

I/O-Efficient Flow Modeling on Fat Terrains

Mark de Berg¹, Otfried Cheong², Herman Haverkort¹, Jung Gun Lim², and
Laura Toma³

¹ TU Eindhoven, the Netherlands, mdberg@win.tue.nl, cs.herman@haverkort.net

² KAIST, Korea, otfried@kaist.ac.kr, araste@gmail.com

³ Bowdoin College, USA, ltoma@bowdoin.edu

Abstract. We study the flow of water on *fat terrains*, that is, triangulated terrains where the minimum angle of any triangle is bounded from below by a positive constant. We give improved bounds for the worst-case complexity of river networks on fat terrains, and show how to compute the river network and other flow-related structures I/O-efficiently.

1 Introduction

In this paper we study one of the most important problems on terrains, analyzing the flow of water. The basic questions in flow analysis are to identify the river network and, for any given point q , its watershed (the part of the terrain from which water flows to q). Acquiring real flow data for a terrain is tedious, time-consuming and often impossible. Fortunately high-resolution elevation data is now widely available. As a result, flow modeling and analysis based on elevation data is a popular topic for researchers in GIS (geographic information systems).

One common representation of terrain data in a GIS is the TIN (*triangulated irregular network*). A TIN—in computational-geometry terms: a triangulated polyhedral terrain—is obtained by triangulating a collection of irregularly spaced sample points and then giving each triangulation vertex the elevation of the corresponding sample point. When working with very large terrains, the data is too large to fit into the computer’s main memory. Most of the data must therefore reside on disk during the computation, making I/O (moving data between main memory and disk) the bottleneck of the computation. This leads us to the topic of our paper: the study of river networks and watersheds on TINs, and the design of I/O-efficient algorithms for computing these structures.

We analyze our algorithms with the model introduced by Aggarwal and Vitter [2], which has become the standard model for I/O-efficient algorithms. In this model, a computer has an internal memory of size M and an arbitrarily large external memory (disk) where data is stored in blocks of size B . The *I/O-complexity* of an algorithm in this model is measured in terms of the number of I/O’s—reading or writing a block from or to external memory—it performs. In this model, scanning (reading a set of n consecutive items from disk) takes $Scan(n) = \Theta(n/B)$ I/O’s, and sorting $Sort(n) = \Theta((n/B) \log_{M/B}(n/B))$ I/O’s.

The previous work on modeling flow on TINs falls into two classes. Most GIS papers [9, 15–18] adopt a *discrete* approach and route flow from a triangle

to one of its three neighbour triangles using the direction of steepest descent. This approach is appealing because of its simplicity; it is problematic, however, because it discretizes flow and tends to lead to inconsistencies when the triangles in the TIN differ a lot in size [10, 19]. The approach taken in the computational-geometry literature considers the TIN as a *continuous* surface on which water always flows in the direction of steepest descent. De Berg et al. [6], McAllister [11, 12], McAllister and Snoeyink [13] and Yu and Snoeyink [19] study the structure and the complexity of the river network and other structures on TINs under this model. In particular, De Berg et al. prove that the complexity of the river network—see Section 2 for a formal definition—on a TIN of n vertices is $\Theta(n^3)$ in the worst case: there can be $\Theta(n)$ separate rivers, each with complexity $\Theta(n^2)$. None of the papers mentioned above provide I/O-efficient algorithms.

I/O-efficient flow modeling was first studied—on grids, the other type of data representation in GIS—by Arge et al. [4]. Their system, **Terraflow**, has become the state of the art in flow modeling on massive grids. **Terraflow** uses a discrete approach which can be easily extended to TINs. However, discrete flow is only an approximation of real flow. Thus, the challenge is to develop I/O-efficient algorithms to model continuous flow on TINs, which is ultimately more accurate.

The main step in the computation of flow, and at the same time the bottleneck in the I/O-model, is tracing paths of steepest descent across the triangles that they intersect—in particular, any river is such a path of steepest descent. While in internal memory a path of size k can be traced in $O(k)$ time, the best known I/O-bound is $O(k/\log B)$ I/O's on planar graphs [1]. This results in a straightforward bound of $O((r+n)/\log B)$ I/O's for the computation of a river network of size r , but this would be prohibitively expensive.

Moreover, De Berg et al. [6] showed that r is $\Theta(n^3)$ in the worst case. However, the worst case is a construction that is unlikely to occur in real life. In computational geometry such discrepancies between worst case and practice have led to the study of input models that resemble realistic inputs better. Moet et al. [14] studied visibility and distance problems on *realistic terrains*. In this paper we consider flow modeling on *fat terrains*, that is, terrains where the minimum angle of any triangle is bounded from below by a positive constant⁴. Our notion of a fat terrain is less restrictive than the notion of realistic terrains from Moet et al.

Our results. In this paper we give improved bounds for the complexity of the river network on a fat terrain and show how to compute a number of flow-related structures I/O-efficiently. The main ingredient in our solution is to represent the terrain by a directed graph, which we call the *descent graph*, $\mathcal{G}_{\text{desc}}$. The nodes of $\mathcal{G}_{\text{desc}}$ represent the edges of the triangulation, and we define the arcs of $\mathcal{G}_{\text{desc}}$ such that following a path of steepest descent on the terrain corresponds to following a path in $\mathcal{G}_{\text{desc}}$. Unfortunately, in its basic form $\mathcal{G}_{\text{desc}}$ can have cycles, and a path of steepest descent can visit the same edge more than once. In fact this is exactly the reason why the complexity of a river in an arbitrary terrain can

⁴ Here the angle of a triangle is measured in space. Our results also hold if the angles are measured in the projection on the xy -plane, or if all triangles are non-obtuse.

be $\Theta(n^2)$: it can visit a linear number of edges each a linear number of times [6]. In the I/O-model, a descent graph with cycles does not only signify a potentially problematic output size, but it also constitutes an algorithmic problem, because it is not known how to store such a graph on disk such that any path of length k can be traced using $O(k/B)$ I/O's [1].

The cornerstone of this paper is an idea that solves both problems at the same time: we subdivide each edge of the triangulation into a number of segments, in such a way that the descent graph, defined on these segments instead of the original edges, is *acyclic*. Moreover we show that for fat terrains a constant number of segments per edge suffices. This implies that any path of steepest descent can visit each segment at most once, and hence its worst-case complexity is $\Theta(n)$. This in turn implies an $O(n^2)$ bound on the complexity s of the *strip map*: the subdivision \mathcal{S} of the terrain induced by the paths of steepest ascent and descent from all vertices. It also follows that the complexity r of the river network on a fat terrain is $O(n^2)$, which is a linear factor smaller than the $O(n^3)$ bound for river networks on general terrains. In the full version of this paper we show that our bounds are tight in the worst case.

The acyclicity of the descent graph allows us to apply time-forward processing [3, 7] and traverse paths in a batched I/O-efficient manner. By applying and refining ideas from McAllister [11] and Yu et al. [19], we obtain the following algorithms and data structures, all computable in $O(\text{Sort}(s))$ I/O's: (1) an algorithm to compute the river network, with a piecewise quadratic function of $O(r + n)$ pieces whose value is the area of the watershed for each point of the network; (2) a data structure that reports the boundary of the watershed of any query point q in $O(l + w/B)$ I/O's, where l is the number of I/O's needed to locate q in \mathcal{S} and w is the complexity of the reported watershed; (3) a structure that reports the flow path from any point q (the course of water flowing from q) in $O(l + c/B)$ I/O's, where c is the complexity of the path;

One of the open questions posed by De Berg et al. [6] was if one could prove an $O(n^2)$ bound on the complexity of river networks in Delaunay triangulations. In the full version of this paper we answer this question negatively and show that we can construct Delaunay triangulations with river networks of size $\Theta(n^3)$.

2 Preliminaries

Let \mathcal{T} be a TIN defined on n vertices. To model flow we assume that water always runs downhill in the direction of steepest descent. Furthermore, we assume that the direction of steepest descent is unique for any point in the terrain (so there are no horizontal triangles); no water flows off the terrain, and no edge is parallel to the direction of steepest descent on an adjacent triangle. We discuss how to do without the last three assumptions in the long version of this paper.

Following Yu et al. [19], we distinguish three types of edges in \mathcal{T} : *transfluent edges* are edges that receive water from one adjacent triangle which continues its way down the other triangle; *channels* are edges that receive water from both adjacent triangles; *ridges* are edges that do not receive water from any triangle.

The direction of steepest descent from a vertex may be along an incident edge, or on an incident triangle orthogonal to its contour lines. To capture this, we define the *slope profile* of a point p on the terrain as the function $s_p : S^1 \rightarrow \mathbb{R}$ such that $s_p(\theta)$ is the slope of the path that leaves p in the direction θ . The interesting directions for a vertex v of \mathcal{T} correspond to the local maxima and minima in the slope profile of v . Note that these directions include the directions of channels and ridges incident to v , and also the directions of steepest descent and ascent. A vertex v is a *pit* if its slope profile is entirely positive.

We define up-paths and down-paths as paths of locally steepest ascent or descent as follows. An *up-path* from p is a path that starts at p , goes into a direction θ that is a positive local maximum in the slope profile of p leading onto the interior of an incident triangle, and then follows the steepest ascent until a vertex or a ridge of \mathcal{T} is reached. (The requirement that the path leads onto the interior of the triangle incident to p prevents an up-path from leaving p along a ridge.) Similarly, a *down-path* from p is a path that starts at p , goes into a direction θ that is a negative local minimum in the slope profile of p leading onto the interior of an incident triangle, and then follows the steepest descent until a vertex or a channel of \mathcal{T} is reached. An up-path from p to q is a down-path from q to p . Therefore, we may sometimes refer to a down-path as an up-path or the other way around, depending on our point of view.

The *strip map* \mathcal{S} of \mathcal{T} is the subdivision of \mathcal{T} induced by the channels, ridges, up-paths and down-paths from all vertices of \mathcal{T} . We call the $O(n)$ faces of this map *strips*. Each strip is bounded by a portion of a ridge, a portion of a channel (the foot), and two possibly empty chains of up-paths. Note that from every point at the foot of a strip, the up-path through the strip has the same combinatorial structure: it crosses the same triangles and leads to the same ridge.

The *watershed* $W(q)$ of a point q on \mathcal{T} is the set of all points on \mathcal{T} from which the water flows to q . The *river network* of \mathcal{T} is the set of all points on \mathcal{T} with watersheds of non-zero area, or in other words, the set of points whose watersheds are two-dimensional regions. De Berg et al. [6] argue that the river network of \mathcal{T} is the union of the channels of \mathcal{T} and the paths of steepest descent that start from the lower endpoints of channels.

3 Modeling flow with a descent graph

In this section we describe how to model flow on a triangulated terrain \mathcal{T} using a *descent graph* $\mathcal{G}_{\text{desc}}$. Our goals are to define $\mathcal{G}_{\text{desc}}$ such that it is acyclic and such that any path of steepest descent in \mathcal{T} corresponds to a path in $\mathcal{G}_{\text{desc}}$. For α -fat terrains, that is, terrains \mathcal{T} where the minimum angle of each triangle (measured in the plane supporting the triangle) is at least a constant $\alpha > 0$, we show how to construct $\mathcal{G}_{\text{desc}}$ with only $O(n/\alpha^2)$ nodes. This leads to the following bounds on the size of down-paths, up-paths, river networks and strip maps:

Theorem 1. *Any down-path or up-path in an α -fat terrain has $O(n/\alpha^2)$ vertices, and the total complexity of its river network or the strip map is $O(n^2/\alpha^2)$.*

To define $\mathcal{G}_{\text{desc}}$, assume that the edges of \mathcal{T} have been subdivided into *segments*, as will be described below. Then the descent graph $\mathcal{G}_{\text{desc}}$ of \mathcal{T} contains a node for each vertex and each segment in \mathcal{T} ; let $\text{seg}(v)$ denote the vertex/segment corresponding to node v . To ensure that the sets $\text{seg}(v)$ are disjoint, we consider a segment to include its upper endpoint (unless this is a vertex of \mathcal{T}), and to exclude its lower endpoint. $\mathcal{G}_{\text{desc}}$ contains a directed arc from node a to node b if and only if one of the following applies: (1) $\text{seg}(a)$ is the upper endpoint of $\text{seg}(b)$, and $\text{seg}(b)$ is a ridge or channel; (2) $\text{seg}(b)$ is the lower endpoint of $\text{seg}(a)$, and $\text{seg}(a)$ is a ridge or channel; (3) $\text{seg}(a)$ and $\text{seg}(b)$ are on the boundary of the same triangle Δ , and there is a path of steepest descent across the interior of Δ from $\text{seg}(a)$ to $\text{seg}(b)$. It can be seen that any path of steepest descent in \mathcal{T} corresponds to a path in $\mathcal{G}_{\text{desc}}$.

To ensure that $\mathcal{G}_{\text{desc}}$ is acyclic we impose the following requirements on the subdivision of the edges of \mathcal{T} . Channels/ridges of \mathcal{T} are not subdivided: each channel/ridge is one segment. For a segment s on a transfluent edge e and a triangle Δ incident to e , we have: (1) If s is not incident to a vertex, then the (open) smallest enclosing sphere of s (that is, the sphere with s as a diameter) does not intersect any other edge of Δ ; (2) If s is incident to a vertex p of Δ , then the (open) sphere centered at p with s as a radius does not intersect any segment s' on an edge of Δ , where s' is not incident to p and s' is not separated from s by a line of steepest descent through p on Δ . We call a subdivision of the edges of \mathcal{T} that meets these requirements *compliant*.

We denote the upper endpoint, midpoint and lower endpoint of $\text{seg}(v)$ by $up(v)$, $md(v)$ and $lw(v)$. If $\text{seg}(v)$ is a vertex we have $up(v) = md(v) = lw(v) = \text{seg}(v)$. We define the anchor $an(v)$ and relative position $rp(v)$ of v as follows:

- if $\text{seg}(v)$ is a channel, then $an(v) = lw(v)$ and $rp(v) = 1$ (above the anchor);
- if $\text{seg}(v)$ is a vertex or a segment of a transfluent edge, then $an(v) = md(v)$ and $rp(v) = 0$ (on the anchor);
- if $\text{seg}(v)$ is a ridge, then $an(v) = up(v)$ and $rp(v) = -1$ (below the anchor).

We now prove that any descent graph $\mathcal{G}_{\text{desc}}$ based on a compliant subdivision is acyclic. To this end, we define a partial order \succ as follows: For two nodes a and b in $\mathcal{G}_{\text{desc}}$, we define $a \succ b$ if and only if $z(an(a)) > z(an(b))$, or $z(an(a)) = z(an(b))$ and $rp(a) > rp(b)$, where $z(p)$ is the elevation of p .

Lemma 1. *Given $\mathcal{G}_{\text{desc}}$ of a compliant subdivision, if $\text{seg}(a)$ and $\text{seg}(b)$ are on the boundary of the same triangle Δ , $\text{seg}(b)$ is not a channel, and there is a path of steepest descent across the interior of Δ from $\text{seg}(a)$ to $\text{seg}(b)$, then $z(up(a)) \geq z(up(b))$, and equality can only hold if $\text{seg}(a)$ is incident to $up(b)$.*

Proof. Let $p \in \text{seg}(a)$ and $q \in \text{seg}(b)$ such that the segment pq lies in Δ and follows the direction of steepest descent. If $\text{seg}(b)$ is a vertex of \mathcal{T} , then $z(up(a)) \geq z(p) > z(q) = z(up(b))$. Since $\text{seg}(b)$ is not a ridge or a channel, the only remaining case is that $\text{seg}(b)$ is a segment on a transfluent edge.

Consider now the plane Γ supporting Δ . Let S be the open half-plane on Γ containing p , bounded by the line through $\text{seg}(b)$. Let ℓ be the intersection of Γ

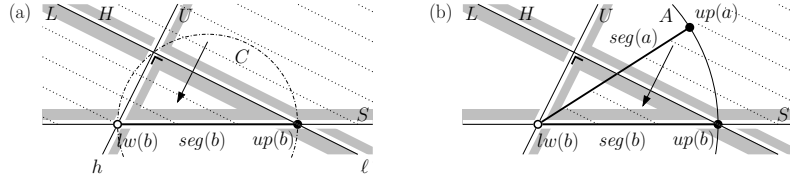


Fig. 1. Proof of Lemma 1.

with the horizontal plane through $up(b)$, and let h be the line on Γ orthogonal to ℓ through $lw(b)$. Note that ℓ is a contour line of Γ , the line h follows the direction of steepest descent on Γ , and pq is parallel to h . Let L and H be the closed lower and the open upper half-planes of Γ bounded by ℓ , and let U be the open half-plane of Γ bounded by h containing $seg(b)$. Let C be the minimum circumscribed circle C of $seg(b)$ on Γ . See Fig. 1 (a) for an illustration. Since $q \in seg(b)$, we must have $p \in U \cap S$. We distinguish three cases:

First, if $seg(b)$ is not incident to a vertex of \mathcal{T} , then by compliance condition (1) C does not intersect $seg(a)$. By Thales' theorem, the rectangular triangle $U \cap S \cap L$ lies in C , and so $p \notin L$. This implies $z(up(a)) \geq z(p) > z(up(b))$.

Second, if $lw(b)$ is a vertex of \mathcal{T} , then let A be the open disk centered at $lw(b)$ through $up(b)$. If $seg(a)$ is not incident to $lw(b)$, then by compliance condition (2) it does not intersect A (since $p \in U \cap seg(a)$, $seg(a)$ is not separated from $seg(b)$ by h). Since $C \subset A$, we again have $z(up(a)) \geq z(p) > z(up(b))$. If $seg(a)$ is incident to $lw(b)$ (Fig. 1 (b)), then by compliance condition (2) $up(a)$ must lie in $U \setminus A$, and hence in H , implying $z(up(a)) > z(up(b))$.

Finally, if $up(b)$ is a vertex of \mathcal{T} , then let A' be the open disk centered at $up(b)$ through $lw(b)$. If $seg(a)$ is incident to $up(b)$, then clearly $z(up(a)) \geq z(up(b))$, otherwise by condition (2) $seg(a)$ lies outside A' , and p lies again in H . \square

Corollary 1. *If $seg(a)$ and $seg(b)$ are on the boundary of the same triangle Δ , $seg(a)$ is not a ridge, and there is a path of steepest descent across the interior of Δ from $seg(a)$ to $seg(b)$, then $z(lw(a)) \geq z(lw(b))$, and equality can only hold if $seg(b)$ is incident to $lw(a)$.*

Lemma 2. *Given \mathcal{G}_{desc} of a compliant subdivision, if \mathcal{G}_{desc} contains an arc from a to b , then $a \succ b$.*

Proof. We can verify that if $seg(a)$ is the upper endpoint of $seg(b)$, or if $seg(b)$ is the lower endpoint of $seg(a)$, then $a \succ b$. It remains to discuss the case where there is a path of steepest descent from some point $p \in seg(a)$ to some point $q \in seg(b)$ across the interior of their common triangle Δ . We note that $seg(a)$ cannot be a channel, and $seg(b)$ cannot be a ridge. We have $z(up(a)) \geq z(p) > z(q) \geq z(lw(b))$. We now distinguish four cases, proving $z(an(a)) > z(an(b))$ and thus $a \succ b$ in each: First, if $seg(a)$ is a ridge and $seg(b)$ is a channel, then $z(an(a)) = z(up(a)) > z(lw(b)) = z(an(b))$. Second, if $seg(a)$ is a ridge and $seg(b)$ is a vertex or a segment on a transfluent edge, then $z(an(a)) = z(up(a)) \geq z(up(b))$ (by

Lemma 1). Since $z(up(a)) > z(lw(b))$, then $z(an(a)) > z(md(b)) = z(an(b))$. Third, if $seg(a)$ is a vertex or a segment and $seg(b)$ is a channel, it is handled symmetrically. Finally, if both $seg(a)$ and $seg(b)$ are segments of transfluent edges, then we have both $z(up(a)) \geq z(up(b))$ and $z(lw(a)) \geq z(lw(b))$. We cannot have equality in *both* inequalities, since if $seg(a)$ and $seg(b)$ would share both $up(b)$ and $lw(a)$, that would imply $up(b) = lw(a)$ and thus $z(up(a)) = z(lw(b))$, contradicting $z(p) > z(q)$. Hence equality holds in at most one of the above comparisons, so $z(an(a)) = z(md(a)) > z(md(b)) = z(an(b))$. \square

It follows that \mathcal{G}_{desc} of a compliant subdivision is a directed acyclic graph. We now sketch how to construct a compliant subdivision of size $O(n)$ for an α -fat terrain. Consider a vertex p of \mathcal{T} , and let r be the distance from p to the nearest edge not incident on p . We create a segment of length $r/2$ on each transfluent edge incident to p . Since the ratio of the longest and shortest edge in a fat triangle is bounded by a constant, and assuming the degree of p is constant, each of these segments covers a constant fraction of its edge. We can subdivide the remainder of the transfluent edges into a constant number of segments satisfying condition (1). In the long version of this paper we prove that the assumption on the vertex degree is not necessary and we analyse the dependency on α .

Theorem 2. *Any α -fat terrain has a compliant subdivision of size $O(n/\alpha^2)$.*

4 Computing the rivers and the watershed area map

This section shows how the acyclic descent graph can be used to construct the river network and the strip map of the terrain I/O-efficiently. We also show how to compute, for every point on the river network, the area of its watershed.

First we need the following. In Section 3 we showed that an α -fat terrain can be modeled by a descent graph \mathcal{G}_{desc} of $O(n/\alpha^2)$ nodes and arcs. We have:

Lemma 3. *\mathcal{G}_{desc} of an α -fat terrain can be computed in $O(\text{Sort}(n/\alpha^2))$ I/O's.*

Recall that the river network consists of the channels in the terrain and the paths of steepest descent that start at the lower endpoints of channels. The channels can be extracted from the terrain in a straightforward way. The challenge is tracing paths of steepest descent I/O-efficiently: we need to trace each such path from its upper endpoint across the triangles that it intersects, one segment at a time. We would like that a path that crosses k triangles can be computed in $O(\text{Sort}(k))$ I/O's. A general solution for this problem is not known in the I/O-model. This is where \mathcal{G}_{desc} comes to rescue. The key idea is that every segment of a path of steepest descent is captured by an arc in the descent graph. Thus, instead of tracing such paths on the original terrain, we trace them in \mathcal{G}_{desc} . Doing this path by path would not be any faster, since even for planar directed acyclic graphs it is not known how to preprocess them for fast path traversals. Instead, we trace *all* paths of steepest descent in parallel while traversing \mathcal{G}_{desc} in topological order (highest nodes first). In this way we compute the segments of the paths of steepest descent I/O-efficiently in a batched way.

More precisely, our approach is as follows. We put the arcs of $\mathcal{G}_{\text{desc}}$ on a stack A sorted by \succ -order of their nodes of origin, with the nodes that appear first in \succ -order on top. Furthermore, we initialize an I/O-efficient priority queue Q that stores pairs of the type (v, p) , where v is a node in $\mathcal{G}_{\text{desc}}$, and p is a point of $\text{seg}(v)$. The queue is also organized by \succ -order. Initially we fill Q with all pairs (v, p) where p is a vertex of a channel, and v is the corresponding node in $\mathcal{G}_{\text{desc}}$.

We now repeat the following until Q is empty. From Q we extract the pair (v, p_1) with highest priority, and all further pairs $(v, p_2), (v, p_3), \dots$ with the same priority. From A we pop arcs $\overrightarrow{uw_1}$ until $u = v$ or until $v \succ u$. In the latter case, no arcs that lead out of v are found, and all paths of steepest descent that reach v end there—we proceed to extracting the next pair (v, p_1) from Q . Otherwise at least one arc leads out of v . We also pop any remaining arcs $\overrightarrow{vw_2}, \overrightarrow{vw_3}, \dots$ that originate from v from A . For each pair (v, p_i) , we now select the arc $\overrightarrow{vw_j}$ that captures the path of steepest descent from p_i to a point q_i on $\text{seg}(w_j)$. If $\text{seg}(w_j)$ is not incident to p_i , we output the line segment from p_i to q_i as a river segment. If $\text{seg}(w_j)$ is not a channel or its bottom endpoint, we insert (w_j, q_i) in Q so that the path of steepest descent is traced further. After handling all pairs $(v, p_1), (v, p_2), \dots$, we proceed by extracting the next pair from Q . We get:

Theorem 3. *The river network of an α -fat terrain can be computed in $O(\text{Sort}(r + n/\alpha^2))$ I/O's, where r is the size of the river network.*

The above approach can be extended to compute the subdivision of \mathcal{T} induced by *all* channels, ridges, up-paths and down-paths starting from vertices of \mathcal{T} , that is, the strip map, in $O(\text{Sort}(s + n/\alpha^2))$ I/O's, where s is the size of the map.

Let q be a point on a channel segment. The watershed $W(q)$ of q contains parts of the at most two strips that have q at their feet [19]; the area of each part is given by a quadratic function of the position of q that is determined by scanning the list of triangles that intersect the strip. All other strips lie either inside or outside $W(q)$; more precisely, a strip lies in $W(q)$ if and only if its lowest point lies at q or on the river network upstream of q .

To compute the piecewise quadratic function whose value is the watershed area for every point on the river network, we process the river network from the leaves to the root and collect, for every edge of the river network, the area functions for the strips to the left and to the right and the total area of the strips that drain into the subtree below (that is, upstream of) that edge. This can be done with standard techniques (e.g. a post-order traversal of the river network).

Theorem 4. *Given an α -fat terrain \mathcal{T} we can compute in $O(\text{Sort}(s + n/\alpha^2))$ I/O's a piecewise quadratic function of $O(r + n)$ pieces whose value is the watershed area of every point in \mathcal{T} , where r is the size of the river network and s is the size of the strip map of \mathcal{T} .*

5 A data structure for flow path and watershed queries

In this section we describe an I/O-efficient data structure for fast flow path and watershed boundary queries: given a point q on the terrain, we want to report

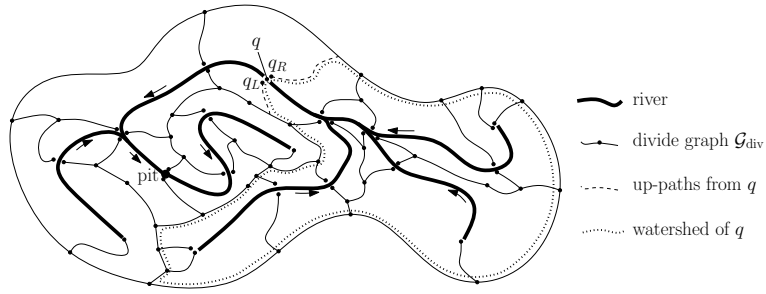


Fig. 2. Finding the watershed of q .

the flow path starting at q and the boundary of the watershed $W(q)$ of q . As explained in the previous section, the watershed of a point q on the river network can be found by traversing the subtree of the river network upstream of q , and collecting the strips that drain into that subtree and parts of the strips that have q at their feet. However, this would yield all strip boundaries that lie in the interior of the watershed, whose total size may be much bigger than the boundary of the complete watershed given as a simple polygon. To address this problem McAllister [11] suggested to build a divide graph, \mathcal{G}_{div} , that consists of sections of ridges and certain up-paths in \mathcal{T} . The watershed boundary of a query point q can then be found by adding the up-paths from q to \mathcal{G}_{div} . This will divide the face of \mathcal{G}_{div} that contains q into two subfaces: the subface upstream of q is the watershed of q (see Fig. 2). Below we describe a refined version of \mathcal{G}_{div} and sketch how to store it face by face in a way that makes it possible to report watershed boundaries I/O-efficiently.

For every vertex v of \mathcal{T} , and for every endpoint v of an up-path or a down-path from a vertex of \mathcal{T} , consider an infinitesimally small circle centered at the projection of v on the horizontal plane. Cut this circle where it is crossed by the projections of up-paths that start from v , channels that lead down to v , or the path of steepest descent from v . Note that we do not cut the circle at every down-path from v , but only at the down-path that descends steepest from v . Every piece of the circle that results constitutes a node of \mathcal{G}_{div} , which is not (directly) connected to the other pieces of the circle. The arcs of \mathcal{G}_{div} correspond to (i) the ridges of \mathcal{T} and (ii) two copies of each up- or down-path starting from a vertex v of \mathcal{T} . The two copies are assumed to lie at an infinitesimally small distance from the real course of the path, such that one copy runs to the left of the path, and the other to the right, leaving a narrow corridor between them.

The arcs of \mathcal{G}_{div} are connected to the nodes of \mathcal{G}_{div} as follows. Every arc that corresponds to a path π that has v as an endpoint, is connected to the node whose piece of the circle around v is crossed by the projection of π on the plane. When an up- or down-path passes between two pieces of the circle on its way to or from v , the copy of the path that is offset to its left is connected to the piece of the circle to its left, and the copy that is offset to its right is connected to the piece of the circle to its right (Fig. 3). Note that arcs are incident to the same

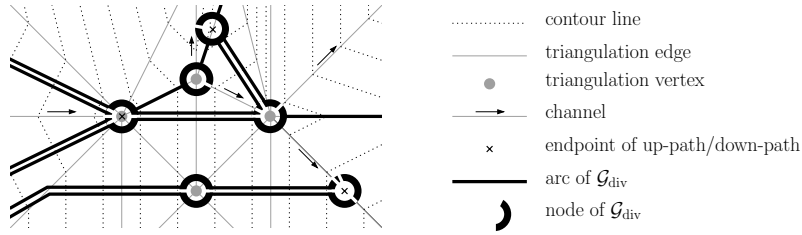


Fig. 3. An example of nodes and arcs in a divide graph.

node—that is, the same piece of a circle—if and only if the geometric entities to which these arcs correspond are not separated by any path of steepest descent. Thus, no watershed is divided by face boundaries of \mathcal{G}_{div} . The projection of a vertex v on the horizontal plane always lies in the *interior* of a face of \mathcal{G}_{div} , and never on a node or arc of \mathcal{G}_{div} . Thus every vertex of \mathcal{T} lies inside a well-defined watershed of a pit of the terrain.

We now find the watershed boundary of a point q as follows. If q is a pit, we report the boundary of the face $F(q)$ in \mathcal{G}_{div} that contains q . Otherwise, let q_L and q_R be the nodes in \mathcal{G}_{div} corresponding to the circular pieces around q immediately counterclockwise and clockwise, respectively, of the direction of steepest descent from q . The area $W(q)$ that drains through q is the area enclosed by the path in \mathcal{G}_{div} that follows the boundary of $F(q)$ clockwise from q_R to q_L (see Fig. 2).

We now sketch the data structure that makes it possible to trace the up-paths and the flow path from q efficiently and to trace the boundary of $W(q)$ without going down and back up arcs of \mathcal{G}_{div} in the interior of $W(q)$. Our solution consists of four ingredients: (1) the divide graph \mathcal{G}_{div} , stored face by face as explained below; (2) the river network, preprocessed for fast downstream traversals (with Hutchinson et al. [8]); (3) the strip map, stored strip by strip to facilitate fast traversals of steepest-ascent and steepest-descent paths and to provide pointers into the divide graph and the river network; (4) a point location structure on the strip map so that we can locate, for each query point q , the strip that contains it (with Arge and Vahrenhold [5]). Below we sketch our solution for storing a face of the divide graph. Details can be found in the long version of this paper.

Let p be a pit. The boundary of the face $F(p)$ of \mathcal{G}_{div} that represents $W(p)$ can be seen as a clockwise cycle γ with trees protruding to its right, into the watershed of p . Pick any node r that corresponds to a circular piece around p ; and let π be the path from r to a node $s(p)$ of the boundary γ , such that $s(p)$ is the first and only node of γ on π . Our idea is to “cut” the graph at r , and convert the boundary of $F(p)$ to a tree (see Fig. 4). To do this, we start by splitting all edges and vertices on π in a left copy and a right copy. Every right vertex is connected to the right edges incident to it, and to any trees that protrude into the watershed of p to the right of π . Furthermore, the right copy $s_R(p)$ of $s(p)$ is connected to the edge that follows $s(p)$ in γ . Every left vertex is connected to the left edges incident to it, and to any trees that protrude into the watershed

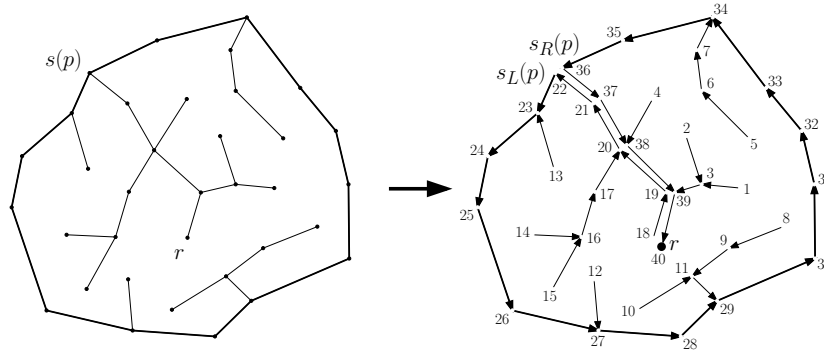


Fig. 4. Transforming the boundary of a face into a rooted tree. In the right figure, the edges are directed from child to parent, and numbers represent postorder ranks.

of p to the left of π . Furthermore, the left copy $s_L(p)$ of $s(p)$ is connected to the edge that precedes $s(p)$ in γ . We root the resulting tree at the right copy of r . We break up arcs that represent up-paths or down-paths in segments: one segment for each triangle crossed by the path, with vertices at the points where the path crosses a transfluent edge of the triangulation. Let the resulting tree be $\mathcal{B}(p)$; we store with each node its rank in a postorder traversal of the tree, and preprocess the tree for fast leaf-to-root traversals with the method of Hutchinson et al. [8].

To find the watershed boundary of a query point q , we briefly discuss the (interesting) case when q is in the interior of a channel. Assume we have the point location structure in [5] on the strip map so that we can locate the strips that have q at their feet. We follow the up-paths from q until they meet \mathcal{G}_{div} in points u_L and u_R on the ridges at the top of these strips. We then report the path between them that encloses the area upstream from q ; from our construction, this is the path that connects u_L and u_R in $\mathcal{B}(\text{pit}(q))$, where $\text{pit}(q)$ is the pit in the face of \mathcal{G}_{div} that contains q . The triangulation edges crossed by the up-paths from q to u_L and u_R are stored in the strips on each side of the channel, and they can be retrieved in a linear number of I/O's. The arcs of $\mathcal{B}(\text{pit}(q))$ that contain u_L and u_R are stored with the strip. The path between u_L and u_R is obtained from $\mathcal{B}(\text{pit}(q))$ in a linear number of I/O's by tracing it from u_L and u_R up to their lowest common ancestor, which is easily recognized using the post-order numbering of the nodes.

Let us denote $q(s) = O((s/B) \log_B s)$ and $l(s) = O(\log_B^2 s)$ the number of I/O's to build and query, respectively, a point location structure as described by Arge and Vahrenhold [5]. We have:

Theorem 5. *A data structure of size $O(s)$ for answering flow path and watershed boundary queries on an α -fat terrain \mathcal{T} can be computed in $O(q(s) + \text{Sort}(n/\alpha^2))$ I/O's, where s is the size of the strip map of \mathcal{T} . The structure reports the watershed boundary or the flow path of any query point q in $O(l(s) + k/B)$ I/O's, where k is the complexity of the answer.*

References

1. P. K. Agarwal, L. Arge, T. M. Murali, K. R. Varadarajan and J. S. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. *Symp. on Discrete Algorithms '98*, p117–126.
2. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
3. L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.
4. L. Arge, J. Chase, P. Halpin, L. Toma, D. Urban, J. S. Vitter, and R. Wickremesinghe. Flow computation on massive grid terrains. *GeoInformatica*, 7(4):283–313, 2003.
5. L. Arge and J. Vahrenhold. I/O-efficient dynamic planar point location. *Comp. Geom.* 29:147–162, 2004.
6. M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink and S. Yu. The complexity of rivers in triangulated terrains. *Canad. Conf. Comp. Geom. '96*, p325–330.
7. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff and J. S. Vitter. External-memory graph algorithms. *Symp. on Discr. Alg. '95*, p139–149.
8. D. Hutchinson, A. Maheshwari, and N. Zeh. An external memory data structure for shortest path queries. In *ACM-SIAM Computing and Combin. Conf. '99*, p51–60.
9. N. L. Jones, S. G. Wright, and D. R. Maidment. Watershed delineation with triangle-based terrain models. *Journal of Hydraulic Engineering*, 116(10), p1232–1251, 1990.
10. M. van Kreveld. Digital elevation models and TIN algorithms. In *Algorithmic Foundations of Geographic Information Systems*, LNCS 1340:37–78.
11. M. McAllister. *The computational geometry of hydrology data in geographic information systems*. PhD th., Univ. of British Columbia, 1999.
12. M. McAllister. *A watershed algorithm for triangulated terrains*. *Canad. Conf. Comp. Geom. '99*.
13. M. McAllister and J. Snoeyink. Extracting consistent watersheds from digital river and elevation data. *Ann. Conf. Amer. Soc. for Photogrammetry and Remote Sensing'99*.
14. E. Moet, M. v. Kreveld, and A. F. v/d Stappen. On realistic terrains. *Symp. on Computational Geom. '06*, p177–186.
15. O. Palacios-Velez, W. Gandoy-Bernasconi, and B. Cuevas-Renaud. Geometric analysis of surface runoff and computation of unit elements in distributed hydrological models. *J. Hydrology*, 211:266–274, 1998.
16. A. T. Silfer, G. J. Kinn, and J. M. Hassett. A geographic information system utilizing the triangulated irregular network as a basis for hydrologic modeling. In *Auto-Carto 8*, p129–136, 1987.
17. D. M. Theobald and M. F. Goodchild. Artifacts of TIN-based surface flow modeling. In *Proc. GIS/LIS'90*, p955–964, 1990.
18. G. Tucker, S. Lancaster, N. Gasparini, and S. Rybarczyk. An object-oriented framework for hydrology and geomorphic modeling using TINs. *Computers and Geosc.*, 27(8):959–973, 2001.
19. S. Yu, M. van Kreveld and J. Snoeyink. Drainage Queries in TINs: from local to global and back again. *Symp. on Spatial Data Handling '96*, p1–14.