Trekking in the Alps Without Freezing or Getting Tired*

Mark de Berg† Marc van Kreveld†

Abstract

Let F be a polyhedral terrain with n vertices. We show how to preprocess F such that for any two query points on F one can decide whether there exists a path on F between the two points whose height decreases monotonically. More generally, one can compute the minimum height difference along any path between the two points. It is also possible to decide, given two query points and a height, whether there is a path that stays below this height. All these queries can be answered with one data structure which stores the so-called height level map of the terrain. Although the height level map has quadratic worst case complexity, it is stored implicitly using only linear storage. The query time for all the above queries is $O(\log n)$ and the structure can be built in $O(n \log n)$ time. A path with the desired property can be reported in additional time that is linear in the description size of the path.

1 Introduction

Polyhedral terrains (mountain landscapes) are important concepts in several application areas of computational geometry, including Geographic Information Systems. For that reason they are well-studied in computational geometry and efficient algorithms have been given for e.g. hidden surface removal, ray shooting and intersections of terrains [1, 4, 6, 9, 11]. The problem of computing shortest paths in the Euclidean metric on polyhedral terrains has also received attention. See for instance Sharir and Shorr [12] and Chen and Han [5], who solve the shortest path problem on the surface of a simple polyhedron in $O(n^2)$ time. For a fixed target, they obtain a data structure such that the shortest path from a given query point to the target can be found in $O(\log n)$ time.

*This research was performed when the second author was affiliated at McGill University, and visited the first author at Utrecht University. The research of the first author is supported by the Dutch Organization for Scientific Research (N.W.O.) and the research of both authors by ESPRIT Basic Research Action 7141 (project ALCOM II: *Algorithms and Complexity*). The research of the second author was also supported by an NSERC international fellowship.

†Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

However, the Euclidean length is not the only important measure when walking in terrains. Indeed, the shortest path is probably not advisable if it involves going to great height, with the risk of freezing to death. Furthermore, it is well known that walking in mountainous terrain can be very tiresome, so it is wise to take a path that is as level as possible. These considerations have led us to study the following questions for a polyhedral terrain F:

1. Given two points s and t in F and a height z, is there a path between them that stays below height z?
2. Given two points s and t in F, is there a path between them that stays on one height?
3. Given two points s and t in F, is there a path between them whose height is monotonically decreasing?
4. Given two points s and t in F, what is the minimum summed height difference of any path between them?

Observe that question 2 is a special case of question 3, which, in turn, can be answered using question 4. For all of the above questions, we develop an optimal data structure that requires linear storage and answers queries in $O(\log n)$ time. An actual path can be found in additional time proportional to the description size of the path.

It turns out that in these questions the saddle vertices play an important role. (A vertex is called a saddle vertex if it has four incident edges that are going up, down, up, and down, when they are visited in cyclic order. See the next section for a more formal definition.) Imagine drawing for each saddle vertex all paths of constant height emanating from it. Project these paths onto the xy-plane and add the projections of all peak vertices and valley vertices (that is, the local maxima and minima). We call the resulting planar map the *height level map* of the terrain and we prove that this map contains all the information that is necessary to answer the above questions. Unfortunately, the complexity of the height level map can be quadratic. Therefore our data structure is an implicit representation of the map; it uses only linear storage and point location queries in the map can still be answered in logarithmic time.

The remainder of this paper is organized as follows. In Section 2 we give the necessary definitions. Properties of the height level map are proved in Section 3. In Section 4 we present the data structure that underlies the solution to all four query problems. The actual query algorithms are given in Section 5. Section 6 extends to the situation that vertices may have the same height. In Section 7 the concluding remarks are given.

2 Preliminaries

Let F^* be a polygonal terrain—the graph of a piecewise linear, continuous, bivariate real function—defined over the entire xy-plane. We shall restrict our attention to the part of this terrain defined on some rectangular area of the xy-plane. We denote this part by F and we define n to be the number of vertices of F, including vertices that arise because edges of F^* are clipped. The boundary of F, that is, the rectangle enclosing F, is denoted by ∂F. Vertices on ∂F are called *boundary vertices*; the remaining vertices are called *interior vertices*. We denote the height of a point p on F (that is, its z-coordinate) by $h(p)$, and we assume that no two vertices of F have the same height. In a later section we remove this restriction, but for the present it will make the description of our solution much simpler. Because we can triangulate every face of the terrain in linear time [3] we may assume without loss of generality that every face of F is a triangle. With a slight abuse of notation, we denote by F both the (clipped) polyhedral terrain in 3-space and its projection.[1]

For a vertex v of F, we denote by $N_1(v), \ldots, N_i(v)$ the cyclic sequence of neighbor vertices of v, ordered counterclockwise, where i is the degree of v in F. If v is a boundary vertex then we order its neighbors such that $N_1(v)$ and $N_i(v)$ are also boundary vertices. A vertex v is a *local maximum* if all neighbors of v have smaller height than v. A vertex v is a *local minimum* if all neighbors of v have greater height than v. A vertex that is a local maximum or local minimum is a *local extremum*. An interior vertex v is a *saddle vertex* if v has two higher neighbors and two lower neighbors which alternate around v. More precisely, there are four neighbors $N_{i_1}(v), N_{i_2}(v), N_{i_3}(v), N_{i_4}(v)$ of v with $1 \leq i_1 < i_2 < i_3 < i_4 \leq i$, such that $h(v) > h(N_{i_1}(v))$ and $h(v) < h(N_{i_2}(v))$ and $h(v) > h(N_{i_3}(v))$ and $h(v) < h(N_{i_4}(v))$, or such that $h(v) < h(N_{i_1}(v))$ and $h(v) > h(N_{i_2}(v))$ and $h(v) < h(N_{i_3}(v))$ and $h(v) > h(N_{i_4}(v))$. A boundary vertex v is a saddle vertex if there are three neighbors $N_{i_1}(v), N_{i_2}(v), N_{i_3}(v)$ of v with $1 \leq i_1 < i_2 < i_3 \leq i$, such that $h(v) > h(N_{i_1}(v))$ and $h(v) < h(N_{i_2}(v))$ and $h(v) > h(N_{i_3}(v))$, or $h(v) < h(N_{i_1}(v))$ and $h(v) > h(N_{i_2}(v))$ and $h(v) < h(N_{i_3}(v))$. A vertex that is a local extremum or a saddle vertex is a *special vertex*. For a height h, the *h-map of F* is defined to be the intersection of F with the horizontal plane $z = h$. Any h-map consists of cycles and paths. By non-degeneracy, it contains at most one vertex of F. Moreover, there is at most one point shared by two or more cycles or paths. If there is such a point, then it must be a saddle vertex v of F, and $h = h(v)$. Note that any path (which is not a cycle) starting at v must end at the boundary of F. Let v be a saddle vertex of F and consider the $h(v)$-map of F. We call the cycles and paths that contain v the *main component* of v, denoted $\mathcal{M}_v$. Finally, we define the *height level map* of F to be the planar map consisting of the main components of all saddle vertices of F, to which are added all local extrema, see Figure 1. We denote the height level map by $\mathcal{M}_F$. In the next section we will prove that $\mathcal{M}_F$ contains all the necessary information to solve the four above problems efficiently. Since any h-map has linear size, it follows

[1] Here and in the sequel, projection always means orthogonal projection onto the xy-plane.

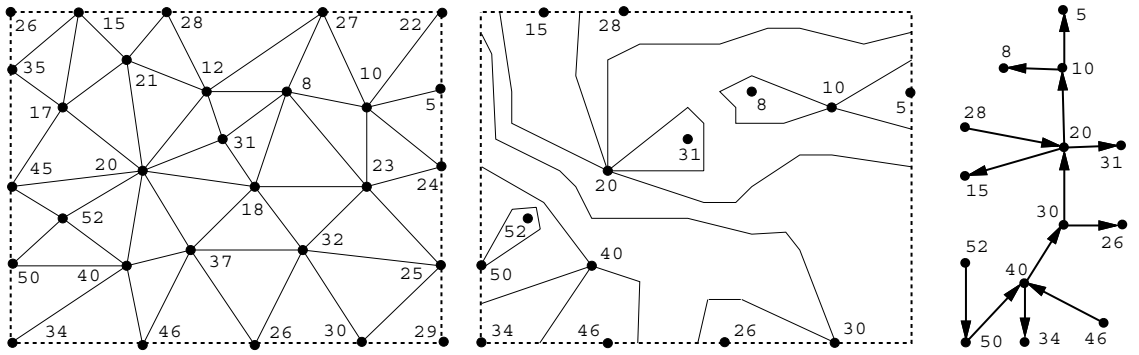

Figure 1: A polyhedral terrain, its height level map and the graph G_F on the special vertices.

that $\mathcal{M}_F$ has size $O(n^2)$. Unfortunately, there exist polyhedral terrains for which $\mathcal{M}_F$ really has quadratic size. Therefore, we devise a data structure that stores the height level map implicitly: using only linear storage, we can find the region of $\mathcal{M}_F$ that contains a given query point in $O(\log n)$ time.

A second structure we use is a directed graph G_F of which the nodes correspond to the saddle vertices and local extrema of F. For convenience, for any special vertex v of F, we denote the corresponding node in G_F also by v. There is a directed arc (v, w) in G_F if and only if $h(v) > h(w)$, and there is a region in $\mathcal{M}_F$ with v and w on the boundary of that region. We prove later that the undirected version T_F of G_F is actually a tree and that the degree of a node v in G_F is the number of regions incident to v. Hence, the local extrema correspond to the leaves of G_F. Furthermore, every region of $\mathcal{M}_F$ corresponds to one edge of G_F, and vice versa.

3 The height level map

In this section we prove several important properties of the height level map $\mathcal{M}_F$ and the corresponding graphs G_F and T_F.

Consider the regions in $\mathcal{M}_F$. They are bounded by pieces of the main components $\mathcal{M}_v$ of some saddle vertices v and by pieces of ∂F, and may have isolated boundary points (local extrema) in their interior. We define the *border* of any region R to be the boundary of R minus the pieces of ∂F that are not parts of main components; thus the border of R consists of pieces of the main components $\mathcal{M}_v$ of some saddle vertices v and of the local extrema in R. The following lemma shows that the border of a region is actually defined by exactly two special vertices.

Lemma 3.1 *Any region in $\mathcal{M}_F$ is incident to exactly two special vertices, which appear in different components of the border of that region.*

Proof: Let R be any region of $\mathcal{M}_F$. Let v_{min} be the vertex inside or on the boundary of R with minimum height. If v_{min} is a local minimum, then v_{min} is a special vertex by definition. We claim that if v_{min} is not a local minimum then it must be on the border of R and, hence, a saddle vertex. Indeed, if v_{min} is interior to R (or on $\partial R \cap \partial F$) but not a local minimum, then v_{min} must have some edges going down and crossing the border of R. But then v_{min} is higher than this part of the border of R and the corresponding saddle vertex. Similarly, we can show that the highest vertex v_{max} of R is a local maximum or saddle vertex. So there are at least two special vertices in R. Since all vertices have a unique height, these vertices cannot be on the same component of the border of R.

Suppose that there exists a third special vertex w in R. Assume without loss

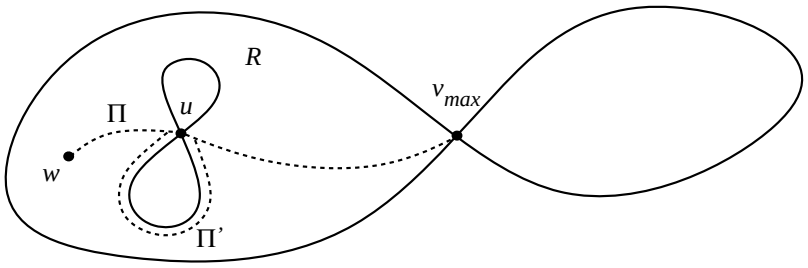

Figure 2: Illustration of the proof of Lemma 3.1.

of generality that w is locally maximal inside R, that is, w is a local maximum if we restrict our attention to the points in R. (Observe that a special vertex in R is always locally extreme inside R.) Let Π be a path in R from w to v_{max} such that the minimum height on Π is maximal. Since v_{max} and w are both local maxima in R, path Π must contain a lowest point u distinct from w and v_{max}. This point must be a saddle vertex, otherwise we obtain a contradiction with the fact that we chose the lowest point on Π as high as possible. Now consider the main component $\mathcal{M}_u$. If we can 'walk around' on $\mathcal{M}_u$, as in Figure 2, then we also obtain a contradiction with the definition of Π, because we can push Π a small distance ε away from $\mathcal{M}_v$ to obtain a higher path Π'. But if we cannot walk around, then v_{max} and w are not incident to the same region. □

Corollary 3.1 *T_F is a tree.*

Proof: It is easy to see that T_F is connected, so it remains to show that T_F does not contain any cycle. Obviously, there is no cycle of length two, so suppose for a contradiction that there is a cycle $v_1, v_2, \ldots, v_k, v_1$ with $v_1 \neq v_2 \neq v_k$. Consider node v_1. By the previous lemma v_1 cannot be local maximum or minimum, because such nodes have only one incident arc. So consider the main component $\mathcal{M}_{v_1}$. $\mathcal{M}_{v_1}$

partitions the plane into several cells. By definition of T_F there are no arcs between special vertices that are separated by $\mathcal{M}_{v_1}$, so $v_2, \ldots, v_k$ must all lie in the same cell. In particular, v_2 and v_k lie in the same cell. But this contradicts the previous lemma which implies that there is only one arc from each vertex into a certain cell. □

Lemma 3.2 *Two points s and t on F can be connected with a path on one height if and only if $h(s) = h(t)$ and s and t lie in the same region of the height level map.*

Proof: Suppose there is a constant height path between s and t. Obviously we must have $h(s) = h(t)$. Moreover, such a path cannot cross an edge of some main component and, hence, s and t are in the same region.

Now suppose for a contradiction that s and t lie in the same region R and $h(s) = h(t)$, but they cannot be connected with a constant height path. Assume without loss of generality that there is a path in R between s and t such that for the lowest point u on the path we have $h(u) < h(s)$. Let Π be such a path where $h(u)$ is maximal. As in the proof of Lemma 3.1, we can now show that u must be a saddle vertex and derive a contradiction. □

For any region R of $\mathcal{M}_F$, the *lower border* is the component of the border of R with height smaller than the interior of R. The *lower border vertex* of R is the saddle vertex or local minimum on the lower border of R. The lower border and lower border vertex of a point p are the lower border and lower border vertex, respectively, of the region R in which it lies. If p lies on the border of R, then the lower border vertex of p is the saddle vertex that also lies on that border. Similarly, we define the *higher border* and the *higher border vertex* of R and of p. For any point p, we denote the lower border vertex by $lbv(p)$ and the higher border vertex by $hbv(p)$.

Lemma 3.3 *Two points s and t on F with $h(s) > h(t)$ can be connected with a path of monotonically decreasing height if and only if s and t lie in the same region of the height level map or there exists a directed path in G_F from the node corresponding to $lbv(s)$ to the node corresponding to $hbv(t)$.*

Proof: We first show that any directed arc (u, v) in G_F implies the existence of a monotonically decreasing path from the special vertex u to the special vertex v on F. First, observe that there exists a region R for which u is the upper border vertex and v is the lower border vertex. Since the region R does not contain any local minima except v, we can find a monotonically decreasing path inside R from u to the main component of v. From such an intersection point, follow the main component using a constant height path until it reaches v. This path is clearly monotone and decreasing. For the same reason, there is a monotonically decreasing path from any point inside R to the lower border vertex and a monotonically increasing path to the higher border vertex.

Now, if s and t lie in the same component, then a decreasing path starting at s to the lower border vertex will contain some point in R on height $h(t)$. By the previous lemma, there is a constant height path between this point and t.

On the other hand, suppose that there exists a monotonically decreasing path Π from s to t. Let $R_1, \ldots, R_k$ be the sequence of regions in $\mathcal{M}_F$ that Π crosses. Since for every region R_i, there is an arc from the higher border vertex to the lower border vertex, it follows that Π is represented in the graph G_F. □

Lemma 3.4 *Let s and t be two points on F. If s and t lie in the same region of $\mathcal{M}_F$ then the minimum height difference on any path from s to t is $|h(s) - h(t)|$. Otherwise the minimum height difference is the minimum, over all four choices $v \in \{lbv(s), hbv(s)\}$, $w \in \{lbv(t), hbv(t)\}$, of the quantity $|h(s) - h(v)| + W(v, w) + |h(w) - h(t)|$, where $W(v, w)$ is the sum of the height differences over the edges of the path from v to w in T_F.*

Proof: The first part of the lemma—where s and t lie in the same region—follows directly from the previous lemma. Now consider the case where s and t lie in different regions. We have seen in the proof of the previous lemma that there are monotone paths from s to $lbv(s)$ and to $hbv(s)$, from t to $lbv(t)$ and to $hbv(t)$, and between any two special vertices v that define an edge in T_F. Hence, for every choice $v \in \{lbv(s), hbv(s)\}$, $w \in \{lbv(t), hbv(t)\}$ there really exists a path from s to t whose height difference is $|h(s) - h(v)| + W(v, w) + |h(w) - h(t)|$.

On the other hand, any path from s to t is represented in T_F, since for every region R_i that is crossed by the path there is an arc in T_F between the higher border vertex and the lower border vertex. □

Using a similar argument as in the previous two lemmas it is easy to show the following.

Lemma 3.5 *Two points s and t on F can be connected with a path that has height at most z, with $z \geq h(s)$ and $z \geq h(t)$, if and only if s and t lie in the same region of the height level map, or the path in T_F from the node corresponding to $lbv(s)$ to the node corresponding to $lbv(t)$ does not contain any nodes with height greater than z.*

4 The point location data structure

In this section we describe the data structure for point location in the height level map. More precisely, we will prove the following theorem.

Theorem 4.1 *Let F be a polyhedral terrain with n vertices. There exists a data structure for point location in the height level map $\mathcal{M}_F$ of the terrain which uses $O(n)$ storage and has $O(\log n)$ query time. The data structure can be built in $O(n \log n)$ time.*

In the next subsection we describe the search tree $\mathcal{T}$ in which the point location query is performed, and the query algorithm. After that we describe how the preprocessing is performed.

4.1 The structure and the query algorithm

Below we describe the data structure for point location in the height level map $\mathcal{M}_F$. For simplicity we start by assuming that every node in the tree T_F has constant degree or, in other words, that every main component consists of a constant number of distinct paths and cycles. After we have described the data structure we give the query algorithm. Conceptually, the query algorithm is very simple. However, it uses a somewhat complicated oracle which we describe separately. At the end of this subsection we show how to deal with saddle vertices that have non-constant degree in $\mathcal{M}_F$.

The search tree $\mathcal{T}$. Before we describe the search tree, recall that the arcs of the tree T_F correspond one-to-one with the regions of $\mathcal{M}_F$. The *hbv* and *lbv* of that region are the two nodes incident to that arc.

We construct a constant degree search tree $\mathcal{T}$ as follows. Let δ be the root of $\mathcal{T}$, and let it correspond to the whole polyhedral terrain F. Find a node u in the graph T_F such that the removal of u from T_F gives c subgraphs[2] with at most $\lfloor n/2 \rfloor$ nodes each. We add a copy of node u back to all c subgraphs, so that every arc in T_F appears in exactly one subgraph. Therefore, the subgraphs correspond exactly to the cells into which F is partitioned by the main component of u. We associate the saddle vertex u with δ. However, we do not store the main component explicitly, since it may have large complexity. For every subgraph that arises in the above way (or, equivalently, every cell defined by $\mathcal{M}_u$) we make a child node γ of δ which is the root of a recursively defined subtree for that subgraph (or, that cell). If the subgraph consists of one single arc then γ is a leaf of $\mathcal{T}$. The leaf γ represents the region of $\mathcal{M}_F$ that corresponds to the single arc. We store $lbv(R)$ and $hbv(R)$—the two nodes incident to the arc—explicitly at γ. See Figure 3 for an example of T_F and the search tree $\mathcal{T}$. The following lemma follows immediately from the definition of $\mathcal{T}$.

Lemma 4.1 *The tree $\mathcal{T}$ has linear size and it has depth $O(\log n)$.*

The conceptual query in $\mathcal{T}$. Let q be a point on F. We wish to know the region of $\mathcal{M}_F$ that contains it. In particular, we wish to know $lbv(q)$ and $hbv(q)$. The search with point q starts at the root δ of $\mathcal{T}$. Each of the children corresponds to a cell in the plane separated by the paths and cycles of the main component

[2]Even though T_F is a tree, we will speak of *subgraphs* of T_F to avoid confusion with *subtrees* of $\mathcal{T}$.

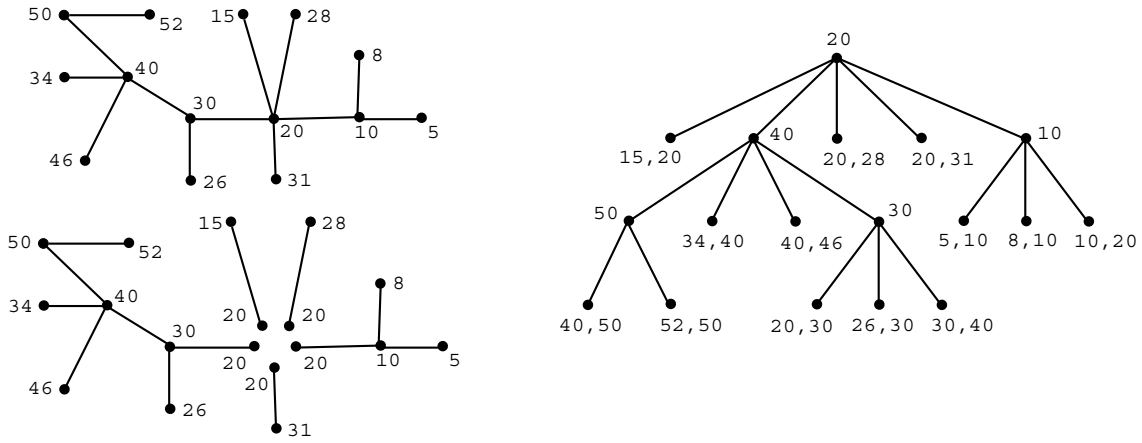

Figure 3: The tree T_F of Figure 1, its partitioning into subgraphs and the search tree $\mathcal{T}$. Note that there is precisely one leaf for every region of $\mathcal{M}_F$.

corresponding to δ. We use an oracle $\omega(q, \delta)$ to determine in which of the cells q lies. Then the search continues recursively in the subtree corresponding to that cell, until a leaf is reached. At the leaf $hbv(q)$ and $lbv(q)$ are stored. We obtain the following lemma.

Lemma 4.2 *The query time in the search tree $\mathcal{T}$ is $O(\log n)$ times the time needed for the oracle.*

The oracle for decisions in $\mathcal{T}$. We now explain how to implement the oracle $\omega(q, \delta)$. The oracle uses two data structures. First, we need a planar point location structure on the xy-projection of the terrain F. This structure also stores for every vertex v of F its hbv and lbv, which we have precomputed. (How this precomputation is done is discussed later.) Second, after assigning an arbitrary node of T_F to be the root, we preprocess T_F for $O(1)$ time lowest common ancestor queries [8].

We now show how to answer the oracle $\omega(q, \delta)$ using these two data structures. Recall that the oracle must determine in which cell q lies with respect to the cycles and paths corresponding to node δ. At the start of the query with q, we determine the triangle t_q of F that contains q. (If q lies on an edge, then we take one of the two triangles containing this edge. If q is a vertex then we are ready, since we know for each vertex its hbv and its lbv.) Let $v_{\max}$ be the vertex of triangle t_q with maximal height, and let $v_{\min}$ be the vertex of t_q with minimal height. We have precomputed $lbv(v_{\max})$, $hbv(v_{\max})$, $lbv(v_{\min})$ and $hbv(v_{\min})$, which correspond to nodes in the tree T_F. See Figure 4(i).

Let $\gamma_1, \ldots, \gamma_c$ be the children of δ in $\mathcal{T}$. Let us recall our convention that we denote special vertices in F and their corresponding nodes in T_F and in $\mathcal{T}$ by the same identifier. Also recall that the term 'subtree' is reserved for the search tree

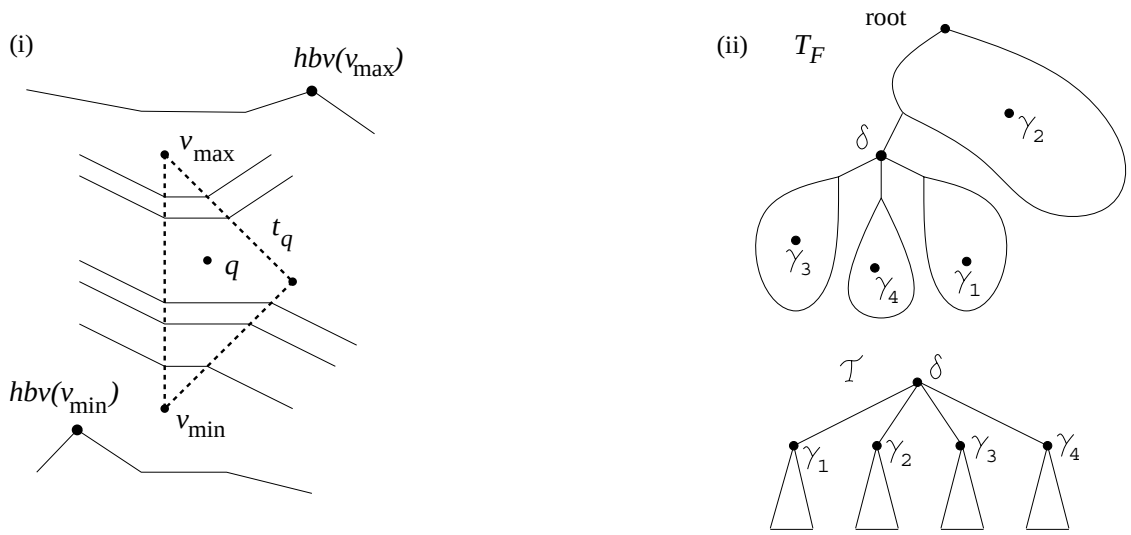

Figure 4: (i) The triangle t_q in a part of the height level map, and the vertices $hbv(v_{\max})$ and $lbv(v_{\min})$. (ii) Deciding in which subtree below δ to continue by means of lowest common ancestor queries in T_F.

$\mathcal{T}$ and that we use the term 'subgraph' for T_F. Finally, recall that a subtree of δ corresponds to a cell defined by the main component of the saddle vertex δ. To determine the subtree of δ in which to continue the search, we should determine which cell contains q. To this end we first determine which of the subtrees of δ contains $hbv(v_{\max})$, as follows. Observe (see also Figure 4(ii)) that only one of the children γ_i of δ in $\mathcal{T}$ is not a descendant of δ in T_F. We can decide which child this is by computing the lowest common ancestor in T_F of each γ_i with δ; the unique γ_j (γ_2 in Figure 4(ii)) for which the answer is not δ is the one that is not a descendant in T_F. We first test if $hbv(v_{\max})$ lies in the subtree of $\mathcal{T}$ rooted at this γ_j; this is the case if and only if the lowest common ancestor of δ and $hbv(v_{\max})$ is not δ. If $hbv(v_{\max})$ does not lie in the subtree of γ_j then we test the other children of δ: $hbv(v_{\max})$ lies in the subtree of $\mathcal{T}$ rooted at γ_i if and only if the lowest common ancestor of $hbv(v_{\max})$ and γ_i is not δ. (This last test is correct because we already know that $hbv(v_{\max})$ is a descendant of δ in T_F.)

In the same way we compute the subtree of $\mathcal{T}$ that contains $lbv(v_{\min})$. If $hbv(v_{\max})$ and $lbv(v_{\min})$ lie in the same subtree then q must lie in a region to be found in that subtree as well and we have solved the oracle. (If either one of $hbv(v_{\max})$ or $lbv(v_{\min})$ is δ, then q must lie in the subtree chosen by the other of $hbv(v_{\max})$ and $lbv(v_{\min})$.) The reason that q must lie in that subtree is the following. The cells corresponding to the subtrees are separated by curves of the main component $\mathcal{M}_\delta$ that contains the saddle vertex δ. Recall that $\mathcal{M}_\delta$ contains edges on one height only. Therefore, the triangle t_q of F intersects $\mathcal{M}_\delta$ in at most one edge. Since $hbv(v_{\max})$

and $lbv(v_{\min})$ lie in the same cell of $\mathcal{M}_\delta$, so must the whole triangle t_q.

On the other hand, if $hbv(v_{\max})$ and $lbv(v_{\min})$ lie in different subtrees then t_q intersects $\mathcal{M}_\delta$ and we do not know yet which of the two subtrees contains the region with q. There are two candidates: the subtree that contains $hbv(v_{\max})$ and the subtree that contains $lbv(v_{\min})$. However, we know the height of q, the height of the saddle vertex δ and the heights of $hbv(v_{\max})$ and of $lbv(v_{\min})$. We have $h(hbv(v_{\max})) > h(\delta) > h(lbv(v_{\min}))$. If $h(q) > h(\delta)$, then the search with q should continue in the same subtree of $\mathcal{T}$ as $hbv(v_{\max})$. If $h(q) < h(\delta)$, then the search with q should continue in the same subtree of $\mathcal{T}$ as $lbv(v_{\min})$. If $h(q) = h(\delta)$, then q lies on the main component $\mathcal{M}_\delta$.

We conclude that we can determine the cell of $\mathcal{M}_\delta$ containing q with a constant number of lowest common ancestor queries in T_F. So we obtain the following:

Lemma 4.3 *The oracle queries can be answered in $O(1)$ time with a structure that uses $O(n)$ storage.*

Accounting for saddle vertices with non-constant degree. Let v be a node in the unrooted tree T_F with degree $k > 3$. We replace v with a binary tree $T(v)$ of $k-2$ nodes $v_1, \ldots, v_{k-2}$. The k edges that were incident to node v are now used to connect $T(v)$ to the rest of T_F, as in Figure 5. After this augmentation we proceed

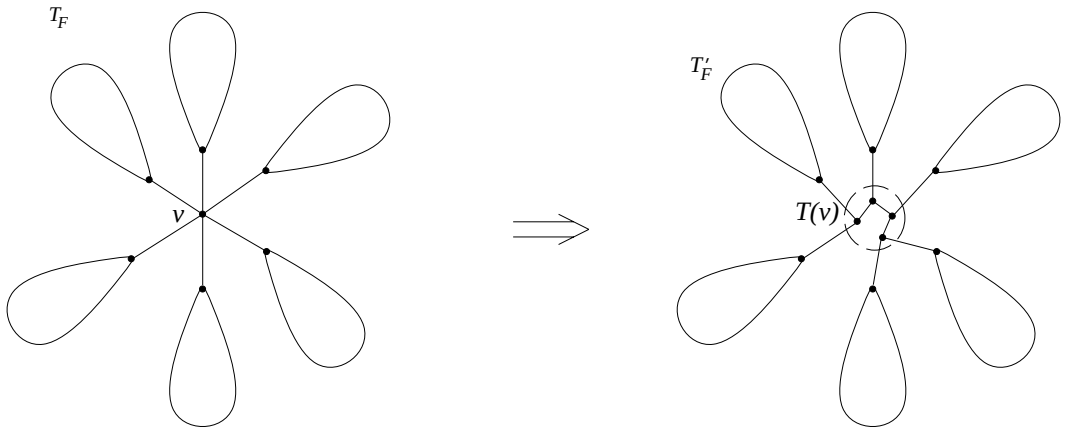

Figure 5: Replacing a node of degree $k > 3$ by a subtree.

in the same way as before: we assign an arbitrary node of the augmented tree T'_F to be the root, preprocess it for lowest common ancestor queries, and construct the search tree $\mathcal{T}$ from this graph. A connection of the nodes of $\mathcal{T}$ with the geometry of the main components in the height level map has become less clear. Still, every leaf node represents a part of the height level map which may either be a region or a (part of a) main component. Notice that the augmentation can be done in linear time, so it does not influence the asymptotic preprocessing time.

There is another matter to deal with: we stored for every vertex u its hbv and lbv and we should be careful if one or both of these nodes is split. If, for example, $hbv(u)$ is split we have to decide which node in $T(hbv(u))$ is going to be the representative of $hbv(u)$. To see what happens we must go back to the oracle, where the hbv's and lbv's are used. Recall that the oracle $\omega(q, \delta)$ uses the special nodes $hbv(v_{\max})$ and $lbv(v_{\min})$, where $v_{\max}$ is the highest vertex of the triangle t_q containing the query point q and $v_{\min}$ is the lowest vertex of t_q. The oracle determines the position of $hbv(v_{\max})$ and $lbv(v_{\min})$ with respect to δ in T_F, and it uses this information to decide the position of q in $\mathcal{M}_\delta$. In our augmented tree T'_F we may have replaced, say, $hbv(v_{\max})$ by a subtree $T(hbv(v_{\max}))$. Clearly, if $\delta \neq hbv(v_{\max})$ then the position with respect to δ is the same for each of the nodes in $T(hbv(v_{\max}))$. So in this case it does not matter which node in $T(hbv(v_{\max}))$ is chosen as the representative of $hbv(v_{\max})$. But when $\delta = hbv(v_{\max})$ then the answer to the oracle is determined by the position of $lbv(v_{\min})$. So also in this case we can take any node as representative. In both cases the query algorithm remains correct.

We conclude that we can handle nodes with non-constant degree without affecting the query or preprocessing times.

4.2 The preprocessing

In the preprocessing phase we have to construct the search tree $\mathcal{T}$ and the data structures for the oracle. The oracle uses two data structures: a point location structure for the terrain F with the hbv and lbv stored at every vertex and the tree T_F preprocessed for lowest common ancestor queries. The point location structure for F can be constructed in linear time [7, 10] once the hbv and lbv are known and preprocessing T_F for lowest common ancestor queries also takes linear time [8]. Furthermore, $\mathcal{T}$ can be constructed in $O(n \log n)$ time once T_F has been computed. (This follows from the fact that a node δ which splits T_F into subgraphs with at most $\lfloor n/2 \rfloor$ nodes each can be found in linear time.)

Hence, the most difficult tasks during the preprocessing are to compute the tree T_F—which amounts to computing for every saddle vertex v which other special vertices are incident to the regions incident to v—and to compute $hbv(u)$ and $lbv(u)$ for every vertex u of the terrain. To this end we first present a divide-and-conquer algorithm that computes for every vertex u of the terrain—except for the local maxima and saddle vertices—the lowest vertex of the terrain which can be reached from u with a path whose height is (strictly) monotonically increasing. We call this vertex the *next higher vertex for* u. The algorithm also computes for each saddle vertex v the *next higher vertices*, one for each region incident to and higher than v. After that we show how to use this information to compute T_F and the hbv and lbv of each vertex u.

Let F be a terrain with n vertices and let d be the number of distinct heights of the vertices in F. (Initially, all vertices of F have different heights, but the algorithm will create new vertices with the same height as vertices in F.) If all vertices have

the same height—that is, $d = 1$—then we are finished: none of the vertices can reach any other vertex with a strictly monotonically increasing height path. When $d > 1$ we compute a value h_{mid}, different from the heights of the vertices, such that the number of distinct heights less than h_{mid} and the number of distinct heights greater than h_{mid} are both at most $\lceil d/2 \rceil$. We delete all the edges which intersect the plane $z = h_{mid}$. This splits F into a number of connected components, the 'subterrains' $F_1, F_2, \ldots$. The idea is to recurse on these subterrains. This will tell us for every vertex u its next higher vertex in the subterrain containing u. The next higher vertex in any other subterrain is found in a 'merging step'. We have to exercise some care, however, so that each vertex in a subterrain can still reach every vertex in the subterrain (with a path of monotonically increasing height) that it could reach before. We make sure that this is the case by adding to every subterrain some new vertices in a way to be described next.

Let us call two edges which are intersected by the plane $z = h_{mid}$ and which are edges of the same triangle *neighbors*. Observe that any triangle has either zero or two edges intersected by the plane $z = h_{mid}$. Hence, if we group the set of intersected edges by taking the transitive closure of the neighbor relation, we obtain a number of groups $E_1, E_2, \ldots, E_k$ where every E_i consists of a sequence (or a cycle) of edges such that the adjacent edges in the sequence (cycle) are neighbors. Note that the removal of the edges in some group E_i splits the terrain into two subterrains. Hence, the groups $E_1, E_2, \ldots, E_k$ split the terrain into $k + 1$ subterrains $F_1, \ldots, F_{k+1}$. See Figure 6(i).

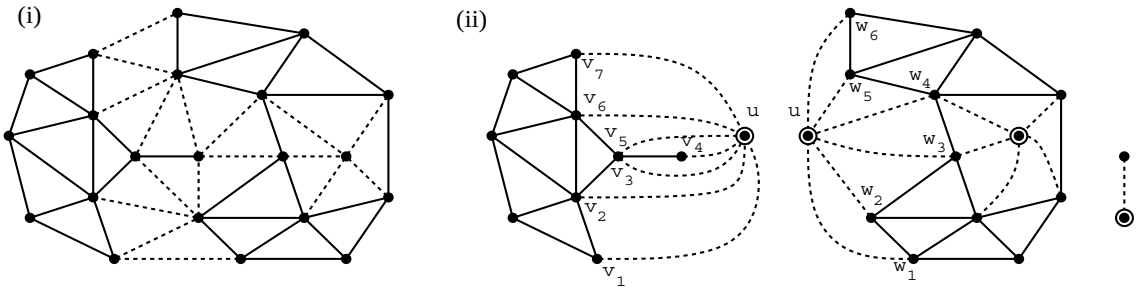

Figure 6: (i) A terrain F that is split by a path of edges and a cycle of edges (dashed) into three components. (ii) The three components and how they are restored.

Consider one group E_i of edges. Let $v_1, \ldots, v_m$ be the sequence (or cycle) of endpoints of the edges in E_i which are higher than the split value h_{mid}, where duplicates of the same vertex that are adjacent in the sequence have been removed. Let F_j be the subterrain that contains the vertices $v_1, \ldots, v_m$. For some vertices in F_j the path to the next higher vertex—even if this vertex is also in F_j—may have passed through triangles that are incident to the edges in E_i. These triangles are destroyed by the removal of E_i. To make sure that these vertices can still reach

their next higher vertex inside F_j we add a new vertex u to F_j with an edge from u to every $v_1, \ldots, v_m$. The height of u is chosen equal to the height of the lowest of $v_1, \ldots, v_m$. This creates $m-1$ triangles if E_i is a sequence (and m triangles if E_i is a cycle). See Figure 6(ii). Note that in the resulting graph the sequence of neighbors around any vertex is fixed, and that it is planar and triangulated (except the outer face). However, it may contain multiple edges between two vertices. Consequently, a straight line embedding does not exist. But this is of no concern to us, since we only work with the underlying graph of the terrain and the heights of its vertices. For convenience we will view the new subterrain F_j as a terrain with faces that can be bounded by curves. Notice that multiple edges are never adjacent, so the graph has complexity linear in the number of vertices.

For the sequence $w_1, \ldots, w_{m'}$ of endpoints incident to the edges in E_i that lie lower than h_{mid} we use a similar strategy. Thus we add a new node u' to the subterrain $F_{j'}$ that contains the nodes $w_1, \ldots, w_{m'}$. The height of u' is equal to the maximum height among the $w_1, \ldots, w_{m'}$ and it has edges to each of these vertices.

We claim that after this has been done for every group E_i, each vertex can reach the same set of vertices within its subterrain as it could in the whole terrain F. Let us denote the subterrain F_j augmented with the extra vertices by $\widetilde{F}_j$.

Lemma 4.4 *Let v, w be two vertices in F that lie in the same subterrain F_j. There is a path from v to w in F whose height is monotonically increasing if and only if there is such a path in $\widetilde{F}_j$.*

Proof: Suppose that there is a path Π from v to w in F whose height is monotonically increasing. If Π remains inside F_j then there trivially exists a path from v to w in $\widetilde{F}_j$ whose height is monotonically increasing. Otherwise, let Π' be a maximal subpath of Π that is outside F_j. The two endpoints p_1, p_2 of Π' must lie on two edges (v_{i_1}, v_{i_2}) and (v_{i_3}, v_{i_4}) that are on the same component of the boundary of F_j. Hence, an extra vertex u has been added, with edges to all vertices on that boundary component. It is easily verfied that the choice of the height of u implies that there exists an increasing path from p_1 to p_2 through the new triangles incident to u. Hence there is an equivalent of the subpath Π' in $\widetilde{F}_j$. It follows that there is a path from v to w in $\widetilde{F}_j$ whose height is monotonically increasing.

The reverse is proved in a similar way. □

The above lemma implies that the next higher vertex of some vertex v will be found if it is in the same subterrain F_j. The next lemma shows that the vertices for which this is not the case are easily identified and handled correctly.

Lemma 4.5 *Let v be a vertex in F_j whose next higher vertex w is in a different subterrain $F_{j'}$. Then F_j lies below the plane $z = h_{mid}$, vertex v is incident to a deleted edge e, and v is locally maximal inside F_j. Let E_i be the group of deleted edges containing e and let W_i be the set of higher endpoints of the edges in E_i. Then w is the lowest vertex in W_i.*

Proof: Note that any path from v to another subterrain $F_{j'}$ must cross the plane $z = h_{mid}$. It follows that the height of w is greater than h_{mid}, and that the subterrain F_j containing v must lie below $z = h_{mid}$. Assume that v is not locally maximal inside F_j. Then v can reach a neighbor in F_j whose height is less then h_{mid} with a path whose height increases monotonically, contradicting the statement that w is in a different subterrain. If v is locally maximal inside F_j then it must clearly be on the boundary of F_j, otherwise it would be a local maximum in F and, hence, have no next higher vertex. Thus v is incident to a deleted edge e, as claimed.

It is easy to see that from v we can reach any vertex in W_i—and in particular the lowest one—with a path of monotonically increasing height via the triangles incident to the edges in E_i. On the other hand a monotonically increasing path from v to a vertex w' not in W_i must cross an edge between two vertices in W_i and, hence, w' is higher than the lowest of these two vertices. □

Summarizing, our algorithm works as follows. We split F into subterrains by deleting the edges that intersect the plane $z = h_{mid}$. We augment the subterrains F_j to obtain new subterrains $\widetilde{F}_j$ by adding two extra vertices for each of the deleted groups E_i of edges as described above. Inside each $\widetilde{F}_j$ we find find the next higher vertices recursively. After returning from all recursive calls, we compute for each vertex v whose next higher vertex is not in the same subterrain its correct next higher vertex, according to the lemma above. This can be done in linear time. Observe that a subterrain may contain *added vertices*, which are not on the original terrain F but were added to restore paths after previous splits, and *real vertices*, which are on F. So strictly speaking the lowest vertex in W_i mentioned in Lemma 4.5 is not unique. Of course we should assign a real vertex as next higher vertex. Because all real vertices have distinct heights this vertex is uniquely determined.

The recursion ends when all vertices in a subterrain have the same height. For such a subterrain we do not have to do anything. The vertices will be assigned the correct next higher vertex when we return from the recursion.

Notice that the algorithm finds more than one next higher vertex for each saddle vertex, since a saddle vertex is adjacent to several groups of deleted edges. (See for example the vertices w_3 and w_4 in Figure 6, of which one could be a saddle vertex.) In fact, one easily verifies that a next higher vertex is found for each incident region that is higher than the saddle vertex, which is exactly what we needed.

Lemma 4.6 *The time required to compute for every vertex the next higher vertex (or next higher vertices for a saddle vertex) is $O(n \log n)$.*

Proof: First we show that the number of extra vertices created during the preprocessing is $O(n)$. Suppose we create new vertices u and u' for a group E_i of deleted edges that is connected to sequences $v_1, \ldots, v_m$ and $w_1, \ldots, w_{m'}$ of vertices. One easily shows by induction that E_i has $m + m' - 1$ edges which are all non-horizontal. The new vertices u and u' are connected to $v_1, \ldots, v_m$ and $w_1, \ldots, w_{m'}$ using m and

m' edges, respectively, of which two are horizontal. Therefore, there is one fewer non-horizontal edge for every two added vertices, and it follows that the number of added vertices is bounded by twice the number of edges of F. Hence, the total number of added vertices is $O(n)$.

Next, we show that the depth of the recursion is $O(\log n)$. Denote the number of distinct z-coordinates of the terrain by d. The choice of h_{mid} ensures that the number of distinct z-coordinates in each subterrain is at most $\lceil d/2 \rceil$. Added vertices always get a height equivalent to the height of a real vertex in that subterrain. Hence, the recursion depth is $O(\log d) = O(\log n)$.

The lemma now follows from the fact that the divide step and the merge step take linear time. □

Once we have computed for every vertex the next higher vertex, the higher border vertex *hbv* can be found easily. Consider the graph on all vertices, with directed arcs from every vertex to the next higher vertex (or to the next higher vertices in case of a saddle vertex). This graph is a directed acyclic graph with a distinguished set of special vertices, namely the saddle vertices and local maxima. To compute the *hbv* of every vertex, we have to determine the first special vertex that can be reached with a directed path. To compute for a saddle vertex v all special vertices that are incident to a region that is incident to and higher than v, we have to find the first special vertex that can be reached via each outgoing edge of v. All of this can be done in linear time by a standard graph traversal algorithm.

We have shown how to compute for every vertex its *hbv* and for every saddle vertex the other special vertices that are incident to a region which is incident to and higher than v. In the same way we can compute the *lbv* of every vertex of F, and for each saddle vertex v all special vertices that are incident to a region that is incident to and lower than v. This gives us all the information needed to construct T_F, leading to the following lemma.

Lemma 4.7 *The preprocessing time for the data structure for point location in $\mathcal{M}_F$ is $O(n \log n)$.*

This finishes the proof of Theorem 4.1.

5 Answering the queries

In this section we describe the query algorithms to the four problems stated in the introduction of this paper. The solutions are based on the properties of the height level map $\mathcal{M}_F$ that were given in Section 3 and they make use of the data structures described in the previous section.

Constant-height paths. Given two points s and t with equal height, we wish to determine whether there exists a path between them that stays on the height $h(s) = h(t)$. By Lemma 3.1, such a path exists if and only if s and t lie in the same region of $\mathcal{M}_F$. Therefore, to solve the query, we determine $hbv(s)$, $lbv(s)$, $hbv(t)$ and $lbv(t)$. Since the hbv and lbv of any point determines uniquely the region of $\mathcal{M}_F$ that contains it, we conclude that there is a constant height path between s and t if and only if $hbv(s) = hbv(t)$ and $lbv(s) = lbv(t)$.

The other queries. We now discuss monotone paths, minimum height difference paths and paths below a given query height z. The solutions to these problems turn out to be very similar.

Consider the minimum height difference query. To be able to answer these queries we prepocess T_F for so-called weighted subpath queries: given two nodes in T_F, compute the sum of the weights of the arcs on the path between the two nodes. Such queries can be answered in $O(\alpha(n))$ time after $O(n)$ preprocessing [2] (see also [13, 14]). In fact, the function *sum* can be replaced by any other function which, together with the weights, forms a commutative semigroup (for instance, the function *maximum*).

To solve the minimum total height difference problem on F, we let the arcs in T_F be weighted by the height difference of their endpoints. Given two query points s and t, we first perform a point location in the height level map $\mathcal{M}_F$ using the data structure described in the previous section. This will give us the vertices $s_1 = hbv(s)$, $s_2 = lbv(s)$, $t_1 = hbv(t)$ and $t_2 = lbv(t)$. Then we determine the weights of the subpaths of s_1 and t_1, of s_1 and t_2, of s_2 and t_1, and of s_2 and t_2, denoted by W_{11}, W_{12}, W_{21} and W_{22}, respectively. By Lemma 3.4 the answer to the query is the minimum weight of $W_{ij} + |h(s_i) - h(s)| + |h(t_j) - h(t)|$, where $i, j \in \{1, 2\}$. Note that this also solves the monotone path problem: we simply check whether the total weight is equal to the absolute height difference $|h(s) - h(t)|$ of s and t.

To solve the paths below height z problem, we take a similar approach. However, we weight the arcs of T_F with the maximal height of their two endpoints. Furthermore, we do not compute the sum of the weights of a subpath, but the maximum. With the above notation, we need only test the weight (maximum height) of the subpath between s_2 and t_2 in T_F. See Lemma 3.5. If the weight is at most z, then there exists a path between s and t with height at most z.

Theorem 5.1 *Let F be a polyhedral terrain with n vertices with distinct heights. F can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two query points one can determine in $O(\log n)$ time whether there exists a path between them that has constant height or that is monotone. It is also possible to compute in $O(\log n)$ time the minimum height difference of any path between the two query points, and to decide, given two query points and a query height z, in $O(\log n)$ time if there is a path that stays below height z (assuming a RAM model of computation).*

Reporting a path. In this paragraph we explain how to report a path with the desired property in time proportional to the complexity of the path. From the preceding discussion it should be clear that we only need one new subroutine: given two points s and t in the same region R of $\mathcal{M}_F$ (possibly on the boundary), report a monotone path between them. Next we show how to implement this subroutine.

First, consider the case where $h(s) = h(t)$ and we want to find a constant height path. Because all vertices are at different heights there are only two ways to leave point s if we want to stay on the same height, unless s is a saddle vertex. We will explore these two constant height paths using a *tandem search*: we go one step on one path, one step on the other path, one step on the first path, and so on. Here taking one step means crossing one triangle. At some point one of the two paths must reach t (this is true because s and t are in the same region) and we are ready. Notice that it may happen that one of the two paths gets stuck at the boundary ∂F; in that case we only continue with the other path. In this way the time that we have spent is proportional to the number of triangles crossed by the path to t. It takes only constant time to cross a triangle: we simply go with one single link from a point on one edge of the triangle to the unique point on another edge with the same height. If this second point is on the interior of the edge then we immediately know the next triangle to cross. If, however, this other point is a vertex then we need some extra information. So we determine in the preprocessing phase for every non-special vertex the two incident triangles on which constant height paths through v must pass. The time needed during the preprocessing to compute this extra information is $O(n)$. Now we can find a constant height path from s to t in time proportional to its complexity. If s is a saddle vertex, then we start a tandem search from t to find the saddle vertex.

Next we look at the case where s and t have different heights. First, assume that s does not lie on the boundary of region R and $h(s) > h(t)$. We choose a path from s that goes down across triangles and edges of F until it reaches the height $h(t)$. Then a constant height path is found as before. If s does not lie on the boundary of R and $h(s) < h(t)$ we choose a path that goes up in a similar way. Next, assume that s lies on a main component that bounds the region R. The discussion of constant height paths shows how to go from s to the saddle vertex v_i of that main component. Assume first that $h(s) = h(v_i) > h(t)$ and let v_{i+1} be the lower boundary vertex of R. We must choose the correct path from v_i that goes down, because otherwise we might enter the wrong region. During the preprocessing, we store with every arc of T_F between saddle vertices v_i and v_{i+1} one edge of F incident to v_i and one incident to v_{i+1}. These two edges enter the region bounded by the main components of v_i and v_{i+1}. Using this extra infomation, we can easily decide how to leave v_i to get to v_{i+1} and thus enter R. The rest is as before. If $h(s) = h(v_i) < h(t)$ similar actions are taken.

Theorem 5.2 *Using $O(n)$ additional preprocessing time and $O(n)$ additional space, a path between two query points that has minimum height difference, or a path that*

stays below a given height z, can be reported in $O(\log n + k)$ time, where k is the complexity of the path.

Note that we have not made any claim on the value of k—indeed, our method may report a path with many links when there is exists a path with only few links that also satisfies the height properties.

6 Dealing with vertices of the same height

Let F be a polyhedral terrain, but this time without the restriction that all vertices have different heights. We discuss in what way to adapt the definitions and techniques so that our results are still valid.

Firstly, we remove all vertices v for which all neighbors have the same height as v. We retriangulate the face in which v was contained, and observe that the new terrain is another graph of the same continuous bivariate real function as the old terrain. We adapt our definitions as follows. A vertex v is a local maximum (minimum) if no neighbor of v is strictly higher (resp. lower) than v. An interior vertex v is a saddle vertex if there are four neighbors $N_{i_1}(v), N_{i_2}(v), N_{i_3}(v), N_{i_4}(v)$ of v with $1 \leq i_1 < i_2 < i_3 < i_4 \leq i$, such that $h(v) \geq h(N_{i_1}(v))$ and $h(v) \leq h(N_{i_2}(v))$ and $h(v) \geq h(N_{i_3}(v))$ and $h(v) \leq h(N_{i_4}(v))$, or such that $h(v) \leq h(N_{i_1}(v))$ and $h(v) \geq h(N_{i_2}(v))$ and $h(v) \leq h(N_{i_3}(v))$ and $h(v) \geq h(N_{i_4}(v))$. For boundary vertices the definition is similar. The definition of main components and the height level map is as before, but now all special vertices have a main component, not just the saddle vertices. See Figure 7 for a height level map in which vertices have the same height. Notice that—although there may be a constant height path between a local maximum and a local minimum—one vertex cannot be a local maximum and a local minimum at the same time. However, it is possible that a vertex is classified as local maximum and saddle vertex at the same time. But this does not impose any problems.

Any region of the height level map is still bounded by two borders, the higher border and the lower border. But both borders may contain many special vertices of the same height, which implies that the lower and higher border vertices are not necessarily unique for a region. However, it still holds that for any region in the height level map, the two main components bounding it uniquely determine that region (i.e., there cannot be two regions bounded by the same main components). For the definition of T_F, we let its nodes correspond one-to-one to the main components rather than the special vertices on them. If we use *hbv* and *lbv* to denote the higher and lower main component that bound a region, then all results in the paper remain valid.

Let us briefly discuss the consequences of this slightly different definition of *hbv* and *lbv*. First, the formulation of Lemma 3.1 has to be changed according to the remark above. The proof of this lemma and all the other results of Section 3 now go through almost verbatim. In Subsection 4.1 only the oracle uses some geometric

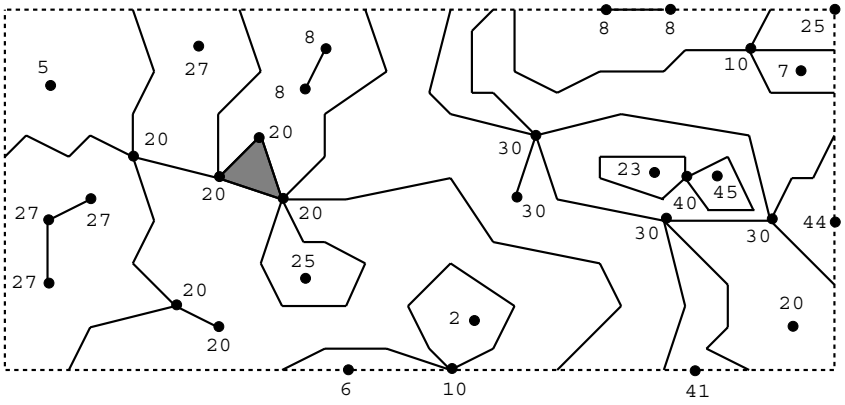

Figure 7: The height level map of a terrain with vertices of the same height.

information (refer to Figure 4) to decide in which subtree of $\mathcal{T}$ to continue. One easily checks that the oracle still decides correctly when there are more vertices with the same height. The preprocessing algorithm in Subsection 4.2 has to be changed slightly; the way this has to be done is rather straightforward, so we leave the details to the reader. To report a path some adaptations need also be made, but we do not think it is worth the effort to describe any details here.

7 Conclusions

This paper dealt with several path problems in polyhedral terrains, in which the height of paths played the most important role. We captured the structure of a polyhedral terrain by means of a height level map, and showed that it contains all necessary infomation. Then we showed how the height level map could be stored implicitly for point location, using only linear storage and $O(n \log n)$ preprocessing time. Path queries between any two points on the terrain can be answered in $O(\log n)$ time. We also showed how to report a path with the desired height properties, but no bound on the combinatorial or Euclidean length of the path could be given. This is an interesting open problem that remains. It would also be interesting to combine the work of this paper with the results from [5, 12], to obtain paths that are good both with respect to height difference and to Euclidean length. Furthermore, in some applications it is required that very steep parts of the terrain are avoided. This is also well worth studying.

References

[1] Bern, M., D. Dobkin, D. Eppstein, and R. Grossman, Visibility with a Moving Point of View, *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms* (1990), pp. 107–117.

[2] Chazelle, B., Computing on a Free Tree via Complexity-Perserving Mappings, *Algorithmica* **3** (1987), pp. 337–361.

[3] Chazelle, B., Triangulating a Simple Polygon in Linear Time, *Discrete Comput. Geom.* **6** (1991), pp. 485–524.

[4] Chazelle, B., H. Edelsbrunner, L. J. Guibas, and M. Sharir, Lines in Space: Combinatorics, Algorithms and Applications, *Proc. 21nd ACM Symp. on the Theory of Computing* (1989), pp. 382–393.

[5] Chen, J., and Y. Han, Shortest Paths on a Polyhedron, *Proc. 6th ACM Symp. on Comp. Geometry* (1990), pp. 360–369.

[6] Cole, R., and M. Sharir, Visibility Problems for Polyhedral Terrains, *J. Symb. Computation* **7** (1989), pp. 11–30.

[7] Edelsbrunner, H., L. J. Guibas, and J. Stolfi, Optimal Point Location in a Monotone Subdivision, *SIAM J. Computing* **15** (1986), pp. 317–340.

[8] Harel, D., and R. E. Tarjan, Fast Algorithms for Finding Nearest Common Ancestors, *SIAM J. Computing* **13** (1984), pp. 338–355.

[9] Katz, M.J., M.H. Overmars and M. Sharir, Efficient Hidden Surface Removal for Objects with Small Union Size, *Computational Geometry: Theory and Applications* **2** (1992), pp. 223–234.

[10] Kirkpatrick, D. G., Optimal Search in Planar Subdivisions, *SIAM J. Computing* **12** (1983), pp. 28–35.

[11] Reif, J., and S. Sen, An Efficient Output-Sensitive Hidden Surface Removal Algorithm and its Parallelization, *Proc. 4th ACM Symp. on Comp. Geometry*, 1988, pp. 193–200.

[12] Sharir, M., and A. Schorr, On Shortest Paths in Polyhedral Spaces, *SIAM J. Comput.* **15** (1986), pp. 193–215.

[13] Sleator, D.D., and R.E. Tarjan, A data structure for dynamic trees, *J. Comp. Syst. Sci.* **26** (1983), pp. 362–391.

[14] Sleator, D.D., and R.E. Tarjan, Self-adjusting binary search trees, *J. ACM* **32** (1985), pp. 652–686.