I/O-Efficient Contour Queries on Terrains

Pankaj K. Agarwal[*] Thomas Mølhave[†] Bardia Sadri[‡]

Abstract

A terrain $\boldsymbol{M}$ can be represented as a triangulation of the plane along with a height function associated with the vertices (and linearly interpolated within the edges and triangles) of $\boldsymbol{M}$. We investigate the problem of answering *contour queries* on $\boldsymbol{M}$: Given a height ℓ and a triangle f of $\boldsymbol{M}$ that intersects the level set of $\boldsymbol{M}$ at height ℓ, report the list of the edges of the connected component of this level set that intersect f, sorted in clockwise or counter-clockwise order. Contour queries are different from level-set queries in that only one contour (connected component of the level set) out of all those that may exist is expected to be reported. We present an I/O-efficient data structure of linear size that answers a contour query in $O(\log_B N + T/B)$ I/Os, where N is the number of triangles in the terrain and T is the number of edges in the output contour. The data structure can be constructed using $O(\text{Sort}(N))$ I/Os.

1 Introduction

Over the last two decades there has been extensive work on modeling, analyzing, and visualizing terrain data in computational geometry, GIS, and spatial databases. A three-dimensional (digital elevation) model of a terrain is often represented as a triangulated xy-monotone surface, also known as a *triangulated irregular network* (TIN). This surface can be regarded as the graph of a piecewise-linear *height* or *elevation* function $h : \mathbb{R}^2 \to \mathbb{R}$. A *contour* (or *contour line*) of a terrain $\boldsymbol{M}$ is a connected component of a level set of h. Each contour K in a *regular* level set (see Section 2 for the definition) is a polygon in $\mathbb{R}^2$ (See Figure 1). Many terrain processing applications involve computing of one or many contours on a given terrain. In particular, a fundamental operation integral to such systems is answering *contour queries*: the user selects a point on the terrain (possibly using some provided rendering), and the system computes (and perhaps displays) the contour that goes through that particular point. These queries are related to *level-set queries* in which all contours at a chosen height, comprising an entire *level set*, are reported. Contour queries can be used, among other things, to compute the projected area of some feature of the terrain. In fact, computing the projected area of a hill or mountain was one of the earliest motivations for computing contours [14]. If h represents other measured quantities over a planar domain, e.g. the barometric pressure, then contour queries can be used to compute, e.g. the domain or area of a high-pressure system. These queries also define many fundamental operations in terrain editing software, where a contour defines a "cut" in a terrain.

An alternative definition for contour queries replaces the query point with a pair (f, ℓ) where ℓ is a height and f is a terrain triangle that spans height ℓ. The target contour in this case is the (unique) contour at height ℓ that crosses f, denoted by $K_{\boldsymbol{M}}(f, \ell)$.[1] The *contour-query* problem thus can be formulated as follows: Preprocess a terrain $\boldsymbol{M}$ into a data structure so that for a height ℓ and a triangle f, the con-

[*]Duke University, Durham, NC. Supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

[†]Duke University, Durham, NC. Supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

[‡]University of Toronto, Toronto, ON, Canada

[1]If $\boldsymbol{M}$ is obvious from the context or not important, we will simply use the notation $K(f, \ell)$.

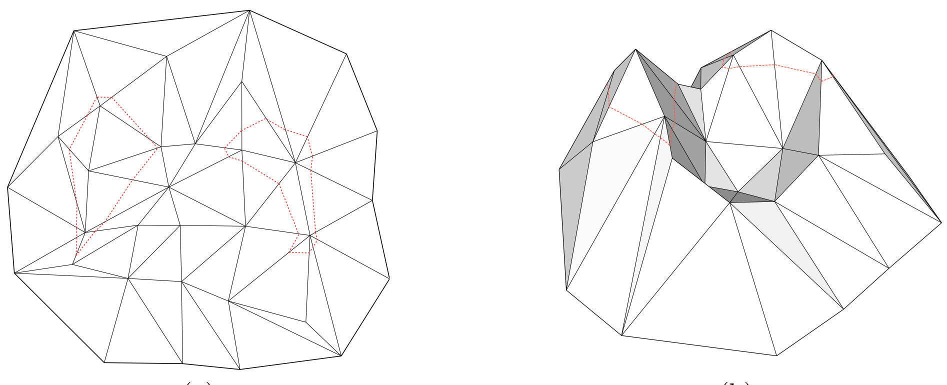


Figure 1: (a) the plane triangulation of a terrain together with a level set of that terrain consisting of two contours (shown in dotted lines). Only the bounded edges and triangles of the triangulation are displayed. (b) The three-dimensional interpretation of the same terrain.

tour $K_{\boldsymbol{M}}(f, \ell)$ can be reported quickly in an output-sensitive manner — edges of the contour should be reported sorted in clockwise or counterclockwise order.

With recent advances in mapping technologies, such as laser based LIDAR, billions of points on a geographical terrain, at sub-meter resolution, can be acquired with very high accuracy (∼10-15 cm) in a short period of time. The terrain models generated from these data sets are too large to fit in main memory and thus reside on disks. Transfer of data between disk and main memory is often the bottleneck in the efficiency of algorithms for processing these massive terrain models (see e.g. [12]). We are therefore interested in developing efficient algorithms in the two-level I/O-model [4]. The machine consists of a main memory of size M and an infinite-size disk. A block of B consecutive elements can be transferred between main memory and disk in one *I/O-operation* (or simply *I/O*). Computation can only take place on elements in main memory, and the complexity of an algorithm is measured in terms of the number of I/Os it performs. See the recent surveys [5, 17] for a comprehensive review of I/O-efficient algorithms. Here we mention that sorting N elements takes $\Theta\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os, and we denote this quantity by $\mathrm{Sort}(N)$. It is common to assume that $M \geq B^2$. This is usually called the *tall cache* assumption.

Related work. A natural way of computing a contour K of a terrain $\boldsymbol{M}$ is simply to start at one triangle of $\boldsymbol{M}$ that intersects K, trace out K by walking through $\boldsymbol{M}$ one triangle at a time, and stop on arriving at the starting point. However, it is not clear how to trace a contour efficiently in the I/O-model since a naive implementation requires $O(T)$, instead of $O(T/B)$, I/Os to trace a contour of size T. Even the best-known algorithm for blocking a planar triangulation [1, 15] only leads to an $O(T/\log_2 B)$ I/O solution. See [7] for improved bounds in some special cases. The known I/O-efficient algorithms for answering level-set queries on terrains [2, 1, 9] are not ideal for answering contour queries because the size of a level set to which a contour belongs can be much larger than the size of the contour. This is especially true for large global terrains that next-generation GIS software must face. A common approach when dealing with spatially restricted areas of large terrains is to create a subdivision by splitting the terrain into disjoint *tiles* of relatively small size, and then to extract contour information from a small subset of the tiles. However, finding the tiles containing a specific contour is hard and walking between tiles is in general no better than any other method of traversing a plane graph.

Agarwal *et al.* [2] proposed a linear-size data structure for storing a TIN $\boldsymbol{M}$ of size N so that a level-set query (but not a contour query) of output size T can be answered in $O(\log_B N + T/B)$ I/Os. This data structure improves the previously best known I/O-efficient approaches [9, 1]: it reports the contours in circular order (unlike Chiang and Silva [9]), and it can be efficiently constructed in $O(\mathrm{Sort}(N))$ I/Os (unlike Agarwal et al. [1]). They prove the existence of, and present an efficient algorithm for computing, the so-called *level ordering* for any given $\boldsymbol{M}$. A level ordering is a total order $\prec$ on the triangles of $\boldsymbol{M}$ with the following two crucial properties:

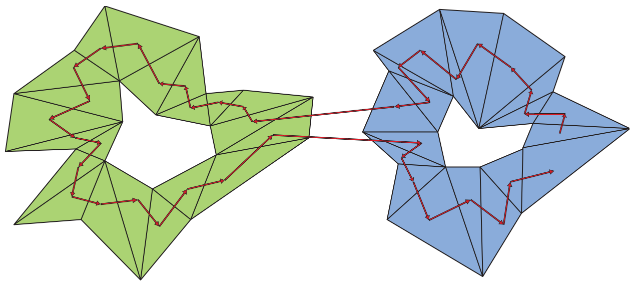


Figure 2: Level-ordering on triangles intersecting two contours of the same level set. An arrow points to the successor of a triangle within the set.

(i) for any contour K at any level, the $\prec$-sorted list of triangles that intersect K enumerates these triangles in the cyclic order they appear along K (in either clockwise or counterclockwise order), and

(ii) for any two contours K_1 and K_2 of the same level set, in the $\prec$-sorted list of triangles that intersect $K_1 \cup K_2$, the triangles intersecting at least one of K_1 or K_2 appear contiguously (See Figure 2).

See Section 4 for precise definitions. In the $\prec$-sorted sequence of all triangles that intersect a given level set, those triangles that intersect a particular contour appear in maximal contiguous subsequences, called *fragments*. Agarwal *et al.* [2] showed how the entire collection of contours, i.e., the whole level set, can be retrieved quickly from these fragments using (i) and (ii). Their data structure can be modified to answer a contour query using $O\Big(\log_B N + J \log_{M/B} N/B + T/B\Big)$ I/Os, where T is the number of triangles that intersect the output contour and J is the number of its fragments in the level ordering used by the algorithm. Unfortunately, J may be as large as T in the level ordering computed by Agarwal *et al.*, so the query uses $O(\log_B N + T)$ I/Os in the worst case.

Our results. In this paper we propose an I/O-efficient algorithm for preprocessing $\boldsymbol{M}$, using $O(\text{Sort}(N))$ I/Os, into a linear-size data structure so that a contour query can be answered in $O(\log_B N + T/B)$ I/Os, where N is the number of triangles in $\boldsymbol{M}$ and T is the size of the query output. The structure relies on the level-ordering method of Agarwal *et al.* [2]. It tackles the fundamental problem of contour fragmentation by transforming $\boldsymbol{M}$ into another terrain $\boldsymbol{M}'$ that has two key properties: (i) $\boldsymbol{M}'$ "preserves" individual contours of $\boldsymbol{M}$. (ii) The image of each contour of $\boldsymbol{M}$ on $\boldsymbol{M}'$ has $O(1)$ fragments in the level ordering computed on $\boldsymbol{M}'$. The key contributions of our paper are the following:

Stretchings We describe in Section 3 a general operation, called "stretching", that modifies a terrain $\boldsymbol{M}$ into $\boldsymbol{M}'$ by changing the heights of some vertices but maintains its underlying triangulation. We prove that the "stetched" terrain $\boldsymbol{M}'$ has the same (augmented) contour tree as that of $\boldsymbol{M}$. We preprocess $\boldsymbol{M}$ and $\boldsymbol{M}'$ in $O(\text{Sort}(N))$ I/Os into a linear-size data structure (see Section 3) that converts, in $O(\log_B N)$ I/Os, any contour query (f, ℓ) on $\boldsymbol{M}$ into a contour query (f, ℓ') on $\boldsymbol{M}'$ such that $K_{\boldsymbol{M}}(f, \ell)$ and $K_{\boldsymbol{M}'}(f, \ell')$ intersect the same sequence of triangles of their commmon underlying triangulation.

Bounding Fragmentation In Section 4, we describe an algorithm that computes a specific stretching $\boldsymbol{M}'$ of $\boldsymbol{M}$ and a level ordering of $\boldsymbol{M}'$, using $O(\text{Sort}(N))$ I/Os, such that each contour of $\boldsymbol{M}$ maps to a contour in $\boldsymbol{M}'$ with $O(1)$ fragments in the level ordering of $\boldsymbol{M}'$.

2 Preliminaries

A *triangulation of* $\mathbb{R}^2$ is a planar subdivision of $\mathbb{R}^2$ in which each face is a (possibly unbounded) triangle. Let M be a triangulation, and let $V(M)$, $E(M)$, and $F(M)$ denote the sets of vertices, edges, and faces (triangles), respectively, of M. We add a vertex v_∞ at infinity and connect all unbounded edges (rays) of M to v_∞. Triangles of M that have v_∞ as a vertex are *unbounded* and all other triangles are *bounded.* A *height function consistent with* M is any continuous function $h : \mathbb{R}^2 \to \mathbb{R}$ whose restriction to each triangle of M is linear and approaches $-\infty$ at infinity, i.e., $h(v_\infty) = -\infty$. A *terrain* $\boldsymbol{M} = (M, h)$ is a triangulation M of $\mathbb{R}^2$ endowed with a height function h that is consistent with M. M can be viewed as the graph of h, an *xy*-monotone piecewise-linear surface in $\mathbb{R}^3$. For simplicity we assume that vertices of terrains have *distinct* heights. We sometimes write an edge uv as a pair (u, v) to indicate that $h(u) < h(v)$, i.e., we implicitly regard uv to be oriented from its *lower endpoint* u to its *higher endpoint* v.

Critical vertices. For any vertex v the *link* $\text{lk}(v)$ is the cycle formed by the edges that are *not* incident to v but belong to triangles that are incident to v. The *upper* (resp. *lower*) link $\text{lk}^+(v)$ (resp. $\text{lk}^-(v)$) is the subgraph of $\text{lk}(v)$ induced by vertices u with $h(u) > h(v)$ (resp. $h(u) < h(v)$). If $\text{lk}^-(v)$ (resp. $\text{lk}^+(v)$) is empty then v is a (local) *minimum* (resp. *maximum*). An *extremum* is a minimum or a maximum (See Figure 3). A non-extremal vertex v is called *regular* if

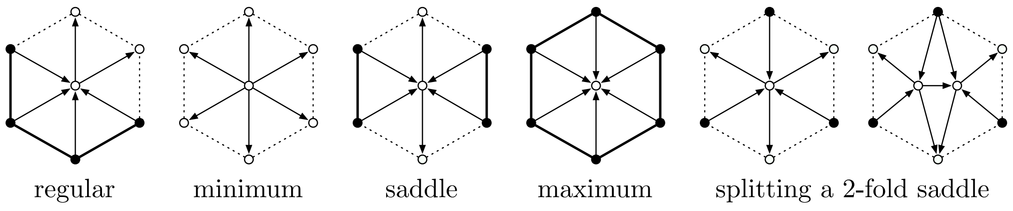


Figure 3: Determining the type of a vertex; lower link vertices are depicted by filled circles.

$\mathrm{lk}^+(v)$ (or $\mathrm{lk}^-(v)$) is connected, otherwise it is called a *saddle*. Non-regular vertices (extrema and saddles) are called *critical*, and their heights are *critical values* of h. For simplicity, we assume that all saddles are *simple*, i.e., their upper (equivalently lower) links have exactly two connected components. (see [13] and Figure 3).

Level sets and contours. For $\ell \in \mathbb{R}$, the *level set*, *sublevel set*, and *superlevel set* of a terrain $\boldsymbol{M} = (M, h)$ at height ℓ are subsets $\boldsymbol{M}_\ell$, $\boldsymbol{M}_{<\ell}$, and $\boldsymbol{M}_{>\ell}$ of $\mathbb{R}^2$ consisting of points x, respectively with $h(x) = \ell$, $h(x) < \ell$, and $h(x) > \ell$. $\boldsymbol{M}_{\leq\ell}$ and $\boldsymbol{M}_{\geq\ell}$ are defined analogously.

Each connected component of a level set $\boldsymbol{M}_\ell$ is called a *contour* of $\boldsymbol{M}$ at level ℓ. A contour of $\boldsymbol{M}_\ell$ can be represented by a pair $(f, \ell) \in F(M) \times \mathbb{R}$, as the contour $K_{\boldsymbol{M}}(f, \ell)$ at height ℓ that intersects the triangle f of M. Each vertex $v \in V(M)$ is contained in exactly one contour in $\boldsymbol{M}_{h(v)}$, which we call *the contour of* v. Contours of regular vertices are simple polygons; contours of extrema are single points; and contours of saddles are polygons that self-intersect exactly once at their corresponding saddles. Being a Jordan curve, the contour K of a regular vertex partitions $\mathbb{R}^2 \setminus K$ into the *inside*, K^{in}, and *outside*, K^{out}, of K.

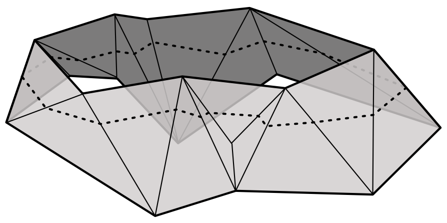


Figure 5: The 3D interpretation of $\bar{F}_{\boldsymbol{M}}(K)$ for a contour K.

A contour is *generic* if it is not the contour of any vertex of the terrain. A generic level set $\boldsymbol{M}_\ell$ is a level set in which all contours are generic, i.e., $\ell \neq h(v)$ for any vertex v. The set of triangles that intersect $\boldsymbol{M}_\ell$ is denoted by $F(\boldsymbol{M}_\ell)$. For a generic contour K in $\boldsymbol{M}_\ell$, let $F_{\boldsymbol{M}}(K)$ be the set of triangles in $F(M)$ that intersect K.[2] Each triangle in $F(K)$ meets K at a single segment. The order in which K intersects the triangles induces a cyclic ordering on $F(K)$. Let $\bar{F}(K)$ be the resulting cyclically ordered set (we take a cyclic ordering and its reverse as equal; See Figure 5). We refer to $\bar{F}(K)$ as the *(combinatorial) representation* of K. Given $\bar{F}(K)$, K (as a plane polygon) can be generated in one scan of $\bar{F}(K)$. Therefore it suffices to generate the cyclic sequence $\bar{F}(K(f, \ell))$ to answer a contour query (f, ℓ).

Let $V^-(K)$ and $V^+(K)$ respectively be vertices from triangles in $F(K)$ with heights smaller and greater than ℓ. The *lower* and *upper supporting vertices* of K are respectively defined as

$$\begin{aligned} v^-(K) &= \arg\max_{v \in V^-(K)} h(v), \\ v^+(K) &= \arg\min_{v \in V^+(K)} h(v). \end{aligned}$$

Two contours K and K' are *equivalent* if $\bar{F}(K) = \bar{F}(K')$. It can be verified that K and K' are equivalent if and only if they have the same lower and upper supporting vertices.

Up- and down-contours. Let $\varepsilon = \varepsilon(\boldsymbol{M})$ denote a sufficiently small positive value, in particular, smaller than height difference between any two vertices of $\boldsymbol{M}$. An *up-contour* of a vertex v is any contour of $\boldsymbol{M}_{h(v)+\varepsilon}$ that intersects an edge incident on v. Similarly, a *down-contour* of v is any contour of $\boldsymbol{M}_{h(v)-\varepsilon}$ that intersects an edge incident on v. If v is maximum (resp. minimum), then its up- (resp. down-) contour is not defined. If v is not a saddle, then its up- and down-contours are unique and they converge to the contour of v as $\varepsilon \to 0$. A saddle is called *positive* if it has two up-contours and one down-contour and *negative* if it has two down-contours and one up-contour. All simple saddles are either negative or positive.

[2] As earlier, $\boldsymbol{M}$ will be dropped from the subscript when it is not important or clear from the context.

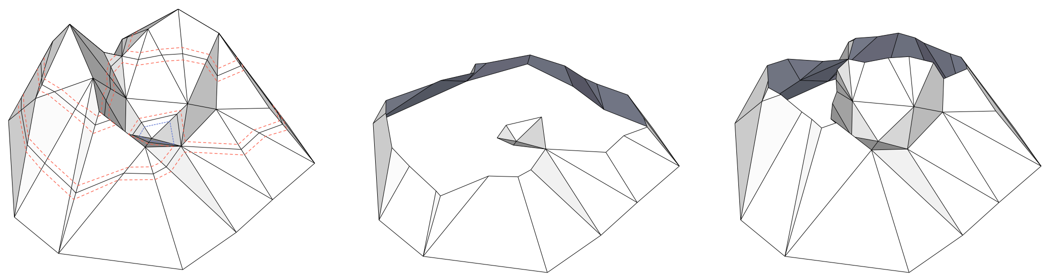


Figure 4: Left: a terrain with one positive and one negative saddle. The saddle contours and their up and down contours are marked in their corresponding colors (or, dotted for blue and dashed for red). Middle: the sub level set of the lower saddle, indicating (together with the colors of the up- and down-contours that it is a red positive saddle. Right: sub level set of the upper saddle: a red negative one.

For a generic contour K, if $v^-(K)$ has a unique up-contour K_v, then K and K_v are equivalent. Otherwise, $v^-(K)$ is a positive saddle and has two up-contours, one of which is equivalent to K and the other intersects no triangles in $F(K)$. The same can be said about down-contour(s) of $v^+(K)$.

Red and blue contours and vertices. A contour K of $\boldsymbol{M}_\ell$ is called *blue* if $\boldsymbol{M}_{\leq \ell}$ is locally in the interior of K and *red* otherwise. Equivalently, a contour K is blue if for any (oriented) edge (u, v) intersecting K, $u \in K^{\text{in}}$ and $v \in K^{\text{out}}$, and red otherwise. We define the *color* of a regular vertex to be the color of the contour that passes through it, which agrees with the color of its up- and down-contours. The color of v_∞ is taken to be red, and the color of all other minima is blue; i.e., the color of their up-contour; the color of a maximum is taken to be red, i.e., the color of its down-contour. The color of a positive (resp. negative) saddle is taken to be the color of its unique down-contour (resp. up-contour). Under this coloring scheme, a red positive saddle will have a red down-contour and two red up-contours while a blue positive saddle will have a blue down-contour together with one red and one blue up-contours (Figure 6). Similarly, a blue negative saddle has all blue up- and down-contours while a red negative saddle has one red down-contour, one blue down-contour, and one red up-contour. The following lemma is equivalent to Lemma 3.1 of [3].

Lemma 2.1. **[3]** *If a generic red (resp. blue) contour K in $\boldsymbol{M}_\ell$ passes through a triangle f, then the vertices of f with height greater than (resp. smaller than) ℓ are red (resp. blue).*

Contour, join, and split trees. Finally, we define the contour tree, and its two relatives the *join* and *split trees* of a terrain [8, 11],[3] which will play crucial roles in our algorithms. They respectively encode the topology of the level sets, sublevel sets, and superlevel sets of $\boldsymbol{M}$ at various heights. The (augmented) contour tree $\mathcal{C}_{\boldsymbol{M}}$ of $\boldsymbol{M}$ (or just $\mathcal{C}$ when $\boldsymbol{M}$ is clear from the context) is a tree on vertex set $V(M)$ in which uv is an edge if u is the upper and v is the lower supporting vertex of some generic contour K of $\boldsymbol{M}$. The edge uv of $\mathcal{C}$ is then said to *carry* K. Equivalently, the contour tree of $\boldsymbol{M}$ is the space that is the quotient of $\boldsymbol{M}$ modulo taking points on the same contour as equivalent. Under the associated quotient map $\pi_{\boldsymbol{M}} : \boldsymbol{M} \to \mathcal{C}_{\boldsymbol{M}}$, the image of each contour of $\boldsymbol{M}$ is a unique point in $\mathcal{C}_{\boldsymbol{M}}$. Contours of vertices of $\boldsymbol{M}$ are mapped into points that are regarded as corresponding vertices of $\mathcal{C}$ while generic contours are mapped into points along edges that carry them (See Figure 7. We *color* each vertex v of $\mathcal{C}$ with the color of contour of v on $\boldsymbol{M}$. Points along the edges of the contour tree are also given the color of the contours they represent. It can be verified that all contours mapped to the interior of the same edge of $\mathcal{C}$ have the same color. We choose v_∞ as the *root* of the contour tree.

The *augmented join tree* $\mathcal{J}$ represents the merging of connected components of $\boldsymbol{M}_{<\ell}$ and the *augmented split tree* $\mathcal{S}$ represents the splittings of the connected components of $\boldsymbol{M}_{>\ell}$ when ℓ varies from $-\infty$ to $+\infty$. More precisely, $\mathcal{J}_{\boldsymbol{M}}$ is a tree on vertex set $V(M)$ in which uv is an edge if v is the highest vertex in a connected component of $\boldsymbol{M}_{<h(u)}$ that contains a vertex of $\text{lk}^-(u)$. $\mathcal{S}_{\boldsymbol{M}}$ is defined symmetrically by negating the height function h. We choose v_∞ as the

[3]In the literature, contour, join, and split trees are often defined as trees on the set of critical vertices and regular vertices are regarded as points along the edges and not as vertices of the tree. As defined here, these trees are usually called the "augmented" versions of themselves.

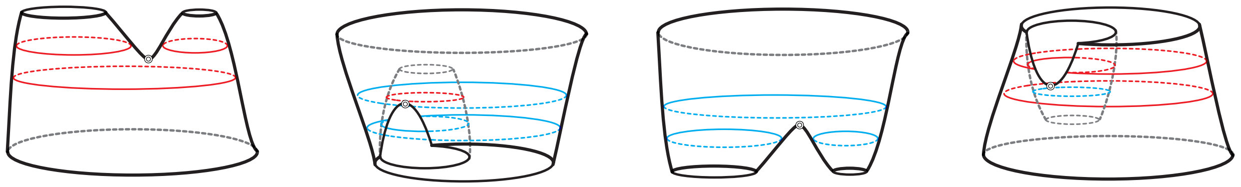
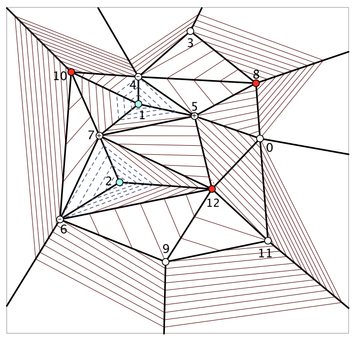
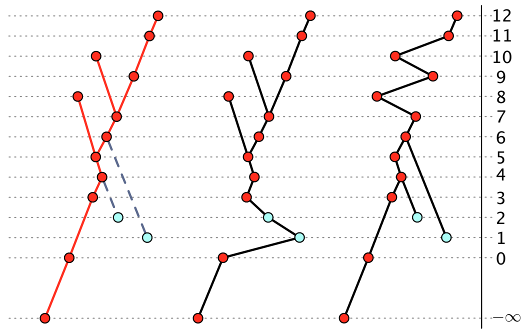


Figure 6: The four possible saddle types. Left to right: red positive, blue positive, blue negative, red negative. The up- and down-contours of the saddle points, but not the saddle contours themselves, are shown.



(a)



(b)

Figure 7: (a) A terrain. Level sets of all vertices shown (dashed for blue). Numbers indicate heights of vertices. (b) (left to right): colored contour tree, split tree, and join tree of the same terrain.

root of $\mathcal{S}$ and the highest maximum of $\boldsymbol{M}$ as the root of $\mathcal{J}$.

3 Stretching Terrains

In this section we introduce *stretching* as a general operation on a terrain $\boldsymbol{M}$, which "preserves" individual contours (as described below) while moving them amongst level sets. We then present an I/O-efficient algorithm for *mapping* a contour $K_{\boldsymbol{M}}(f, \ell)$ to an equivalent contour on the stretched terrain.

The stretching operation. Let K be a generic contour of a terrain $\boldsymbol{M} = (M, h)$, and let δ be an *offset* parameter; $\delta > 0$ if K is red and $\delta < 0$ if K is blue. The *stretching of* $\boldsymbol{M}$ *at* K *by* δ is a terrain $\boldsymbol{M}' = (M, h')$ where for any vertex v:

$$h'(v) = \begin{cases} h(v) + \delta & \text{if } v \in K^{\text{in}}, \\ h(v) & \text{if } v \in K^{\text{out}}. \end{cases}$$

A terrain $\boldsymbol{M}'$ is called a *stretching* of $\boldsymbol{M}$ if there is a sequence $\boldsymbol{M} = \boldsymbol{M}_1, \ldots, \boldsymbol{M}_k = \boldsymbol{M}'$ of terrains where each $\boldsymbol{M}_i$ is obtained from $\boldsymbol{M}_{i-1}$ by stretching it at one of its generic contours. Note that the underlying triangulations of $\boldsymbol{M}'$ and $\boldsymbol{M}$ are the same. If K_1 and K_2 are equivalent generic contours of $\boldsymbol{M}$ with $K_2 \subset K_1^{\text{in}}$, then there are no vertices in $K_1^{\text{in}} \cap K_2^{\text{out}}$. Therefore, stretching $\boldsymbol{M}$ at K_1 or K_2 by δ produces the same terrain. The following lemma shows that certain properties of a terrain are preserved by stretching.

Lemma 3.1. *Let $\boldsymbol{M}' = (M, h')$ be a stretching of a terrain $\boldsymbol{M} = (M, h)$.*

(1) For any adjacent vertices u and v of M, $h(u) > h(v)$ if and only if $h'(u) > h'(v)$.

(2) The type (regular, minimum, maximum, positive saddle, or negative saddle) of any vertex $v \in V(M)$ is the same in both $\boldsymbol{M}$ and $\boldsymbol{M}'$.

(3) For any vertex v, if K_v is the up-contour (resp. down-contour) of v in $\boldsymbol{M}$ that intersects an edge vw and K'_v is the up-contour (resp. down-contour) of v in $\boldsymbol{M}'$ that also intersects vw, then $\bar{F}_{\boldsymbol{M}}(K_v) = \bar{F}_{\boldsymbol{M}'}(K'_v)$.

Proof. It suffices to prove the lemma assuming that $\boldsymbol{M}'$ is the result of applying a single simple stretching at a contour K of $\boldsymbol{M}$. The more general statement then follows inductively.

(1) Suppose K is a red contour of $\boldsymbol{M}$ and therefore $\delta > 0$. The endpoints of any edge (u, v) that does not intersect K lie at the same side of K

and therefore their heights are modified (if at all) by the same amount. For any (oriented) edge (u,v) of $\boldsymbol{M}$ that intersects K, $u \in K^{\text{out}}$ and $v \in K^{\text{in}}$. Using the fact that $\delta > 0$, we obtain $h'(u) = h(u) < h(v) < h(v) + \delta = h'(v)$.

By part (1), the lower and upper links of every vertex remain the same after stretching. Therefore, regular vertices, extrema, and saddles of $\boldsymbol{M}$ remains the same in $\boldsymbol{M'}$. Furthermore, since K is taken to be generic, up- and down-contour(s) of any saddle are all contained either in K^{in} or all in K^{out}. Therefore the type of a saddle (positive or negative) is not affected by the stretching operation. The case where K is blue is similar.

For part (3), assume that K is red (the other case is symmetric). Let K_v be an up-contour of v and assume first that $K \subset K_v^{\text{in}}$. If K_v is blue, then no edge (a,b) of $\boldsymbol{M}$ that intersects K_v can also intersect K since this would mean that $b \in K_v^{\text{out}}$ and $b \in K^{\text{in}}$ while $K \subset K_v^{\text{in}}$. Therefore in this case all the triangles in $F_{\boldsymbol{M}}(K_v)$ are contained in K^{out} and remain unaffected by the stretching at K. Therefore, K_v remains at the same height in $\boldsymbol{M'}$ and continues to be an up-contour of v. Since it also continues to intersects the edge vw, it coincides with K'_v and in particular $\bar{F}_{\boldsymbol{M}}(K_v) = \bar{F}_{\boldsymbol{M'}}(K'_v)$.

If K_v is red, then $V^-(K_v) \subset K_v^{\text{out}}$. Thus $h'(u) = h(u) < h(v)$ for any $u \in V^-(K_v)$ while $h'(u) \geq h(u) > h(v)$ for all $u \in V^+(K_v)$. This means that the triangles in $F_{\boldsymbol{M}}(K_v)$ all intersect some up-contour of v in $\boldsymbol{M'}$. Since the triangles incident to the edge vw are among these triangles, the mentioned up-contour has to be K'_v.

The case where K is contained in K_v^{out} is handled similarly and the argument for down-contours is done symmetrically. $\square$

Corollary 3.1. *If $\boldsymbol{M'}$ is a stretching of $\boldsymbol{M}$, then $\mathcal{C}_{\boldsymbol{M}}$ (resp. $\mathcal{J}_{\boldsymbol{M}}$, $\mathcal{S}_{\boldsymbol{M}}$) and $\mathcal{C}_{\boldsymbol{M'}}$ (resp. $\mathcal{J}_{\boldsymbol{M'}}$, $\mathcal{S}_{\boldsymbol{M'}}$) are combinatorially identical.*

The stretching operation has a natural interpretation in terms of the contour trees. Let K be a contour carried by an edge uv of $\mathcal{C}$. Let v be the parent of u in $\mathcal{C}$ (with respect to the root v_∞ of $\mathcal{C}$). Let $\mathcal{C}^u$ denote the subtree of $\mathcal{C}$ rooted at u. It can be verified that the vertices of $\mathcal{C}^u$ are exactly the vertices of $\boldsymbol{M}$ that are contained in K^{in}. In particular, the leaves of $\mathcal{C}^u$ are the extrema in K^{in}. Furthermore, if K' is a contour carried by an edge of $\mathcal{C}^u$, then $K' \subset K^{\text{in}}$. Thus stretching $\boldsymbol{M}$ at K by δ corresponds to adding δ to the heights of all vertices in $\mathcal{C}^u$. Since stretching $\boldsymbol{M}$ at equivalent contours, i.e., contours carried by the same edge of $\mathcal{C}$, results the same stretched terrain, one can regard stretching $\boldsymbol{M}$ at a contour K by

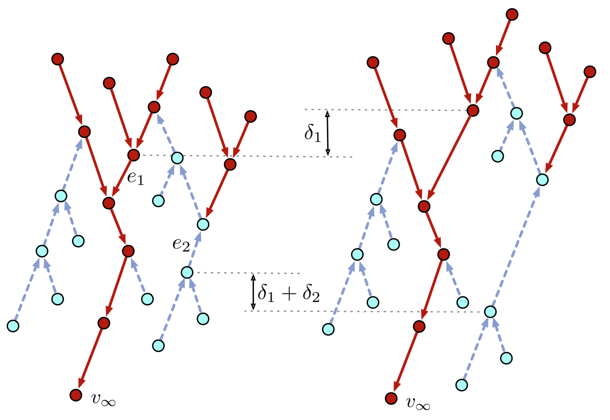


Figure 8: The effect of stretching on contour tree: on the left a red edge e_1 and a blue edge e_2 of a contour tree $\mathcal{C}$ are respectively stretched by $\delta_1 > 0$ and $\delta_2 < 0$. The heights of vertices in the subtree rooted at the tail of e_1 are increased by δ_1. Those vertices in the subtree rooted at tail of e_2 are displaced by an extra δ_2 amount.

δ as associating an offset amount δ to the edge of $\mathcal{C}$ that carries K. The total change to $h(v)$ for a vertex v after a series of stretches is then determined by the sum of the offsets over the edges on the path from v to v_∞ in $\mathcal{C}$ (See Figure 8).

Mapping contour queries. Part (3) of Lemma 3.1 along with the discussion on up- and down-contours implies that if one canonically represent a generic contour K by, say, the appropriate up-contour of $v = v^-(K)$, then $\bar{F}(K)$ is identical to $\bar{F}(K')$ where K' is the corresponding up-contour of v in the stretching $\boldsymbol{M'}$ of $\boldsymbol{M}$; the same is true for the down-contour of $v^+(K)$. In other words, a generic contour K in $\boldsymbol{M}$ can be mapped to one in $\boldsymbol{M'}$ that intersects the same set of triangles in the same order. Since a contour is represented by a pair (f, ℓ), we need to find the lower (or upper) supporting vertex of $K_{\boldsymbol{M}}(f, \ell)$ to map it to $\boldsymbol{M'}$. The following lemma shows how join and split trees can be used to do this.

Lemma 3.2. *Let $\boldsymbol{M}$ be a terrain with augmented join and split trees $\mathcal{J}$ and $\mathcal{S}$. Let K be a generic blue (resp. red) contour in $\boldsymbol{M}_\ell$ and let u be a vertex in $V^-(K)$ (resp. $V^+(K)$). Then the lower (resp. upper) supporting vertex of K is the highest blue (resp. lowest red) vertex v on the path from u to the root of $\mathcal{J}$ (resp. $\mathcal{S}$) that satisfies $h(v) < \ell$ (resp. $h(v) > \ell$).*

Proof. We only prove the lemma for the case where K is blue; the other case is similar. Let $v = v^-(K)$ be the lower supporting vertex of K, and let $u \in V^-(K)$. Let $r_{\mathcal{J}}$ denote the root of $\mathcal{J}$, i.e. the highest maximum in $\boldsymbol{M}$. We show that v is the highest blue vertex with height less than ℓ on the path between u and $r_{\mathcal{J}}$ in $\mathcal{J}$.

Let f be a triangle incident upon u that intersects K and let K_u be the up-contours of u that intersect f. By Lemma 2.1 u is blue and therefore so is K_u. For any α, let U_α be the connected component of $\boldsymbol{M}_{\leq\alpha}$ that contains u. At $\alpha = h(u)$, u lies on the boundary of U_α. By the definition of $\mathcal{J}$, as α grows larger, every time the boundary of U_α reaches a new vertex w, this vertex is turned into the parent of the highest vertex of U_α in $\mathcal{J}$. The color of w is determined by the color of the contour that is the boundary component of U_α that reaches w. We argue that v is the highest blue vertex on P with $h(v) < \ell$: being a connected component of a sublevel set with a blue contour on its boundary, $U_{h(v)+\varepsilon}$ has to be bounded by a single blue contour K_0 (which is the up-contour of v equivalent to K), and zero or more red contours $K_1, \ldots, K_r$ where $K_i \subset K_0^{\text{in}}$ for all $i = 1, \ldots, r$. When α sweeps the interval $[h(v) + \varepsilon, \ell]$, the blue boundary component of U_α continuously changes from K_0 to K without passing through any vertices (since v is the lower supporting vertex of K). This means that all other vertices encountered at the boundary of U_α (and therefore added to P between v and $r_{\mathcal{J}}$) are met by one of the red boundary components of U_α and are therefore all red. □

Let (f, ℓ) be the input to a contour query. Let v_0 (resp. v_1) be the lowest (resp. highest) vertex of f. $K = K(f, \ell)$ intersects the edge v_0v_1. By Lemma 2.1, if K is blue then v_0 is blue, and if K is red then v_1 is red. By Lemma 3.2, if K is blue, then $v^-(K)$ is the highest blue vertex in $\mathcal{J}$ between v_0 and the root with height less than ℓ, and if K is red, then $v^+(K)$ is the lowest red vertex in $\mathcal{S}$ between v_1 and the root with height greater than ℓ. If the colors of v_0 and v_1 agree, K has to be of the same color. Otherwise, by Lemma 2.1, v_0 must be blue, and v_1 red, and the color of contours that intersect v_0v_1 flips once from blue to red at a unique height which we denote by $\eta(v_0v_1)$. The following lemma is relatively straightforward:

Lemma 3.3. *$\eta(v_0v_1)$ is the height of the saddle that is the lowest common ancestor of v_0 and v_1 in the contour tree of $\boldsymbol{M}$.*

One can compute $\eta(e)$ of all terrain edges e with non-matching endpoint colors in $O(\text{Sort}(N))$ I/Os using the so-called time-forward processing technique [10].

We can compute one of $v^-(K)$ or $v^+(K)$ as follows: We preprocess the blue (resp. red) vertices of $\mathcal{J}$ (resp. $\mathcal{S}$) in a persistent B-tree [6, 16] $B_{\mathcal{J}}$ (resp. $B_{\mathcal{S}}$), using their heights as the key. Algorithm ComputeB$_{\mathcal{J}}$ (Figure 9 left) computes $B_{\mathcal{J}}$. A similar algorithm is used to compute $B_{\mathcal{S}}$. The recorded step number $\tau(v)$ is later used to access the instance $B_{\mathcal{J}}[\tau(v)]$ of $B_{\mathcal{J}}$ in which v was first inserted. $B_{\mathcal{J}}[\tau(v)]$ stores the sequence of blue vertices on the path from v to the root, sorted by their heights. Algorithm TranslateQuery (Figure 9 right) computes $v^-(K)$ if K is blue and $v^+(K)$ if K is red. It then returns a pair (f, ℓ') where ℓ' is the height in $\boldsymbol{M}'$ of the up-contour $v^-(K)$ in the former case, or that of the down-contour of $v^+(K)$ in the latter case. We have thus shown the following:

Theorem 3.1. *Let $\boldsymbol{M} = (M, h)$ be a terrain of size N and let $\boldsymbol{M}' = (M, h')$ be a stretching of $\boldsymbol{M}$. $\boldsymbol{M}$ and $\boldsymbol{M}'$ can be preprocessed using $O(\text{Sort}(N))$ I/Os into a linear size data structure that converts, in $O(\log_B N)$ I/Os, any contour query (f, ℓ) on $\boldsymbol{M}$ into a contour query (f, ℓ') on $\boldsymbol{M}'$ such that $\bar{F}_{\boldsymbol{M}}(K(f, \ell)) = \bar{F}_{\boldsymbol{M}'}(K(f, \ell'))$.*

4 Bounding Fragmentations by Stretching

We now describe an algorithm for computing a stretching of $\boldsymbol{M}$ that enables us to answer contour queries quickly.

Level-ordering and fragmentations. Let $\rho : F(M) \to \mathbb{N}$ be an injective *rank* function on the triangles of $\boldsymbol{M}$. A generic contour K *agrees* with ρ if the circular list $\bar{F}(K)$ can be written as $\langle f_1, \ldots, f_k, f_1 \rangle$, such that $\rho(f_1) < \cdots < \rho(f_k)$. The function ρ is a *level-ordering* for $\boldsymbol{M}$ if the following two conditions hold:

C1. Any generic contour K agrees with ρ.

C2. Let K, K' two generic contours of a level set ℓ, let $f_1, f_2 \in F(K)$, and let $f_1', f_2' \in F(K')$. If $\rho(f_1) < \rho(f_1') < \rho(f_2)$ then $\rho(f_1) < \rho(f_2') < \rho(f_2)$.

Given a level-ordering ρ on $\boldsymbol{M}$, let $F_\rho(\boldsymbol{M}_\ell)$ denote the list of triangles that intersect $\boldsymbol{M}_\ell$, ordered by ρ. A *fragment* of a generic contour K in $\boldsymbol{M}_\ell$ is a maximal subsequence $f_1, \ldots, f_r$ of $F_\rho(\boldsymbol{M}_\ell)$ consisting of triangles in $F(K)$. $\boldsymbol{M}$ may have several level-orderings, but we focus on the level-ordering computed by Agarwal *et al.* [2]. As mentioned in Introduction, the only obstacle in achieving efficient contour queries using the algorithm of Agarwal *et al.* [2] is the lack of any control over the number of fragments of contours under the computed level-ordering. Theorem 3.1 shows that if some stretching $\boldsymbol{M}'$ of the input terrain $\boldsymbol{M}$ avoids this obstacle, at least for the contours of $\boldsymbol{M}'$ that are mapped from $\boldsymbol{M}$, then it can be used in place of $\boldsymbol{M}$. This section proves the existence of such a stretching and gives an algorithm to compute it in

Algorithm: ComputeB$_{\mathcal{J}}$

```
1  Compute an Euler tour $\Pi$ of $\mathcal{J}_{\boldsymbol{M}}$ starting at its root
2  for a vertex $v$ visited at step $i$ of $\Pi$ do
3      if $v$ is blue
4          if this is the first visit to $v$
5              insert $v$ into $B_{\mathcal{J}}$; assign time-stamp $i$
               to $B_{\mathcal{J}}$; and record $\tau(v) = i$
6          if this is the last visit to $v$
7              delete $v$ from $B_{\mathcal{J}}$
```

Algorithm: TranslateQuery(f, ℓ)

```
1  $v_0 \leftarrow$ lowest vertex of $f$.
2  $v_1 \leftarrow$ highest vertex of $f$.
3  if $v_1$ is blue  or  $\eta(v_0 v_1) < \ell$
4      $u \in B_{\mathcal{J}}[\tau(v_0)]$ highest vertex with $h(u) < \ell$
5      return  $(f, h'(u) + \varepsilon)$
6  else
7      $u \in B_{\mathcal{S}}[\tau(v_1)]$ lowest vertex with $h(u) > \ell$
8      return  $(f, h'(u) - \varepsilon)$
```

Figure 9: Algorithms for terrain preprocessing (left) and conversion of contour queries (right).

$O(\mathrm{Sort}(N))$ I/Os. We begin by introducing the notion of ascending and descending cut-trees, which are used by Agarwal *et al.*'s level-ordering algorithm.

Ascending and descending cut-trees. For a terrain edge $e = (u, v)$, let $\lambda^{+}(e)$ be the first outgoing edge of v in clockwise order after e, and let $\lambda^{-}(e)$ be the first incoming edge of u in clockwise order before e. A *rightmost ascending* (*leftmost descending*) path P from v_1 to v_k is a path $v_1 \dots v_k$ where for each $1 < i < k$, $(v_i, v_{i+1}) = \lambda^{+}((v_{i-1}, v_i))$ (resp. $(v_{i+1}, v_i) = \lambda^{-}((v_i, v_{i-1}))$. An ascending (resp. descending) path $P = v_1 \dots v_k$ is *maximal* if v_k is a maximum (resp. minimum). Let $V_{\ominus}, V_{\oplus}$ respectively be the sets of negative and positive saddles of $\boldsymbol{M}$. For each positive saddle s, let $P_1(s)$ and $P_2(s)$ be two maximal rightmost ascending paths from s such that the vertex following s on each of P_1 and P_2 is the counterclockwise first neighbor of s in a distinct upper link component of s. Similarly, for each negative saddle s, let $P_1(s)$ and $P_2(s)$ be maximal leftmost descending paths from s such that the vertex following s in each of P_1 and P_2 is the clockwise first neighbor of s in a distinct lower link component of s. For each saddle s, the subgraph $P(s) = P_1(s) \cup P_2(s)$ of M is uniquely determined. We call the subgraphs $\hat{\boldsymbol{T}} = \cup_{s \in V_{\oplus}} P(s)$ and $\check{\boldsymbol{T}} = \cup_{s \in V_{\ominus}} P(s)$ of M the *ascending* and the *descending cut-trees*, respectively, of $\boldsymbol{M}$. Agarwal *et al.* [2] proved that $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$ are indeed both trees and include all maxima and minima, respectively, of $\boldsymbol{M}$ among their vertices. They also showed that they can be computed in $O(\mathrm{Sort}(N))$ I/Os.

Lemma 3.1 implies that a stretching $\boldsymbol{M}'$ of $\boldsymbol{M}$ has exactly the same ascending and descending cut-trees as $\boldsymbol{M}$. In [2], $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$ are used to compute a specific level-ordering of a terrain $\boldsymbol{M}$ in $O(\mathrm{Sort}(N))$ I/Os. This, in conjunction with the level-ordering construction of [2] implies the following:

Corollary 4.1. *Let $\boldsymbol{M}'$ be a stretching of $\boldsymbol{M}$. A level-ordering on $\boldsymbol{M}$ can be computed in $O(\mathrm{Sort}(N))$ I/Os that is also a level-ordering on $\boldsymbol{M}'$.*

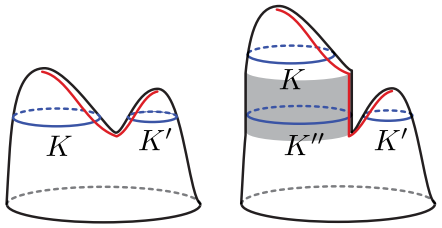


Figure 10: The effect of stretching on contour fragmentation

The specific construction of the level-ordering ρ in [2] entails a characterization of the fragments of contours in relation to $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$. This is summarized in the following lemma. See [2] for details and proof.

Lemma 4.1. *Let K and K' be contours in $\boldsymbol{M}_{\ell}$, and let ρ be the computed level-ordering of $\boldsymbol{M}$. Then a fragment of K is immediately followed by a fragment of K' in $F_{\rho}(\boldsymbol{M}_{\ell})$ if and only if K and K' are connected by a polygonal path contained either in $\hat{\boldsymbol{T}} \cap \boldsymbol{M}_{\leq \ell}$ or in $\check{\boldsymbol{T}} \cap \boldsymbol{M}_{\geq \ell}$.*

4.1 The intuitive idea.

The intuition as to why stretching can help us with fragmentations is as follows. Consider the cartoon terrain of Figure 10. On the left a terrain $\boldsymbol{M}$ with two hills is shown. The saddle between the two hills contributes a path to the ascending cut-tree which ends at the peaks of the two hills. The two contours K and K' on the shown level set are connected by the ascending cut-tree through their corresponding sublevel set. By Lemma 4.1, this means that some fragment of each of these contours is followed by a fragment of the other in the level-ordered list of triangles that intersect their level. Let us assume that K is split into two fragments with K' in between them. On the right, $\boldsymbol{M}$ is stretched at one of the up-contours of the saddle so that the

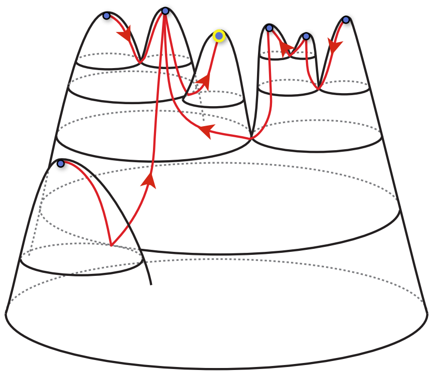


Figure 11: Rooting of the ascending cut-tree in a terrain with only positive saddles.

piece of the terrain bounded by that up-contour (the portion that contains K) is lifted above the other peak. Let $\boldsymbol{M}'$ be this stretched terrain. K is no longer fragmented by K' in $\boldsymbol{M}'$. Although a contour K'' in the shaded area of $\boldsymbol{M}'$ is fragmented by K', no contour of $\boldsymbol{M}$ is mapped to K'', so the fragmentation of K'' is no concern of us.

Next, consider the case when $\boldsymbol{M}$ does not have any negative saddles., i.e., $\boldsymbol{M}$ has only one minimum v_∞; see Figure 11. The descending cut-tree of this terrain is therefore trivial. Each saddle of the ascending cut-tree can be regarded as the midpoint of a *hyper-edge* of the cut-tree that connects the two maxima at which the ascending paths from that saddle end. This step compresses the ascending-cut tree to another tree on the maxima of $\boldsymbol{M}$. Let us pick an arbitrary maximum as the root and orient the hyper-edges toward it. Of the two ascending paths out of each saddle, one ends at the head and the other at the tail of the oriented hyper-edge. We call ascending paths correspondingly as *tail* or *head* ascending paths. It was proved in [2] that the restriction of the ascending cut-tree to the inside of each (red) contour is connected. This in conjunction with the fact that each maximum can be the tail of at most one hyper-edge (though it could be the head of many hyper-edges), implies that each contour of the terrain intersects at most one tail ascending path. Now, we can use our observation from Figure 10 as follows: At each saddle, the up-contour crossing the head ascending path is stretched high enough so that in the final terrain every points inside this up-contour ends up higher than every point in the inside of the other up-contour of the saddle.

In the resulting terrain, any contour can only connect through the sublevel set to another contour of its level via the ascending cut-tree only by following the at most one tail ascending path that crosses it. By Lemma 4.1, this amounts to at most one fragmentation per contour. Terrains with only negative saddles can be handled in a similar way. However, the problem becomes more complicated when both positive and negative saddles are present because in this case blue positive and red negative saddles appear; see Figure 12. The figure on the right demonstrates the idea behind dealing with a red negative saddle. Notice how the outer down contour of the saddle is stretched upward so that the entire pit of the volcano rises above all the points outside of the stretched contour. This eliminates the possibility of the red down-contour of the saddle to be fragmented by its blue down-contour (the one inside the volcano). Notice that in this case the saddle itself also moves up.

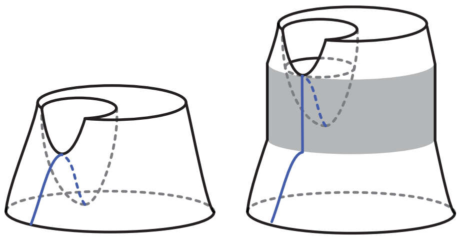


Figure 12: Stretching at a red negative saddle

4.2 The stretching algorithm

We now present our algorithm for stretching $\boldsymbol{M}$. The stretched terrain $\boldsymbol{M}'$ is derived by stretching exactly one up- or -down-contour of every saddle of $\boldsymbol{M}$. We thus characterize the particular up- or down-contour of each saddle s of $\boldsymbol{M}$ that is stretched together with the offset parameter $\delta(s)$.

Our algorithm uses *compressed* versions $\hat{T}$ and $\check{T}$ of $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$ respectively. Specifically, we define $\hat{T} = (\hat{V}, \hat{E})$ and $\check{T} = (\check{V}, \check{E})$ where $\hat{V}$ is the set of maxima and $\check{V}$ is the set of minima of $\boldsymbol{M}$. For $u, v \in \hat{V}$, $uv \in \hat{E}$ if there is a positive saddle s for which $P(s)$ ends in u and v, in which case we label the edge uv with s. $\check{T}$ is defined symmetrically. Furthermore, we pick root vertices for $\hat{T}$ and $\check{T}$ and orient their edges toward their roots. For the root $\check{r}$ of $\check{T}$ we pick v_∞ and the root $\hat{r}$ of $\hat{T}$ is chosen arbitrary among the maxima reachable from v_∞ in the contour tree $\mathcal{C} = \mathcal{C}_{\boldsymbol{M}}$ through a path of red edges.[4] See Figure 13; $\hat{T}$ and $\check{T}$ shown in dotted black.

[4]The asymmetry here is due to the fact that we cannot lower the height of any vertex below that of v_∞. The choice of v_∞ as the root of $\check{T}$ then puts restrictions on the choice of $\hat{r}$.

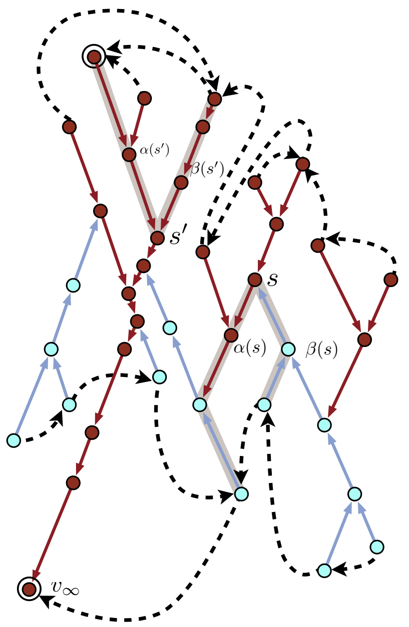


Figure 13: Illustration of $\alpha(s)$, $\beta(s)$, head(s), and tail(s) for two choices of a saddle s on a contour tree. In each case, the paths from s to head(s) and tail(s) are highlighted. The edges of $\hat{T}$ and $\check{T}$ are drawn in dashed lines. Vertices of $\hat{T}$ are the maxima and those of $\check{T}$ are the minima.

For an edge uv of $\hat{T}$ (resp. $\check{T}$) labeled with positive (resp. negative) saddle s, if v is the parent of u with respect to $\hat{r}$ (resp. $\check{r}$), then we set tail$(s) = u$ and head$(s) = v$. There is a one-to-one correspondence between the positive saddles and the edges of $\hat{E}$, and there is a similar correspondence between negative saddles and $\check{E}$. For any saddle s, we distinguish two of the three neighbors of s in $\mathcal{C}$ as $\alpha(s)$ and $\beta(s)$: $\alpha(s)$ is the neighbor on the path in $\mathcal{C}$ from s to head(s) and $\beta(s)$ is the neighbor on the path from s to tail(s) (See Figure 13) For any saddle s, let $n(s)$ denote the number of leaves in the subtree $\mathcal{C}^{\beta(s)}$ of $\mathcal{C}$ rooted at $\beta(s)$.

We are now ready to characterize the stretch associated to each saddle s: the stretch associated with s is applied to the up- or down-contour of s carried by the edge $s\alpha(s)$ of $\mathcal{C}$. The amount of the stretch is given by

$$\delta(s) = \begin{cases} H \cdot n(s) & \text{if } s \text{ is red,} \\ -H \cdot n(s) & \text{if } s \text{ is blue,} \end{cases}$$

where $H = \max_{u,v \neq v_\infty} |h(u) - h(v)|$.

For any vertex v, let $\Pi(v)$ denote the set of vertices in the path from v to the root v_∞ in $\mathcal{C}$. In our algorithm, the stretches that affect the height of a vertex v are precisely those associated with saddles s for which $s\alpha(s)$ is an edge in $\Pi(v)$. We use the shorthand $\Sigma(v)$ for the set of all such saddles on $\Pi(v)$ and write $\Sigma_B(v)$ and $\Sigma_R(v)$ to respectively represent the sets of blue and red saddles in $\Sigma(v)$. For any saddle s, the edge $s\alpha(s)$ has the same color as s itself. Thus defining

$$\begin{aligned} \Delta(v) &= \sum_{s\in\Sigma(v)} \delta(s) \\ &= H \cdot \left(\sum_{s\in\Sigma_R(v)} n(s) - \sum_{s\in\Sigma_B(v)} n(s) \right), \end{aligned} \tag{4.1}$$

one has $h'(v) = h(v) + \Delta(v)$.

This completes the description of our stretching algorithm. We describe in Section 5 how the stretched terrain can be computed using $O(\text{Sort}(N))$ I/Os.

4.3 Bounding fragmentations in $\boldsymbol{M}'$ We now argue that if a contour $K_{\boldsymbol{M}}(f,\ell)$ on $\boldsymbol{M}$ maps to $K_{\boldsymbol{M}'}(f,\ell')$ in $\boldsymbol{M}'$, then $K_{\boldsymbol{M}'}(f,\ell')$ has O(1) fragments in $F_\rho(\boldsymbol{M}'_{\ell'})$. To prove this, we need to mention some structural properties of contour trees and relationship between cut trees and contour trees.

Properties of contour trees. As earlier, let $\mathcal{C} = \mathcal{C}_{\boldsymbol{M}}$ be the contour tree of $\boldsymbol{M}$, rooted at v_∞. We also assume that vertices and edges of $\mathcal{C}$ are colored red or blue as specified in Section 2. Each vertex v of $\mathcal{C}$ has one, two, or three neighbors in $\mathcal{C}$. A neighbor u of v is an *up-neighbor* if $h(u) > h(v)$ and a *down-neighbor* if $h(u) < h(v)$. An edge of $\mathcal{C}$ connecting v to an up-neighbor or down-neighbor of v is referred to as an *up-going* or *down-going* edge of v, respectively. An *up-going path* from u to v in $\mathcal{C}$ is a path along the vertices of which the height function is increasing. A *down-going path* is defined similarly. If we orient the edge of $\mathcal{C}$ toward v_∞ (the root of $\mathcal{C}$) all red edges will be oriented from the higher endpoint to the lower and all blue edges will be oriented from the lower endpoint to the higher (See Figure 14).

The leaves of $\mathcal{C}$ (vertices with only one neighbor) are precisely the extrema of $\boldsymbol{M}$. A maximum has one down-neighbor and a minimum has one up-neighbor. Each regular vertex has one up- and one down-neighbor.

Any saddle s of $\boldsymbol{M}$ has three neighbors in $\mathcal{C}$. If s is positive it has two up-neighbors and one down-neighbor. If s is a red positive saddle, all the edges incident to s are red and if it is a blue positive saddle, then one of its up-going edges is red and its other two incident edges are blue. The case of negative saddles are symmetric with 'red' exchanged with 'blue' and 'up' with 'down'.

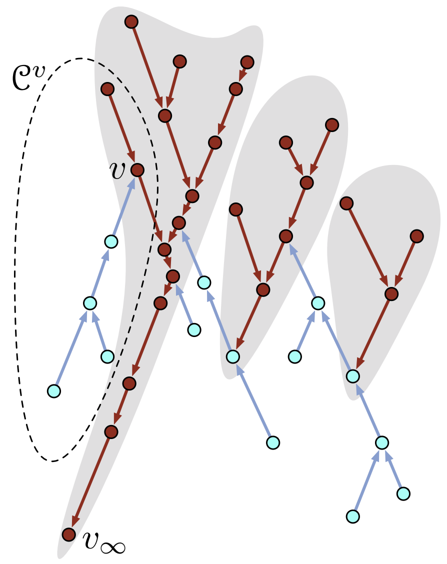


Figure 14: Colored contour tree rooted at v_∞; The red components are shaded. The subtree $\mathcal{C}^v$ rooted at the indicated vertex v is circled by a dashed line.

The *subtree* of $\mathcal{C}$ rooted at a vertex v is denoted by $\mathcal{C}^v$. Consider the forest $\mathcal{C}_R \subseteq \mathcal{C}$ induced by the red edges of $\mathcal{C}$. We call each connected component of $\mathcal{C}_R$ a *red component* of $\mathcal{C}$. We define the forest $\mathcal{C}_B$ induced by the blue edges of $\mathcal{C}$ and the corresponding *blue components* similarly. We call red or blue components the *color components* of $\mathcal{C}$. Observe that if vertex v has a red down-going edge incident to it, then all its up-going edges will be red as well. Symmetrically, if a vertex has a blue up-going edge, all its down-going edges will be blue. This means that red components of $\mathcal{C}$ are *up-ward closed*, i.e., all vertices reachable from a red vertex through up-going paths are also red. Similarly, blue components are *down-ward closed*.

By definition any two red components of $\mathcal{C}$ are vertex disjoint and the same holds for any two blue components. A vertex shared between a red and a blue component must be either a red negative saddle or a blue positive saddle. Since $\mathcal{C}$ is acyclic, a red and a blue component of $\mathcal{C}$ can have at most one vertex in common, in which case we call them *neighboring components* in $\mathcal{C}$. Let v be a vertex common between two color components C and C' of opposite colors. Then either C or C' (but not both) is a subgraph of $\mathcal{C}^v$. If C is a subgraph of $\mathcal{C}^v$, we say C is a *child component* of C' and C' is the *parent component* of C. Note that in this case v is the closest vertex of C to the root v_∞ in $\mathcal{C}$ and is therefore called the *root* of the color component C.

Recall that v_∞ is a red vertex. Therefore, it will be contained in a red component which we call the *root component* of $\mathcal{C}$. The children of a red component R are blue components connected to R through red negative saddles and the children of a blue component B are red components connected to B through blue positive saddles. Thus the root of a blue component is a red negative saddle and the root of a red component is a blue positive saddle. Besides the root, all vertices of a blue component are blue and all vertices in a red component are red.

Cut trees and contour trees. We consider the images of $\hat{\boldsymbol{T}}$ or $\check{\boldsymbol{T}}$ under the quotient map $\pi_{\boldsymbol{M}}$. Observe that monotone paths on $\boldsymbol{M}$ are mapped into $\mathcal{C}$ injectively. Let s be a positive saddle. The vertices head(s) and tail(s) of $\hat{T}$ are the maxima of $\boldsymbol{M}$ in which $P(s)$ ends. Let $P_1(s)$ and $P_2(s)$ respectively be the ascending paths from s to head(s) and tail(s) that together compose $P(s)$. Since $P_1(s)$ and $P_2(s)$ are ascending, their images $\mathcal{P}^+(s) = \pi_{\boldsymbol{M}}(P_1(s))$ and $\mathcal{P}^-(s) = \pi_{\boldsymbol{M}}(P_2(s))$ are *up-going* paths in $\mathcal{C}$, i.e., the heights of vertices along these paths in $\mathcal{C}$ monotonically increase. Similarly, for a negative saddle s, we get *down-going* paths $\mathcal{P}^+(s)$ and $\mathcal{P}^-(s)$ in $\mathcal{C}$ from s to head(s) and tail(s) respectively. For any saddle s, let $\mathcal{P}(s) = \mathcal{P}^+(s) \cup \mathcal{P}^-(s)$. Thus the image of the entire $\hat{\boldsymbol{T}}$ under the map $\pi_{\boldsymbol{M}}$ is a subtree $\hat{\mathcal{T}} = \bigcup_{s \in V_\oplus} \mathcal{P}(s)$ of $\mathcal{C}$. $\check{\mathcal{T}}$ is defined similarly.

Stretching of saddles. Next, we bound relative heights of saddles in the stretched terrain $\boldsymbol{M'}$, which in turn will bound the number of fragments in a contour in $\boldsymbol{M'}$. We need the following two lemmas to prove Lemma 4.4, which states the main property of the saddles in $\boldsymbol{M'}$.

LEMMA 4.2. *For any blue positive saddle s, $\alpha(s)$ and $\beta(s)$ are the up-neighbors of s respectively reachable through the blue and red up-going edges at s. For any red ngative saddle s, $\alpha(s)$ and $\beta(s)$ are the down-neighbors of s respectively reachable through the red and blue down-going edges at s.*

Proof. We prove the lemma for blue positive saddles; the other case is symmetric. Let s be a blue positive saddle and let u_R and u_B be the up-neighbors of s respectively reachable through the red and the blue up-going edges of s. Let R and B respectively be the red and blue components of $\mathcal{C}$ that contain the edges su_R and su_B. Thus s is the vertex shared between R and B. Since s is a blue positive saddle, B is the parent and R is the child component. As discussed above, one endpoint v_B of the $\hat{T}$ edge labeled s is in $\mathcal{C}^{u_B}$ and the other v_R is in $\mathcal{C}^{u_R}$. We claim that v_R = tail(s) and v_B = head(s). To see this, we observe that no ascending path initiated in $\mathcal{C}^{u_R}$ can reach s since the only way for such a path to reach s is through u_R which is an up-neighbor of s. Thus the two endpoints of the $\hat{T}$ edge labeled s' for any $s' \in \mathcal{C}^{u_R}$ are contained in $\mathcal{C}^{u_R}$. Since $\hat{r}$ (the root of $\hat{T}$) is in the root red component of $\mathcal{C}$, v_R can reach $\hat{r}$ only through the parent component B of R and since $\hat{T}$ edge labeled s is the only edge of $\hat{T}$ that has exactly one endpoint in $\mathcal{C}^{u_R}$, v_B has to be head(s). This means that $u_B = \alpha(s)$ and $u_R = \beta(s)$. □

Lemma 4.3. *Let s and s' be two red (resp. blue) saddles so that s, $\alpha(s)$, s', and $\alpha(s')$ are all on the same up-going (resp. down-going) path Π_0 made of only red (resp. blue) edges. Then $\mathcal{C}^{\beta(s)}$ and $\mathcal{C}^{\beta(s')}$ are disjoint subtrees of $\mathcal{C}$.*

Proof. We only prove the red version of the lemma. Let R be the red component of $\mathcal{C}$ that contains Π_0 and let r be the root of R. Being the root of a red component of $\mathcal{C}_{\boldsymbol{M}}$, r is either v_∞ or is a blue positive saddle. The path Π_0 can be extended to an up-going path Π_1 from r. For any vertex $v \in R$, $\Pi(v)$ must pass through r. Now, depending on s being positive or negative, either both $\alpha(s)$ and $\beta(s)$ are up-neighbors of s or they are both down-neighbors of s. Thus at most one of them can be present on Π_1. Since $\alpha(s)$ is on Π_1, $\beta(s)$ cannot be on Π_1. Similarly, $\beta(s')$ cannot be on Π_1. Since $\beta(s)$ is a neighbor of s and s is on a path Π_1 to r, $\Pi(\beta(s))$ has to go through s. Similarly, $\Pi(\beta(s'))$ goes through s'. But this means that neither of $\beta(s)$ and $\beta(s')$ can be on the path from the other to r. □

Lemma 4.4. *For any positive (resp. negative) saddle s, and for any vertices $u \in \mathcal{P}^-(s)$ and $v \in \mathcal{P}^+(s)$ with $v \neq s$, $h'(v) > h'(u)$ (resp. $h'(v) < h'(u)$).*

Proof. We prove the lemma for positive saddles. The proof for negative saddles is symmetric. There are two cases depending on the color of s.

Blue saddle. We first consider the case in which s is a blue positive saddle. Thus s is the root of a red component R of $\mathcal{C}$ whose parent B is a blue component. We name the up-neighbors of s in B and R as u and u' respectively. By Lemma 4.2, $u = \alpha(s)$ and $u' = \beta(s)$. Let $v =$ head(s) and $v' =$ tail(s) as determined by $\hat{T}$ (See Figure 15). Since R is upward closed, v' is a vertex of R and the path $\Pi(v')$ passes through s (the root of R) and u which is the parent of s in $\mathcal{C}$. Since $\mathcal{C}$ is the contour tree of both $\boldsymbol{M}$ and $\boldsymbol{M}'$ and since $\mathcal{P}^+(s)$ and $\mathcal{P}^-(s)$ are up-going paths in $\mathcal{C}$, both h and h' are increasing along both of these paths. It is therefore sufficient to prove that $h'(v') < h'(u)$. Since u is

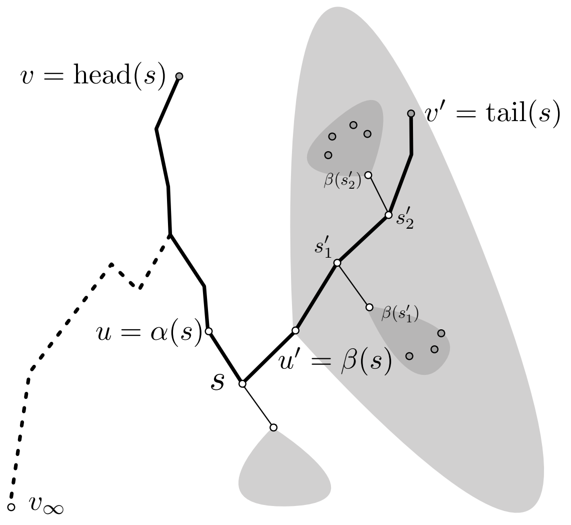


Figure 15: Illustrations for the proof of Lemma 4.4; the blue positive saddle case

a vertex of $\Pi(v')$, $\Sigma(u) \subseteq \Sigma(v')$ and therefore by Eq (4.1) any saddle $s' \in \Sigma(u)$ contributes the same amount to both $\Delta(u)$ and $\Delta(v')$. Therefore it is enough to show that $D = \sum_{s' \in S_0} \delta(s') < 0$ where $S_0 = \Sigma(v') \setminus \Sigma(u)$. Since D is an integer multiple of H, if negative, it must be smaller than or equal to $-H$. The statement of the lemma then follows from the fact that $h(u) > h(v') - H$.

Observe now that the only blue saddle in S_0 is s, and since δ is negative for blue saddles and positive for red ones, letting $S_1 = S_0 \setminus \{s\}$, we need to show that

$$\sum_{s' \in S_1} n(s') < n(s). \tag{4.2}$$

Recall that for any saddle s, $n(s)$ is the number of leaves in $\mathcal{C}^{\beta(s)}$. Since $\beta(s) = u'$ which is on the path from any vertex in R to the root r of R, for any $s' \in S_1$, $\mathcal{C}^{\beta(s')}$ is a subtree of $\mathcal{C}^{\beta(s)}$. On the other hand, the saddles in S_1 all lie on an up-going red path, namely $\mathcal{P}^-(s)$, from s to v_R. Therefore by Lemma 4.3 for any two saddles $s', s'' \in S_1$, $\mathcal{C}^{\beta(s')}$ and $\mathcal{C}^{\beta(s'')}$ are disjoint. This means that the total number of leaves in $\mathcal{C}^{\beta(s')}$ over all saddles $s' \in S_1$ are at most equal to the number of leaves in $\mathcal{C}^{\beta(s)}$ and therefore $\sum_{s' \in S_1} n(s') \leq n(s)$. The strictness of inequality in (4.2) follows from the observation that $v_R \neq$ tail(s') for any $s' \in S_1$, since $v_R =$ tail(s), and therefore is contained in $\mathcal{C}^{\beta(s')}$ for no $s' \in S_1$.

Red saddle. We now assume s to be a red positive saddle. As in the previous case, let $u = \alpha(s)$, $u' = \beta(s)$, $v =$ head(s) and $v' =$ tail(s) (See Figure 16).

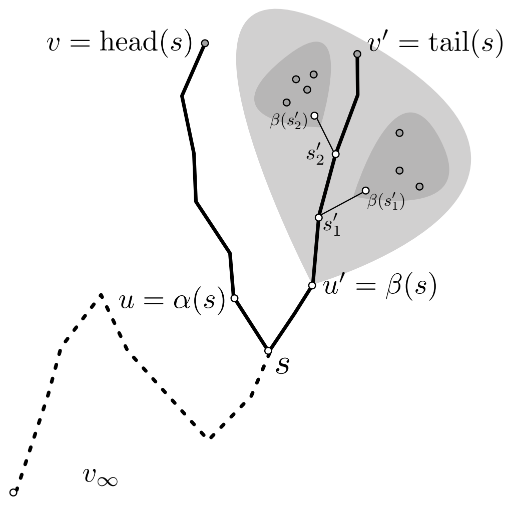


Figure 16: Illustrations for the proof of Lemma 4.4; the red positive saddle case

Both u and u' are up-neighbors of s reached through up-going red edges of s. Again to prove the lemma it is enough to show that $h'(v') < h'(u)$. Notice that this time the parent of s in $\mathcal{C}$ is its down-neighbor w. Thus the paths $\Pi(v')$ and $\Pi(u)$ both pass through s. In other words, all saddles in $\Pi(s)$ contribute equally to $\Delta(u)$ and $\Delta(v')$. Since $\alpha(s) = u \notin \Pi(s)$, $s \notin \Sigma(s)$ while $s \in \Sigma(u)$ and therefore s is the only saddle that contributes to $\Delta(u)$ but not to $\Delta(v')$. Let S_0 be the saddles on $\Pi(v')$ between v' and s, i.e., on $\mathcal{P}^-(s)$, that are contained in $\Sigma(v')$. Note that since s is a red positive saddles, all the saddles on $\mathcal{P}^-(s)$ are red and therefore $\delta(s') > 0$ for any $s' \in S_0$. Since s is a red saddle as well, $\delta(s)$ is also positive. Therefore, as in the previous case, to prove the lemma we show that

$$n(s) > \sum_{s' \in S_0} n(s').$$

Since saddles in S_0 are on an up-going red path, i.e., $\mathcal{P}^-(s)$, Lemma 4.3 implies that for any $s', s'' \in S_0$, $\mathcal{C}^{\beta(s')}$ and $\mathcal{C}^{\beta(s'')}$ are disjoint. On the other hand, $u' = \beta(s)$ and therefore $\mathcal{C}^{\beta(s')}$ is a subtree of $\mathcal{C}^{\beta(s)}$ for any $s' \in S_0$. This means that $n(s) \geq \sum_{s' \in S_0} n(s')$. Again, the inequality follows from observing that v' cannot be in $\beta(s')$ for any $s' \in S_0$ and therefore only contributes to $n(s)$. ☐

Corollary 4.2. *Let s and s' be distinct positive (resp. negative) saddles. If head(s') is a descendent of tail(s) in $\hat{T}$ (resp. $\check{T}$), then for any vertex $u \in \mathcal{P}^-(s') \cup \mathcal{P}^+(s')$ and any vertex $v \in \mathcal{P}^+(s)$ with $v \neq s$, $h'(u) < h'(v)$ (resp. $h'(u) > h'(v)$).*

Proof. Since head(s') is a descendent of tail(s), there is a path from tail(s') to head(s) in $\hat{T}$. Let $s' = s_0, s_1, \ldots, s_k = s$ be labels of the edges on this path. For each $i = 0, \ldots, k$, let $u_i = \text{tail}(s_i)$ and $v_i = \text{head}(s_i)$. Since edges labeled $s_0, \ldots, s_k$ make a path in $\hat{T}$, $u_i = v_{i-1}$ for each $i = 1, \ldots, k$. By Lemma 4.4, $h'(v_i) > h'(u_i)$ for all $i = 1, \ldots, k-1$ and by the same lemma, $h'(u) < h'(v_0)$ and $h'(v_{k-1}) < h'(v)$. Thus $h'(u) < h'(v)$. The proof for negative saddles is similar. ☐

Remark. Lemma 4.4 ensures that in $\boldsymbol{M}'$ a contour K carried by an edge in $\mathcal{P}^+(s)$ of a saddle s is not fragmented by a contours carried by edges of $\mathcal{P}^-(s)$. Corollary 4.2 extends this by saying that K is not fragmented by contours carried by edges of $\mathcal{P}(s')$ for any saddle s' for which head(s') is a descendent of tail(s) in $\hat{T}$ or $\check{T}$.

Bounding fragmentation. We are now ready to bound the number of fragments in the up- or down-contour of any vertex in $\boldsymbol{M}'$.

Theorem 4.1. *Let K be the red down-contour (resp. blue up-contour) of a vertex v in $\boldsymbol{M}'$ and let $\ell = h'(v) - \varepsilon$ (resp. $\ell = h'(v) + \varepsilon$). Then K has at most three fragments in the level-ordered list of triangles in $F(\boldsymbol{M}'_{\ell'})$.*

Proof. Let K be a red down counter of v in $\boldsymbol{M}'_\ell$ and let K' be another contour in $\boldsymbol{M}'_\ell$ such that fragments from K and K' appear consecutively in $F_\rho(\boldsymbol{M}'_\ell)$. By Lemma 4.1 this means that there is either a path in $\hat{\boldsymbol{T}} \cap \boldsymbol{M}'_{\leq \ell}$ or a path in $\check{\boldsymbol{T}} \cap \boldsymbol{M}'_{\geq \ell}$ that connects K and K'. Assume that a path Q of the former type exists. Projecting Q under the map $\pi_{\boldsymbol{M}'}$ into $\mathcal{C}_{\boldsymbol{M}'}$, gives us a walk on $\mathcal{C}_{\boldsymbol{M}'}$ between the points representing K and K' (both points on edges of $\mathcal{C}'_{\boldsymbol{M}}$) that stays below level ℓ throughout. In particular, the points representing K and K' must be connected by a path in $\hat{\mathcal{T}}$ in which all vertices have heights less than ℓ.

Consider any positive saddle s for which $\mathcal{P}^+(s)$ passes through v (and therefore the point representing K) in $\mathcal{C}'_{\boldsymbol{M}}$. By Lemma 4.4 any contour of $\boldsymbol{M}'$ carried by the edges of $\mathcal{P}^-(s)$ belongs to a level strictly smaller than ℓ. Furthermore, by Corollary 4.2, any contour carried by an edge in $\mathcal{P}(s')$, where s' is a positive saddle for which head(s') is a descendent of tail(s) in $\hat{T}$, also belongs to a level strictly lower than ℓ. Therefore, K' can be no such contour. By definition, v can be contained in $\mathcal{P}^-(s_0)$ for only one positive saddle s_0. This is because tail(s_0) has out-degree at most one in $\hat{T}$. Let u be the neighbor of s_0 in $\mathcal{P}^+(s_0)$, i.e., $u = \alpha(s_0)$. By Lemma 4.4 $h'(u) > \ell$. Thus $\pi_{\boldsymbol{M}'}(Q)$ cannot include u or any other vertex of $\hat{\mathfrak{T}}$ reachable through u. Therefore, K' can only be carried by the edge s_0u of $\mathcal{C}'_{\boldsymbol{M}}$. Thus there can be at most one such contour K'. Similarly, there is at most one contour K'' of the latter type (reachable from K by a path in $\check{\boldsymbol{T}} \cap \boldsymbol{M}'_{\geq l}$. This bounds the number of fragments of K to no more than three. $\square$

COROLLARY 4.3. *For a contour represented by (f, ℓ) on $\boldsymbol{M}$, let (f, ℓ') be the pair returned by the algorithm* TRANSLATEQUERY. *Then $K_{\boldsymbol{M}'}(f, \ell')$ has at most three fragments in $F_\rho(\boldsymbol{M}'_{\ell'})$.*

5 I/O-Efficient Implementation

In this section we describe how the stretched terrain $\boldsymbol{M}'$ can be constructed I/O-efficiently and how the data structure in [2] can be adapted to answer contour queries on $\boldsymbol{M}'$ I/O-efficiently. We rely on a few standard I/O-efficient techniques such as list ranking, Euler tour computation of a tree, and time-forward processing. We do not describe these techniques in detail and refer the reader to [18].

5.1 Algorithms for computing the stretching

Here we explain how the individual steps of the algorithm of Section 4 can be implemented using $O(\text{Sort}(N))$ I/Os.

Computing the colored contour tree. The algorithm of Agarwal *et al.* [3] computes the augmented contour tree $\mathcal{C}$ of $\boldsymbol{M}$ in $O(\text{Sort}(N))$ I/Os. If $\mathcal{C}$ is rooted, i.e., if the parent of each vertex with respect to the root v_∞ of $\mathcal{C}$ is determined, finding the colors of the edges of $\mathcal{C}$ can be achieved through a simple scan of the edges of $\mathcal{C}$. This is because an upward edge between a vertex v and a child of v in $\mathcal{C}$ is always red and a downward edge between v and a child of v is always blue. Rooting $\mathcal{C}$ can be achieved in $O(\text{Sort}(N))$ I/Os as described above. Once the color of the edges incident to a vertex are determined the color of the vertex itself follows the local rules described in Section 2.

Computing and rooting $\hat{T}$ and $\check{T}$. Agarwal *et al.* [2] have described an algorithm that computes the cut trees $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$ in $O(\text{Sort}(N))$ I/Os, so we only describe how to compute $\hat{T}$ and $\check{T}$ from $\hat{\boldsymbol{T}}$ and $\check{\boldsymbol{T}}$, respectively. First note that, using the algorithm in [3], we can compute in $O(\text{Sort}(N))$ I/Os the type of each vertex in $\boldsymbol{M}$, and also the counterclockwise first vertex of each connected component of the upper link of every vertex.

Recall that $\hat{T} = (\hat{V}, \hat{E})$ where $\hat{V}$ is the set of maxima in $\boldsymbol{M}$ and $uv \in \hat{E}$ if there is a positive saddle s for which the path $P(s)$ ends in u and v (see Section 4 for details). $\hat{T}$ can be constructed from $\hat{\boldsymbol{T}}$ using the time-forward-processing technique as follows. For the purpose of this algorithm, we assume that the edges of $\hat{\boldsymbol{T}}$ are oriented in decreasing order of the heights of their endpoints. We scan the vertices of $\hat{\boldsymbol{T}}$ in the decreasing order of their heights and compute a labeling φ of each edge of $\hat{\boldsymbol{T}}$ that will be used to construct $\hat{T}$. During the scan we maintain the set of edges whose upper endpoints have been visited in an external priority queue Q; the priority of an edge is the ID of its lower endpoint.

Suppose we reach a vertex v of $\hat{\boldsymbol{T}}$. We first delete all incoming edges at v, i.e., the edges for which v is the lower endpoint, using the delete-min operation repeatedly. If v is a maximum (in which case there are no incoming edges at v), then we label each outgoing edge e of $\hat{\boldsymbol{T}}$ at v with $\varphi(e) = v$. If v is a regular vertex or a negative saddle, then there is exactly one incoming edge e^- at v, and we label each outgoing edge e at v with $\varphi(e) = \varphi(e^-)$. If v is a positive saddle, then there are two incoming edges e_1 and e_2 at v. We report $\varphi(e_1)\varphi(e_2)$ as an edge of $\hat{T}$. For each outgoing edge e at v, we label e with $\varphi(e_1)$ or $\varphi(e_2)$ depending on whichever is $\lambda^+(e)$. Finally, we insert all outgoing edges at v into Q. Omitting further details, we note that this procedure requires $O(\text{Sort}(N))$ I/Os. Next, we pick a maximum $\hat{r}$ that is reachable from v_∞ through an up-going path in the contour tree $\mathcal{C}$. This can be done in $O(\text{Sort}(N))$ I/Os using a similar approach — a label from v_∞ is propagated upward in $\mathcal{C}$ until it is picked up by some maximum which is appointed $\hat{r}$. Once $\hat{r}$ is chosen, the edges of $\hat{T}$ can be oriented toward $\hat{r}$ as described above for the contour tree.

Similarly $\check{T}$ can be computed and rooted at v_∞ in $O(\text{Sort}(N))$ I/Os.

Computing $\alpha(s)$ and $\beta(s)$ for a each s. We perform an Euler tour on $\mathcal{C}$, initiated at v_∞, and record for each vertex v the steps of the tour in which v is visited. The number of steps recorded for a vertex v is its degree in $\mathcal{C}$. Thus extrema get a single number while saddles record three numbers. Let s be a saddle in $\mathcal{C}$ and let u and v be its two children. Assume that the Euler tour reaches u before v for the first time. Since the tour is started at v_∞, if the three numbers recorded at a saddles s, i.e., the steps of the tours in which s is visited, are $a_1 < a_2 < a_3$, then all the step numbers associated to all vertices in $\mathcal{C}^u$ are between a_1 and a_2 and those associated to vertices in $\mathcal{C}^v$ are between a_2 and a_3. All other vertices of $\mathcal{C}$ are visited in steps before a_1 or in steps after a_3. Thus if s is, say a positive saddle, one can determine the neighbors $\alpha(s)$ and $\beta(s)$ on the paths from s to head(s) and tail(s) by comparing a_1, a_2, and a_3 to the steps in which each of u and v are visited. This can therefore be done in $O(\text{Sort}(N))$ I/Os.

Computing $n(s)$ for each saddle. One can compute in $O(\text{Sort}(N))$ I/Os the size of all subtrees rooted at all vertices of a rooted tree of size N. This will be enough to determine $n(s)$ for all saddles as $n(s)$ is by definition the number of leaves in $\mathcal{C}^{\beta(s)}$.

Computing $\boldsymbol{M}'$. With $n(s)$ and the color of the saddles known, $\delta(s)$ is determined for each saddle. To compute $\boldsymbol{M}'$ we perform another Euler tour initiated at v_∞. As explained in Section 3, the stretching associated with a saddle s is applied to all vertices in the subtree of $\mathcal{C}$ rooted at the child endpoint of the edge $s\alpha(s)$. In the Euler tour, we maintain the total change in height applicable to the vertices being visited in a variable L, initially set to zero. For any saddle s, the edge $s\alpha(s)$ is traversed twice in the Euler tour. The first traversal moves from the parent endpoint of the edge to the child endpoint at which time we add $\delta(s)$ to L. When $s\alpha(s)$ is visited for the second time, it will be in the direction from its child endpoint to its parent endpoint in $\mathcal{C}$. We thus subtract $\delta(s)$ from L. Thus the value of L when visiting a vertex v is precisely $\Delta(v)$. To compute $\boldsymbol{M}'$, all one needs to do is to add L to $h(v)$ when it is visited for the first time.

5.2 Answering contour queries

We now briefly review the level-set query algorithm of Agarwal *et al.* [2] and explain how one can answer contour queries using their structure. We refer to the original paper for further details.

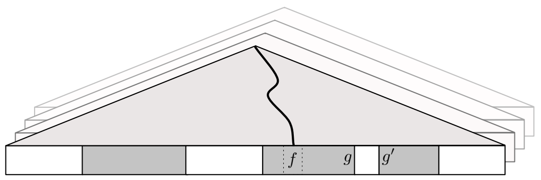


Figure 17: Locating a triangle f in the appropriate version of the B-tree $\mathcal{B}$ and scanning the leafs for the extraction of the contour K containing f. Fragments of K are shown in grey. At the end of a fragment, one locates the subsequent fragment by searching the tree for the rank of the neighbor g' of the last triangle g of the previous fragment.

Level-set queries. Let $v_1, \ldots, v_N$ be the vertices of $\boldsymbol{M}$ in the order of height, i.e., $h(v_1) < \cdots < h(v_N)$. For simplicity we assume that all queries are performed at heights $\ell \notin h(V)$. A level-set query at height ℓ returns $\bar{F}[K]$ for every contour K in $\boldsymbol{M}_\ell$. The resulting collection of lists is exactly the same for two level-set queries at levels ℓ_1, ℓ_2 with $h(v_i) < \ell_1, \ell_2 < h(v_{i+1})$. It is therefore sufficient for our algorithm to generate $M[K]$ for each contour K of $M_{h(v_i)+\varepsilon}$ for $i = 1, \ldots, N$; here ε is a sufficiently small constant as in the definition of up- and down-contours.

The overall data structure is a persistent B-tree $\mathbb{B}$ that holds N *versions*, each corresponding to one vertex of $\boldsymbol{M}$. The version $\mathbb{B}_i$ associated with v_i is a B-tree that stores the sequence $F_\rho(\boldsymbol{M}_{h(v_i)+\varepsilon})$. The search key for a triangle f in any version of $\mathbb{B}$ is its rank $\rho(f)$ as determined by the Agarwal *et al.* [2] level-ordering ρ. To construct $\mathbb{B}$, we scan the list $v_1, \ldots, v_N$. On arriving at v_i, we first delete from $\mathbb{B}$ all triangles for which v_i is the highest vertex and then insert all those for which v_i is the lowest vertex. The resulting version of $\mathbb{B}$ is time-stamped $h(v_i)$. $\mathbb{B}$ can be constructed in $O(\text{Sort}(N))$ I/Os and uses $O(N)$ space.

For any ℓ, let $\mathbb{B}(\ell)$ be the version $\mathbb{B}_i$ of $\mathbb{B}$ corresponding to the highest vertex v_i with $h(v_i) < \ell$. $\mathbb{B}(\ell)$ can be computed in $O(\log_B N)$ I/Os and its entire content, i.e., $F_\rho(\boldsymbol{M}_\ell)$, can be reported, ordered by ρ, in $O(T/B)$ I/Os, where $T = |F(\boldsymbol{M}_\ell)|$. Property C2 of level-ordering allows us to separate the various contours in this sequence using an I/O-efficient stack in $O(T/B)$ extra I/Os [2].

Contour queries. $\mathbb{B}$ can be adapted to answer contour queries, albeit with possible compromise on the I/O complexity by retrieving various fragments of a contour from the relevant version: Given a query triangle f an a level ℓ, to extract the contour K in $\boldsymbol{M}_\ell$ that intersects f, as before one first locates $\mathbb{B}(\ell)$ in $O(\log_B N)$ I/Os. Assuming $\mathbb{B}(\ell)$ stores T_ℓ elements, the leaf of $\mathbb{B}(\ell)$ that stores f is found in $O(\log_B T_\ell/B)$ I/Os by searching in $\mathbb{B}(\ell)$ using $\rho(f)$ as the search key. The leaves of $\mathbb{B}(\ell)$ are then scanned sequentially from f to the left and right to find the remaining triangles in K. Since K may be fragmented in $\mathbb{B}(\ell)$, one may reach the end of the fragment that includes f at a triangles g without having found all of $\bar{F}[K]$ (See Figure 17). If one stores at each triangle of M the rank of its three neighboring triangles, the fragment of $\bar{F}[K]$ that starts with the neighbor g' of g can be located by searching $\mathbb{B}(\ell)$ using $\rho(g')$ as the search key; this step takes in $O\Big(\log_{M/B} T_\ell/B\Big)$ I/Os. Continuing this way, a contour K of length T broken into J fragments is reported in $O\Big(\log_B N + J \log_{M/B} T_\ell/B + T/B\Big)$ I/Os.

If contour queries are issued on the stretched terrain obtained using the algorithm of Section 4, the number J of fragments is constant (Corollary 4.3). Therefore, we obtain the main result of the paper:

THEOREM 5.1. *A terrain $\boldsymbol{M}$ of size N can be preprocessed, in $O(\mathrm{Sort}(N))$ I/Os, into a data structure of size $O(N)$ so that a contour query (f, ℓ) on $\boldsymbol{M}$ can be answered in $O(\log_B N + T/B)$ I/Os, where T is the size of the query output.*

References

[1] P. K. Agarwal, L. Arge, T. M. Murali, K. Varadarajan, and J. S. Vitter. I/O-Efficient Algorithms for Contour Line Extraction and Planar Graph Blocking. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 117–126, 1998.

[2] Pankaj K. Agarwal, Lars Arge, Thomas Mølhave, and Bardia Sadri. I/O-Efficient Algorithms for Computing Contours on a Terrain. In *SCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 129–138, New York, NY, USA, 2008. ACM.

[3] Pankaj K. Agarwal, Lars Arge, and Ke Yi. I/O-Efficient Batched Union-Find and its Applications to Terrain Analysis. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 167–176, New York, NY, USA, 2006. ACM.

[4] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.

[5] L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.

[6] L. Arge, A. Danner, and S-H. Teh. I/O-efficient point location using persistent B-trees. In *Proc. Workshop on Algorithm Engineering and Experimentation*, 2003.

[7] Surender Baswana and Sandeep Sen. Planar graph blocking for external searching. *Algorithmica*, 34(3):298–308, 2002.

[8] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Comput. Geom.*, 24(2):75–94, 2003.

[9] Y.-J. Chiang and C. T. Silva. I/O-Optimal Isosurface Extraction. In *Proc. IEEE Visualization*, pages 293–300, 1997.

[10] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 139–149, 1995.

[11] Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in Reeb graphs of 2-manifolds. *Discrete & Computational Geometry*, 32(2):231–244, 2004.

[12] A. Danner, T. Mølhave, K. Yi, P. K. Agarwal, L. Arge, and H. Mitásová. Terrastream: From elevation data to watershed hierarchies. In *Proc. ACM Symposium on Advances in Geographic Information Systems*, page 28, 2007.

[13] Herbert Edelsbrunner, John Harer, and Afra Zomorodian. Hierarchical morse - smale complexes for piecewise linear 2-manifolds. *Discrete & Computational Geometry*, 30(1):87–107, 2003.

[14] C. Hutton. An account of the calculations made from the survey and measures taken at schehallien, in order to ascertain the mean density of the earth. *Philosophical Transactions of the Royal Society of London*, 68:689–845, 1779.

[15] M. H. Nodine, M. T. Goodrich, and J. S. Vitter. Blocking for external graph searching. *Algorithmica*, 16(2):181–214, 1996.

[16] P. J. Varman and R. M. Verma. An efficient multiversion access structure. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):391–409, 1997.

[17] J. S. Vitter. External memory algorithms and data structures. In *External memory algorithms*, pages 1–38. American Mathematical Society, Boston, MA, USA, 1999.

[18] Norbert Zeh. I/O-efficient graph algorithms. In *EEF Summer School on Massive Data Sets*, to appear.