.

The discrete Morse gradient V induced by the watershed algorithm is built in three steps:

1. *Vertex–edge pairing*: A discrete Morse gradient V' is constructed having only vertex–edge pairs, such that each vertex, which is not a minimum, is paired in V'.
2. *Edge–triangle pairing inside basins*: V' is extended to V'' by adding edge–triangle pairs in such a way that each triangle lying completely inside some basin (i.e., such that all three vertices belong to the basin) is paired (see Fig. 2).
3. *Edge–triangle pairing between adjacent basins*: The final discrete Morse gradient V is built from V'' by determining critical triangles and pairing the remaining edges and (non-critical) triangles whose vertices belong to different basins.

In the following three sections, we describe the three steps in detail, and prove the correctness of each step, and thus of the whole algorithm. We show that the pairing of cells established by our algorithm defines a discrete Morse gradient V on Σ such that there is a one-to-one correspondence between the critical points of the scalar function f and the critical simplexes of V.

Algorithm 1. vertexEdgeInsideFlat(Σ, f, V).

```
1: Input: Σ, triangle mesh
2: Input: f, scalar function
3: Input: W, watershed segmentation
4: Input/Output: V, Forman gradient
5:
6: for each vertex v in Σ do
7:   if !isPaired(v,V) AND !is-Critical(v,V) then
8:     K←collectPairedInsideFlatArea(v, V)
9:     if K = ∅ then
10:        setCritical(v,V) // v is a minimum
11:        K←adjacentsUnpaired(v, Σ, V)
12:        for each u in K do
13:            if (v) = f(u) then
14:              setPair(u, uv, V)
15:    Q←K // Q is queue
16:    while !Q.isEmpty() do
17:        u←Q.pop()
18:        for each w in adjacentsUnpaired(u, Σ, V) do
19:            if W(u) = W(w) AND f(u) = f(w) then
20:              setPair(w, uw, V)
21:              Q←w
22:              break
23: return V
```

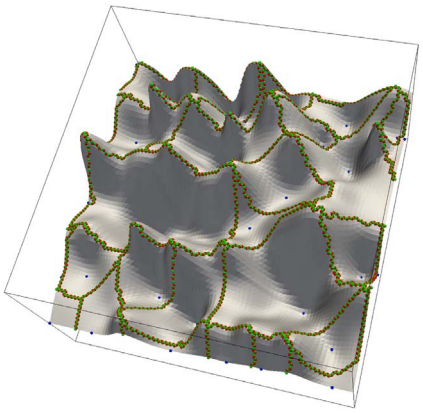

Fig. 2. A triangle mesh representing a terrain, where scalar function f is elevation. Dots indicate the unpaired or critical simplexes after step 2. All triangles contained in some basin are paired, while triangles on the boundary of two or more basins are still unpaired.

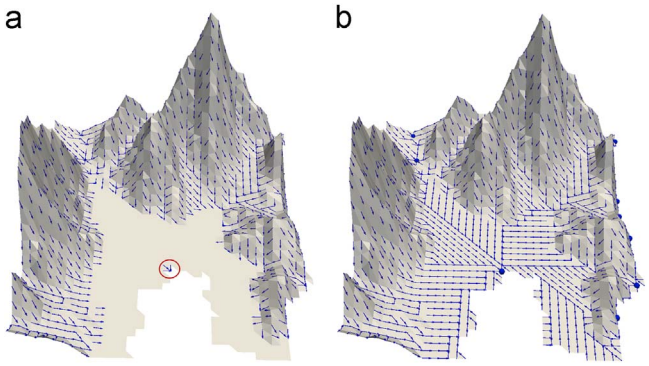

Fig. 3. Pairings inside a plateau minimum. (a) A vertex is chosen as representative minimum (inside the red circle) and all its adjacent vertices are paired. (b) The paths are then extended starting from those vertices until all the vertices in the plateau are paired. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

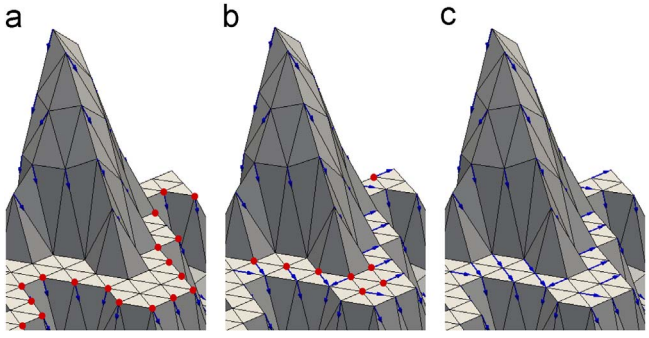

Fig. 4. Pairings inside a non-minimum plateau. (a) Starting from the vertices belonging to the plateau and already paired (red dots), the other vertices are progressively paired (b) and (c). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

5. Step 1: vertex–edge pairing

We first pair the vertices that do not belong to any flat area. We construct the vertex–edge pairs directly from the watershed partition. Each vertex v in Σ is paired with its lowest adjacent vertex u, belonging to the same basin, only if they do not share a flat edge. A second sweep on the vertices of Σ is used for pairing vertices inside flat areas (see Algorithm 1). These are exactly those

left unpaired after the first sweep (row 7). For each unpaired vertex we recursively collect the adjacent vertices inside the flat area. Among them we select those that have been already paired ($K \leftarrow collectPairedInsideFlatArea(\cdot)$). If $K = \varnothing$ the flat area is a minimum and v is set as critical (see Fig. 3a). Each unpaired vertex u, adjacent to v, is then paired with v if the edge uv is flat (rows 11–14). After that, plateau minima and non-minima are treated in the same fashion. We insert the vertices, belonging to the plateau, that have been already paired, in a queue Q (row 17). Vertices in Q are recursively paired, in a breadth first manner, with adjacent vertices having the same value of f and belonging to the same basin (row 20). The latter corresponds to extending the gradient paths upwards, starting from vertices having gradient paths that exit from the plateau (see Fig. 4), or from the artificial minimum (see Fig. 3b) in case of a plateau minimum.

5.1. Correctness of step 1

We have to show that the constructed set of vertex–edge pairs, that we denote with V': (i) has no cycles; (ii) all vertices are paired, with the exception of the minima of f, and of one artificial minimum for each plateau minimum.

We observe that the input of our algorithm is a correct watershed output. Distinct minima of function f (including plateau minima) have different labels, and each other vertex takes its basin label from one of its lower adjacent vertices.

Properties (i) and (ii) are trivially satisfied for non-flat edges. For flat edges, let P be the connected set of vertices that form a plateau, let A be the set of edges in Σ that connect vertices in P and let A^* be the set of edges in A that are paired with a vertex in P. The graph (P, A^*) is a spanning forest of the graph (P, A). The roots of trees in (P, A^*) are vertices paired with an edge connecting it to a lower neighbor (for non-minimum plateaus) or the artificial minimum (for plateau minima). Since there are no cycles in a forest, property (i) is satisfied. Since (P, A) is a connected graph, each node in P is either a node or a root of some tree, implying property (ii).

6. Step 2: edge–triangle pairing inside basins

We extend the collection V' of vertex–edge pairs produced at step 1 by adding edge–triangle pairs to V' for each triangle t where the three vertices of t belong to the same basin R. In that case, for short, we say that triangle t *is inside* R. We denote the extended set as V''.

We use a priority queue Q of edge–triangle pairs organized in descending order of the interpolated function value at the centroids of the edges. Q is initialized with pairs (e, t), where e is an unpaired edge on the boundary of R and t is an unpaired triangle inside R and incident in e. Then, for each edge–triangle pair (e, t) in Q, if t is already paired the pair is skipped. Note that the pairing of t might have occurred while processing another edge of t with higher priority than e. Otherwise, edge e and triangle t are paired in V''. Each unpaired edge e' of t, with $e' \neq e$, is added to Q together with the triangle t' sharing e' with t only if t' is inside the same basin as t. The process continues until Q becomes empty.

The above process progressively erodes the set of unpaired triangles inside a basin, starting from triangles lying on the boundary of the set (which are paired first) and then moving into the basin.

6.1. Correctness of step 2

We have to show that (i) the edge–triangle pairs constructed do not form cycles, and (ii) all triangles lying inside a basin have been paired.

Let us consider the set of triangles inside a basin R. This set consists of one or more connected components, each bounded by one or more cycles of edges.

Q is initialized using the unpaired edges belonging to such cycles. Each time a triangle t is paired with one of those edges, the other (unpaired) edges of t are inserted in the queue. Thus, after each pairing, we obtain a new set of bounding cycles, which are contained in the previous ones, until there are no more unpaired triangles inside R. Property (i) is shown by noting that generated gradient arrows are always directed from outer boundary (before the update) to inner boundary (after the update). Fig. 5 shows a working example; note that a connected component can be split into two, and in that case each of the two cycles has an edge added to the queue.

For property (ii), we have to show that Q is initialized with at least one unpaired boundary edge for each connected component of (unpaired) triangles inside a basin R. This follows from the fact that there is, at least, one unpaired edge e in each cycle C bounding a connected component of triangles inside a basin R in the initial configuration. Otherwise, there would be a vertex in C that is paired with more than one edge, or there would be a closed path in V' consisting of vertices and edges in cycle C, which is false (as we proved in Section 5.1).

Algorithm 2. edgeTrianglePairing(Σ, f, V).

```
1: Input: Σ, triangle mesh
2: Input: f, scalar function
3: Input/Output: V, Forman gradient
4:
5: visited←array[vertices in Σ]
6: for each vertex v in Σ do
7:   if !visited[v] AND isMaximum(v, Σ, f) then
8:     K←collectPlateauVertices(v, Σ, f)
9:     setVisited(K,visited)
10:    t←pickBestTriangle(K, Σ, V, f)
11:    if t != ∅ then
12:      setCritical(t, V)
13:      Q←t
```

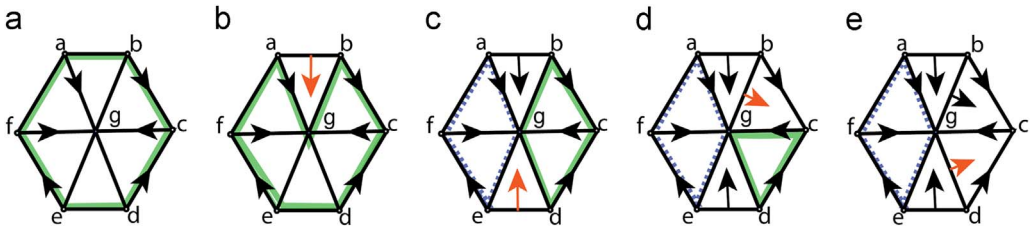

Fig. 5. Successive pairings inside a basin and updates of the corresponding cycles. (a) Discrete Morse gradient V' with only vertex–edge pairs. Vertices *abcdef* define edges in cycle C. (b) After pairing of unpaired edge *ab* with triangle *abg*, edge *ab* is replaced with edges *ag* and *bg* in cycle C. (c) After pairing of unpaired edge *de* with triangle *deg*, cycle C is split in two, C' defined by *bcdg* and C'' by *agef*. (d) After pairing of unpaired edge *bg* with triangle *bcg*, cycle C' surrounds a unique triangle *cdg*, (e) which is then paired with unpaired edge *dg*.

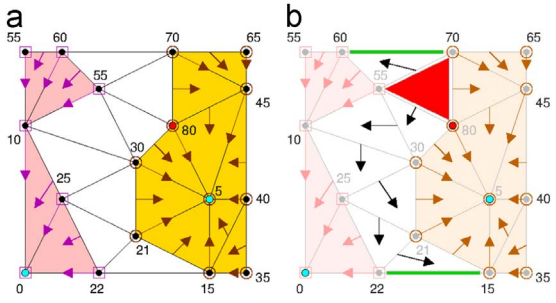

Fig. 6. A terrain with two minima (cyan) and one maximum (red). (a) Pairs obtained after the first two steps. (b) The red triangle is defined as a maximum and the descending paths from its edges to the two green critical edges are created. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

```
14:        while !Q.isEmpty() do
15:            t←Q.pop()
16:            for each unpaired edge e in t do
17:              t′←adjacent(e, t)
18:              if t′ < t then
19:                setPair(e, t′, V)
20:                Q←t′
21:              else
22:                setCritical(e,V)
23:    else
24:        reversePath(Σ, K, f, V)
```

Algorithm 3. reversePath(Σ, K, f, V).

```
1: Input: Σ, triangle mesh
2: Input: K, vertices in the plateau
3: Input: f, scalar function
4: Input/Output: V, Forman gradient
5:
6: t←pickBestTriangleInsideBasin(K, Σ, V, f)
7: setCritical(t, V)
8: e←getPair(t, V)
9: t′←adjacent(e, t)
10: while t′ < t do
11:    e′←getPair(t′, V)
12:    removePair(e, t, V)
13:    setPair(e, t′, V)
14:    e←e′
15:    t←t′
16:    t′←adjacent(e, t, Σ)
17: setCritical(e,V)
```

7. Step 3: edge–triangle pairing on basin boundaries

At the end of the second step, all triangles inside each basin are paired. Unpaired triangles are limited to those on the boundary between basins, and are organized into triangle strips, as depicted in Fig. 2 (general view) and in Fig. 6a (detail).

The following step is described in Algorithm 2. We identify both isolated maxima of f (vertices of Σ having all the adjacent vertices with a lower function value), and plateau maxima, which are sets of connected vertices such that all the vertices adjacent to them have the same or a lower function value. For each maximum, we pick the corresponding critical triangle (row 10) among those incident in the maximum vertex, or in one of the vertices of the plateau maximum. Precisely, the critical triangle is the unpaired triangle having the maximum number of unpaired edges on its boundary. If two or more triangles have the same number of unpaired edges, we pick the one having the centroid with the highest (interpolated) function value. From maximum triangles, we start a descending process in which we create new gradient pairs. We say that $t' < t$ when the third vertex of t' (not in t) has a lower function value than the two shared vertices. Let t be the current triangle (initially, a maximum triangle). We initiate a descending gradient path from each of its unpaired edges (row 16). We pair each of such edges e with its incident triangle t' different from t, if $t' < t$. Otherwise, we declare edge e as critical. A working example is shown in Fig. 6. Cases exist where maxima lie inside some basin R instead of lying on its boundary. In this case the identified critical maximum is surrounded by paired triangles. In such cases (see Algorithm 3), triangle t is chosen based on the function value of the centroid and declared critical (rows 6 and 7). Then, the gradient path originally converging to t is reversed.

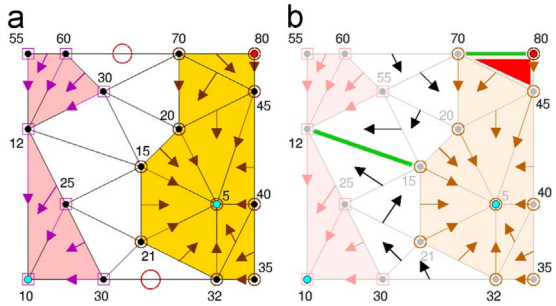

Fig. 7. A terrain with two minima (cyan) and one maximum (red). (a) Pairs obtained after the first two steps. (b) The red triangle is defined as a maximum and the descent from it ends at the green critical edge on the boundary, white triangles remain unpaired. The descent is restarted from the boundary triangles marked with circle ending on the other green (critical) edge. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

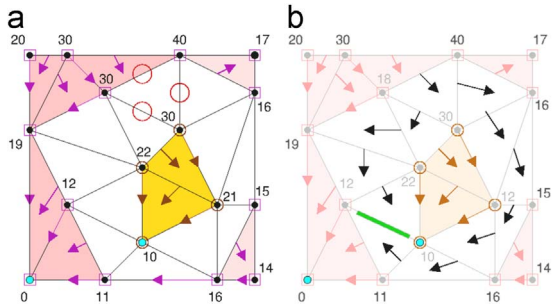

Fig. 8. (a) A basin (yellow) completely contained in another one (pink). The descent starts from the unpaired triangle having three unpaired edges (marked with red circles). (b) The set of triangles paired with edges and the new critical (green) edge. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

7.1. Special cases

The above process may leave unpaired triangles and edges when the domain has a boundary (see Fig. 7a) or when a basin R_1 is completely contained in another basin R_2 (see Fig. 8a).

In such cases, we need to find other triangles (and edges) to restart a descending path. In the first case, an unpaired triangle t must be adjacent to the boundary of the domain along an unpaired edge e. In the second case, either there is an unpaired edge e on the boundary of R_2 (it has endpoints in R_2, and the third vertex of the unpaired triangle t incident to e is in R_1), or there is a critical edge c with endpoints in R_2, such that there is a gradient V''-path starting from an edge e on the boundary of R_2 and ending at a

triangle incident to c. In the latter case, we reverse this gradient path making e unpaired. Among such candidates, we select the one with the highest interpolated function value on the unpaired edge e, then we pair e and t, and we restart a descent from each other unpaired edge of t (as described before). This process continues until all triangles have been paired. An example involving a boundary edge is shown in Fig. 7. An example with a basin contained in another basin is shown in Fig. 8.

7.2. Correctness of step 3

Step 3 creates one maximum triangle in the gradient field for each maximum of function f. We have to show that no cycles are created during the descent. Since all the gradient paths are built following a descent on the function value, no closed V-paths can be obtained.

In the special cases, we have to show that at least one triangle–edge pair exists as a candidate to restart the descent. In case where an unpaired triangle is on the boundary of the domain, the boundary edge must be unpaired, because it could only be paired with the triangle itself. In case where one basin R_1 is completely contained in another basin R_2, if an unpaired edge e exists on the boundary of R_2 that is incident to an unpaired triangle t in the triangle strip between R_1 and R_2, we are done. Otherwise, let us consider the simplicial complex Σ_R defined by connected component of triangles in R_2 that encloses R_1 and all the incident edges and vertices, together with the restriction $V^{''}_R$ of $V^{''}$ to Σ_R. Note that each two vertices in Σ_R are connected through a $V^{''}_R$-path. Since Σ_R has a hole corresponding to R_1 and $V^{''}_R$ is a discrete Morse gradient, there must be at least one critical edge c in Σ_R incident to two triangles t_1 and t_2 in Σ_R. Since all triangles in Σ_R are paired, there is a gradient $V^{''}_R$-path p_1 starting at an edge e_1 on the boundary of R_2 and ending at t_1, and similarly for t_2. Such gradient paths define triangle strips, i.e., each two consecutive triangles in the path share an edge that is paired with one of the two triangles. The two edges e_1 and e_2 must be on two different cycles of boundary edges of R_2, because otherwise the union of triangles in the two paths p_1 and p_2 would disconnect the vertices in Σ_R, i.e., the endpoints of c would not be connected through a gradient path in Σ_R. For similar reasons, such gradient paths for two different critical edges cannot start at edges belonging to the same two cycles. Thus, for each basin R_1 enclosed in R_2 there must be an edge e on the boundary of R_2 that is incident to a triangle t with the third vertex in R_1, such that there is a gradient $V^{''}$-path starting at e and ending at some triangle in R_2 incident to a critical edge c. The reversal of this gradient path will pair e with t, and step 3 of the algorithm can proceed from the remaining two edges of t. From the discussion above, it follows that path reversal and the pairing of triangles between R_1 and R_2 will not create closed gradient V-paths. Thus, the gradient field V, built by our algorithm, is a discrete Morse gradient on Σ: it is defined by vertex–edge and edge–triangle

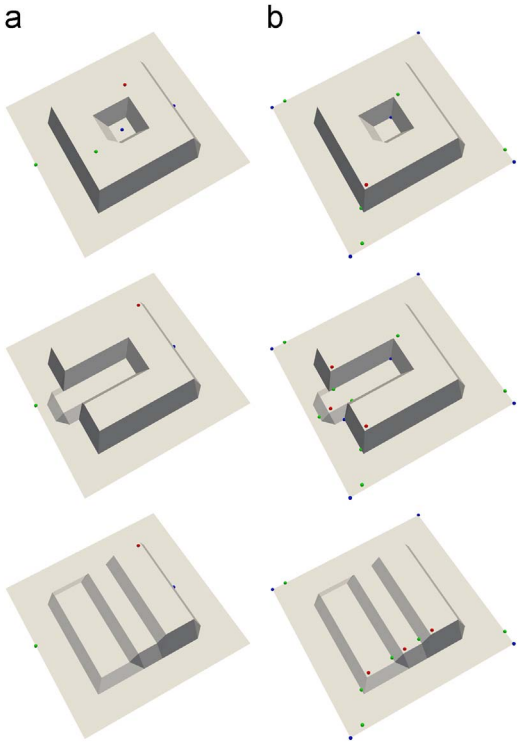

Fig. 9. (a) Our algorithm finds the correct number of critical simplexes, and (b) the one in [16] finds up to 5.6 times more critical simplexes. Dots mark maxima (red), minima (blue), saddles (green). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Table 1
Results obtained from 11 triangle meshes. The first six represent terrain datasets, where the scalar function is elevation. The last five are freeform surfaces, and the scalar function defined on them is mean curvature. The columns show, for each mesh, the number of vertices, triangles, critical simplexes, the computation times for our algorithm for performing steps 1, 2 and 3, and the total computation time of our approach including the preprocessing step computing the input watershed segmentation, the number of critical simplexes obtained while applying simulation of simplicity (SoS) and the improved version described in [22] (SoS++). The last column shows the computational time for the homotopy expansion algorithm (timings are expressed in seconds). Blue and red values indicate the lowest and highest value for each column, respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Dataset	$\|V\|$	$\|T\|$	$\|CP\|$	Our Method.				SoS [4]	SoS++ [22]	Exp.[16]
				1	2	3	Tot.	|CP|		Time
Crater	100K	199K	869	0.4	0.4	0.3	1.3	1637 (+46.9%)	1609 (+45.9%)	2.6 (x2.00)
GardaLake	810K	1.6M	62291	2.9	2.9	2.9	10.1	99259 (+37.2%)	77733 (+19.8%)	20.6 (x2.03)
Kilimanjaro	490K	0.9M	26175	1.5	1.7	1.5	5.4	38591 (+32.1%)	29297 (+10.6%)	11.6 (x2.14)
ComoLake	810K	1.6M	43269	2.3	2.8	2.6	8.9	68729 (+37.0%)	52573 (+17.6%)	19.3 (x2.16)
Baia	4.1M	8.3M	31805	9.5	17.2	16.0	48.8	37541(+15.2%)	34665 (+8.2%)	91.8 (x1.88)
Puget	9M	19M	328235	26.7	39.5	37.2	122.1	417943 (+21.4%)	347649 (+5.5%)	218.9 (x1.79)
OilPump	0.5M	1.1M	24436	1.3	2.1	0.9	5.3	28070 (+13.0%)	26210 (+6.7%)	13.1 (x2.4)
Carter	0.5M	1.0M	35306	1.9	1.3	1.1	5.8	52336 (+32.3%)	47386 (+25.4%)	12.4 (x2.0)
Eros	0.4M	0.9M	41236	1.2	1.5	0.9	4.5	63182 (+34.8%)	45748 (+9.8%)	10.8 (x2.4)
Rim01	0.6M	1.3M	57320	3.0	2.2	2.0	8.4	83722 (+31.6%)	75026 (+23.5%)	14.9 (x1.7)
Neptune	0.2M	4.0M	272546	7.2	6.6	5.4	23.9	390988 (+30.3%)	334558 (+18.5%)	50.3 (x2.1)

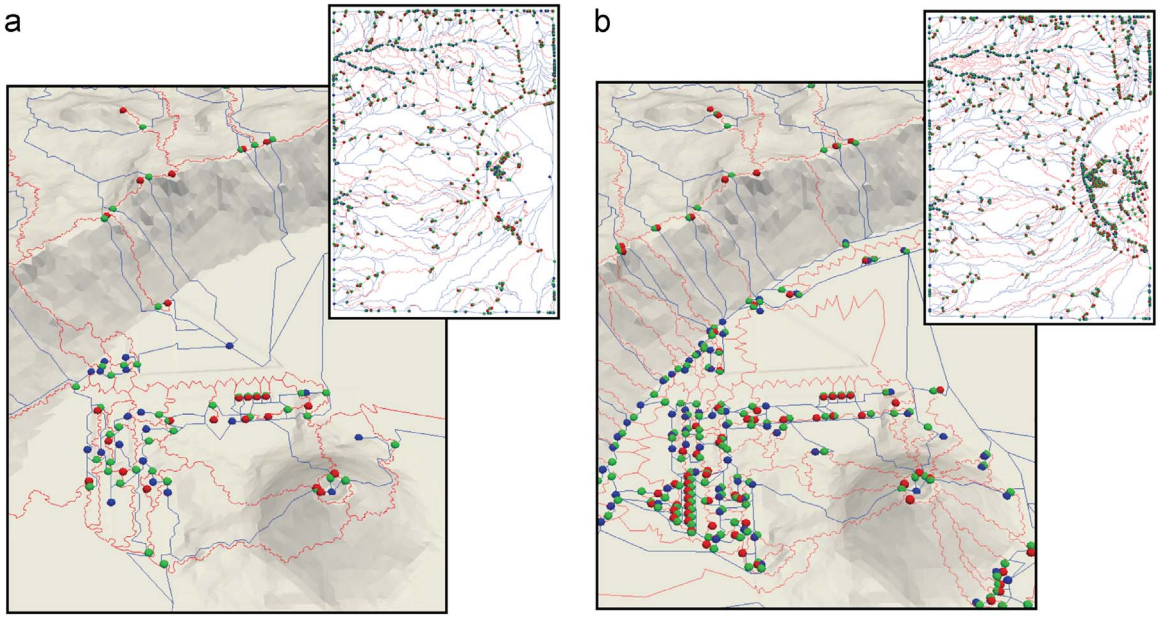

Fig. 10. The 953 critical simplexes, connected through a discrete Morse gradient, computed on the Crater dataset (a) using our algorithm. Using the algorithm in [16] and SoS 1761 critical points are found (b). Red, blue and green dots mark maxima, minima and saddles, respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

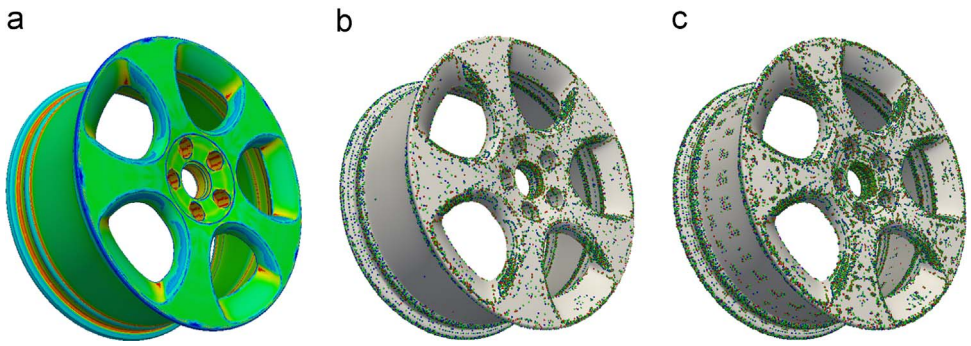

Fig. 11. (a) The Rim01 dataset enriched with the mean curvature for each vertex. Our method identifies 57,320 critical points (b) while using the algorithm in [16] and SoS, 83,722 critical points are found (c). Red, blue and green dots mark maxima, minima and saddles, respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

pairs, each element of Σ is in at most one pair and there are no closed V-paths.

8. Results and discussion

The presented approach is based on the idea of introducing a single critical vertex or triangle wherever there is a minimum or maximum in the dataset in the piecewise-linear sense (see [29]). We have proved that no vertices and triangles (other than those associated with minima and maxima, respectively) remain unpaired after applying our algorithm and, thus, the number of critical minima and maxima is correct. We checked that also the number of critical edges found is always correct by exploiting the Euler formula and the known Euler characteristic of the domain of our test meshes.

In this section we compare the results obtained using our approach with those combining Simulation of Simplicity (SoS) [4], or its improved version (SoS++) [22], with the algorithm for Forman gradient computation based on homotopy expansion in [16]. For encoding the triangle mesh and representing the Forman gradient we are using the efficient representation described in [19]. Experiments have been performed on a MacBook Pro with a 2.8 GHz processor and 16 GB of memory. The segmentation algorithm used in our experiments is the watershed by simulated immersion [11].

Tests have been performed on a variety of triangle meshes representing both terrain datasets and 3D shapes (closed triangulated surfaces). The results are shown in Table 1. For each dataset, we present the number of vertices ($|V|$), triangles ($|T|$) and the total number of critical points ($|CP|$) of the scalar field. Note that this latter is equal to the total number of critical simplexes of the discrete Morse gradient extracted by our algorithm. The timings required by the new algorithm are shown separately for each step. We can notice that the time required by step 1 is generally less than that required by step 2, since steps 1 and 2 cycle over vertices and triangles, respectively, and there are twice as many triangles as vertices in any triangle mesh. For the same reason, step 3 requires about the same time as step 2 since it cycles over the triangles for a second time. Column *Tot* indicates the time required by the algorithm including also the computation of the watershed segmentation. Columns SoS and SoS++ in Table 1 show the number of critical points obtained imposing a total order on the vertices based on the respective methods. We

can notice the drawback of introducing noise for perturbing the input function around flat areas. When the input function has large flat areas, the number of spurious critical simplexes found by the algorithm increases up to a maximum of 46% of the total number of critical simplexes found by our algorithm. SoS++ solves only partially the problem and for some datasets more than 40% of the critical simplices identified are still spurious. Fig. 9 shows this behavior on three toy examples; Figs. 10 and 11 show it on two real datasets. On the left, we show the critical simplexes extracted by our algorithm, on the right those extracted by the algorithm in [16] after SoS is applied. The last column in Table 1 presents the time required for computing the Forman gradient using the algorithm described in [16]. Notice that the latter does not include the time required for creating the total order with SoS or SoS++. By comparing the timings of the two algorithms, we see that our method takes less time in general, being up to 2.1 times faster. The algorithm in [16], however, is dimension independent, while our algorithm is designed for triangle meshes only, and does not extend to higher dimensions in a straightforward manner. Another approach could be to eliminate plateaus with the method in [28] and then compute a discrete Morse gradient with the algorithm in [16]. However, the results presented in Table 1 make such a comparison unnecessary, because our method is already faster than [16] alone.

9. Concluding remarks

We have proposed a new algorithm that constructs a discrete Morse gradient on triangulated surfaces endowed with a scalar field by starting from a watershed decomposition of their set of vertices. The first property of our approach is that the discrete Morse gradient computed maintains the vertex classification inferred by the watershed decomposition, since the vertex–edge pairs are built in the same way as in the watershed decomposition. By combining an algorithm for watershed decomposition as a preprocessing step and the algorithm proposed here, we are able to process scalar fields that are not generic, i.e., that have adjacent vertices with same function value. This opens up to the use of discrete Morse theory in real applications where flat areas are common, such as in terrain modeling but not only. Our method is preferable to other approaches based on simulation of simplicity [4,22], since it preserves the original number of topological features. We have discussed the soundness of the new method and have demonstrated the importance of properly handling flat areas when dealing with morphological analysis of terrains and shapes. Also we have provided a fair comparison between our algorithm and the state of the art in computing a discrete Morse gradient starting from a given scalar field in 2D. We are currently working on a parallel implementation of our algorithm. In particular the vertex–edge pairing (Section 5) and the edge–triangle pairing (Section 6) can be parallelized improving time efficiency. We think that extending the computation of a discrete Morse gradient to functions with flat areas will strongly encourage the wider use of topological tools in shape analysis. The source code is publicly available (⟨https://github.com/IuricichF/FormanGradient2D⟩).

Acknowledgements

All the datasets used are courtesy of the Aim@Shape repository [30] and the Virtual Terrain Project [31]. This work has been partially supported by the US National Science Foundation under grant number IIS-1116747. The support of the Hungarian Academy of Sciences through the DOMUS project (number 5706/11/2015/HTMT) is also acknowledged.

References

[1] Forman R. Morse theory for cell complexes. Adv Math 1998;134:90–145.
[2] Milnor J. Morse theory. New Jersey: Princeton University Press; 1963 ISBN 0-691-08008-9.
[3] De Floriani L, Fugacci U, Iuricich F, Magillo P. Morse complexes for shape segmentation and homological analysis: discrete models and algorithms. Comput Graph Forum 2015;34(2):761–85. http://dx.doi.org/10.1111/cgf.12596.
[4] Edelsbrunner H, Mücke EP. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans Graph 1990;9(1):66–104. http://dx.doi.org/10.1145/77635.77639.
[5] Edelsbrunner H, Letscher D, Zomorodian A. Topological persistence and simplification. Discret Comput Geomet 2002;28(4):511–33. http://dx.doi.org/10.1007/s00454-002-2885-2.
[6] De Floriani L, Iuricich F, Magillo P, Simari P. Discrete Morse versus watershed decompositions of tessellated manifolds. In: Image analysis and processing—ICIAP 2013: 17th international conference, Naples, Italy; September 9–13, 2013. Proceedings, Part II. Berlin, Heidelberg: Springer; 2013, p. 339–48. ISBN 978-3-642-41184-7.
[7] Meyer F. Topographic distance and watershed lines. Signal Process 1994;38:113–25. http://dx.doi.org/10.1016/0165-1684(94)90060-4.
[8] Najman L, Schmitt M. Watershed of continuous functions. Signal Process 1994;38(1):99–112.
[9] Forman R. Combinatorial vector fields and dynamical systems. Math Z 1998;228:629–81. http://dx.doi.org/10.1007/PL00004638.
[10] Forman R. A user's guide to discrete Morse theory. Sémin Lothar Combinat 2002;48:35.
[11] Vincent L, Soille P. Watershed in digital spaces: an efficient algorithm based on immersion simulation. IEEE Trans Pattern Anal Mach Intell 1991;13(6):583–98. http://dx.doi.org/10.1109/34.87344.
[12] Soille P. Morphological image analysis: principles and applications. Berlin, New York: Springer-Verlag; 2004 ISBN 3540429883.
[13] Roerdink J, Meijster A. The watershed transform: definitions, algorithms, and parallelization strategies. Fundam Inform 2000;41:187–228.
[14] Stoev SL, Strasser W. Extracting regions of interest applying a local watershed transformation. In: Proceedings of the IEEE visualization'00. ACM Press; IEEE Computer Society Press, Los Alamitos, CA, USA, 2000. p. 21–8. ISBN 1-58113-309-X.
[15] Mangan A, Whitaker R. Partitioning 3D surface meshes using watershed segmentation. Trans Vis Comput Graph 1999;5(4):308–21. http://dx.doi.org/10.1109/2945.817348.
[16] Robins V, Wood PJ, Sheppard AP. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. IEEE Trans Pattern Anal Mach Intell 2011;33(8):1646–58. http://dx.doi.org/10.1109/TPAMI.2011.95.
[17] Shivashankar N, Maadasamy S, Natarajan V. Parallel computation of 2D Morse–Smale complexes. IEEE Trans Vis Comput Graph 2012;18(10):1757–70. http://dx.doi.org/10.1109/TVCG.2011.284.
[18] Shivashankar N, Natarajan V. Parallel computation of 3D Morse–Smale complexes. Comput Graph Forum 2012;31(3):965–74. http://dx.doi.org/10.1111/j.1467-8659.2012.03089.x.
[19] Weiss K, Iuricich F, Fellegara R, De Floriani L. A primal/dual representation for discrete Morse complexes on tetrahedral meshes. Comput Graph Forum 2013;32(3):361–70. http://dx.doi.org/10.1111/cgf.12123.
[20] Gyulassy A, Bremer P, Pascucci V. Computing Morse–Smale complexes with accurate geometry. IEEE Trans Vis Comput Graph 2012;18(12):2014–22. http://dx.doi.org/10.1109/TVCG.2012.209.
[21] King H, Knudson K, Mramor N. Generating discrete Morse functions from point data. Exp Math 2005;14(4):435–44 ⟨http://projecteuclid.org/euclid.em/1136926974⟩.
[22] Gyulassy A, Bremer PT, Hamann B, Pascucci V. Practical considerations in Morse-Smale complex computation. In: Topological methods in data analysis and visualization: theory, algorithms, and applications. Mathematics and visualization. Heidelberg: Springer-Verlag; 2011. p. 67–78. http://dx.doi.org/10.1007/978-3-642-15014-2_6.
[23] Fellegara R, Iuricich F, De Floriani L, Weiss K. Efficient computation and simplification of discrete Morse decompositions on triangulated terrains. In: SIGSPATIAL'14 Proceedings of the 22nd international conference on advances in geographic information systems; 2014. p. 223–32. http://dx.doi.org/10.1145/2666310.2666412.
[24] Günther D, Reininghaus J, Wagner H, Hotz I. Efficient computation of 3D Morse–Smale complexes and persistent homology using discrete Morse theory. Vis Comput 2012;28(10):959–69. http://dx.doi.org/10.1007/s00371-012-0726-8.
[25] Gyulassy A, Günther D, Levine JA, Tierny J, Pascucci V. Conforming Morse–Smale complexes. IEEE Trans Vis Comput Graph 2014;20(12):2595–603. http://dx.doi.org/10.1109/TVCG.2014.2346434.
[26] Meyer F, Beucher S. Morphological segmentation. J Vis Commun Image Represent 1990;1:21–45. http://dx.doi.org/10.1016/1047-3203(90)90014-M.

[27] Dijkstra EW. A note on two problems in connexion with graphs. Numer Math 1959;1:269–71. http://dx.doi.org/10.1007/BF01386390.
[28] Magillo P, De Floriani L, Iuricich F. Morphologically-aware elimination of flat edges from a TIN. In: Proceedings of the 21th ACM SIGSPATIAL international conference on advances in geographic information systems; 2013. p. 244–53. http://dx.doi.org/10.1145/2525314.2525341.
[29] Banchoff T. Critical points and curvature for embedded polyhedral surfaces. Am Math Mon 1970;77(5):475–85.
[30] Aim@shape. ⟨http://visionair.ge.imati.cnr.it/ontologies/shapes/⟩; 2016.
[31] Virtual Terrain Project. ⟨http://vterrain.org⟩; 2016.