A faster algorithm to compute the visibility map of a 1.5D terrain

Maarten Löffler* Maria Saumell† Rodrigo I. Silveira‡

Abstract

Given a 1.5D terrain, i.e., an x-monotone polygonal line in $\mathbb{R}^2$ with n vertices, and $1 \leq m \leq n$ viewpoints placed on some of the terrain vertices, we study the problem of computing the parts of the terrain that are visible from at least one of the viewpoints. We present an algorithm that runs in $O(n + m \log m)$ time. This improves over a previous algorithm recently proposed.

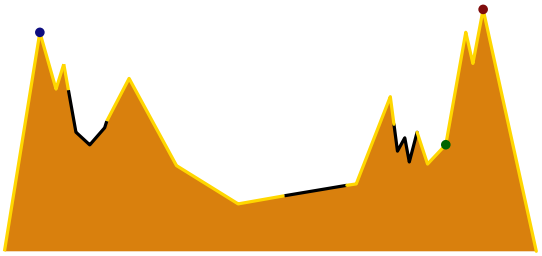

Figure 1: Visibility map for three viewpoints (disks). Visible parts shown in yellow, not visible parts black.

1 Introduction

Determining what parts of a terrain can be seen from a set of viewpoints is a natural problem that until recently has received little attention. Here we focus on 1.5D terrains: a terrain $\mathcal{T}$ is an x-monotone polygonal line in $\mathbb{R}^2$ with n vertices. In addition to $\mathcal{T}$, we are also given a set $\mathcal{P}$ of $1 \leq m \leq n$ viewpoints on the terrain surface, assumed for simplicity to be located on some of the terrain vertices. Perhaps the most basic question one can ask in this setting is: What parts of $\mathcal{T}$ are visible from *at least* one viewpoint in $\mathcal{P}$?

The particular case of $m = 1$, that is, when there is only one viewpoint, has been extensively studied. The region of the terrain visible from a single viewpoint p is commonly known as the *viewshed* of p. For 1.5D terrains, the viewshed of a point can be computed in $O(n)$ time by computing the visibility polygon from p.

For $m > 1$ viewpoints, the *visibility map* of $\mathcal{P}$ is defined as the regions of $\mathcal{T}$ that are visible from at least one viewpoint in $\mathcal{P}$. Figure 1 shows an example. A straightforward way to compute it is by computing the viewshed from each viewpoint, and then computing their union. This leads to $O(mn)$ running time.

The problem of computing the visibility map for $m > 1$ viewpoints in 1.5D (and 2.5D) terrains was studied for the first time in a very recent paper [5]. In particular, it is shown there that the visibility map of an n-vertex 1.5D terrain with $m < n$ viewpoints has $O(n)$ complexity and can be computed in $O(n \log n)$ time. Despite this being an improvement over the straightforward $O(mn)$ time algorithm, it leaves open the intriguing question of whether the visibility map can be computed faster.

In this note we answer this question affirmatively, by presenting an algorithm that computes the visibility map in $O(n+m \log m)$ time. That is, the running time is linear on the terrain complexity (n), an important improvement since in most of the applications that motivate this work the number of terrain vertices is much larger than the number of viewpoints.

Related work. To the best of our knowledge, the visibility map for multiple viewpoints was studied for the first time in [5]. A related problem of determining whether at least two viewpoints *above* a 1.5D terrain can see each other was studied in [1]. Other related work on visibility for 1.5D terrains has been mostly concerned with *placing* viewpoints such that all the terrain is visible from at least one viewpoint (see e.g. [2, 4, 6] and references therein).

Notation. We denote the x- and y-coordinates of a point $p \in \mathbb{R}^2$ by $x(p)$ and $y(p)$, respectively. The terrain $\mathcal{T}$ is specified by a sorted list of n vertices; the $m > 1$ viewpoints in $\mathcal{P}$ lie on some of the vertices of $\mathcal{T}$. We use $\mathcal{T}[a, c]$, for a, c in $\mathcal{T}$ and $x(a) < x(c)$, to denote the closed portion of the terrain between a and c, and $\mathcal{T}(a, c)$ for the open portion. Similarly, we use $\mathcal{T}[a]$ to denote the point on $\mathcal{T}$ with $x = x(a)$ (or with x-coordinate equal to a, if $a \in \mathbb{R}$).

2 The algorithm

We compute the left- and right-visibility maps separately, and then merge them. The *left-visibility map* partitions $\mathcal{T}$ into two regions: the visible and the "invisible" portions of the terrain, where *visible* means

*Department of Computing and Information Sciences, Utrecht University; m.loffler@uu.nl.

†Department of Mathematics, University of West Bohemia; saumell@kma.zcu.cz.

‡Dept. de Matemática & CIDMA, Universidade de Aveiro, and Dept. Matemàtica Aplicada II, Universitat Politècnica de Catalunya; rodrigo.silveira@ua.pt.

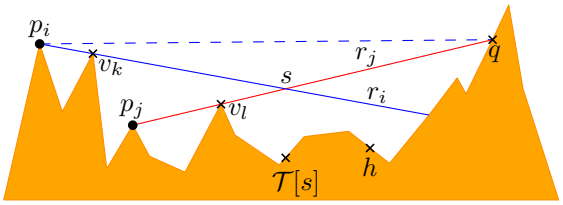

Figure 2: Situation in Corollary 2.

visible from a viewpoint *on* or *to the left* of that point of the terrain (thus in the left-visibility map, viewpoints only see *themselves* and portions of the terrain *to their right*). The right-visibility map is defined analogously. Next we present the construction of the left-visibility map (thus, *visible* stands for *left-visible*).

2.1 Preliminaries

We say that a viewpoint p_1 *dominates* another viewpoint p_2 at a given x-coordinate x_1 if for all $p \in \mathcal{T}$ with $x(p) \geq x_1$ it holds that: if p_2 sees p, then p_1 also sees p. A viewpoint that, at a given x-coordinate, is not dominated by any other viewpoint is called *non-dominated*.

The algorithm uses a couple of consequences of the so-called *order claim*:

Lemma 1 (Claim 2.1 in [2]) *Let a, b, c, and d be four points on $\mathcal{T}$ such that $x(a) < x(b) < x(c) < x(d)$. If a sees c and b sees d, then a sees d.*

Corollary 1 *Let $q \in \mathcal{T}$ be a point visible from viewpoints p_i and p_j, with p_i to the left of p_j. For any $w \in \mathcal{T}$ to the right of q, if p_i does not see w, then p_j does not see w either (i.e. p_i dominates p_j at $x(q)$).*

Given a viewpoint p_i and a vertex $v_k \in \mathcal{T}$, the ray with origin p_i and vector $\overrightarrow{p_i v_k}$ is called a *shadow ray* if: (i) p_i sees v_k; (ii) p_i does *not* see the points of $\mathcal{T}$ immediately to the right of v_k. We use $\rho(p_i, v_k)$ to denote such ray.

The next corollary is illustrated in Figure 2.

Corollary 2 *Let $r_i = \rho(p_i, v_k)$ and $r_j = \rho(p_j, v_l)$ be two shadow rays with $x(v_k) < x(v_l)$. Suppose there exists some $h \in \mathcal{T}$ such that $\mathcal{T}(v_k, h)$ lies below r_i and $\mathcal{T}(v_l, h)$ lies below r_j. Suppose that r_i and r_j cross at a point s such that $x(v_l) < x(s) < x(h)$. Then, for any point $w \in \mathcal{T}$ to the right of $\mathcal{T}[s]$, if p_i does not see w, then p_j does not see w either (i.e. p_i dominates p_j at $x(s)$).*

Proof. We start by showing that p_i is to the left of p_j. We first notice that r_j lies below r_i in the region of the plane to the left of the vertical line $x = x(s)$: Otherwise, the point v_l would lie above the ray r_i, contradicting the fact that $\mathcal{T}(v_k, h)$ lies below r_i. Additionally, we have $x(v_k) < x(p_j)$, since otherwise the point v_k would lie in $\mathcal{T}(v_l, h)$ and above the ray r_j, contradicting our hypothesis. Therefore, we conclude $x(p_i) < x(v_k) < x(p_j)$ and we are in the situation illustrated in Figure 2.

If p_j does not see any point $q \in \mathcal{T}$ to the right of $\mathcal{T}[s]$, the result follows. Otherwise, let q be the leftmost point in $\mathcal{T}$ to the right of $\mathcal{T}[s]$ visible from p_j. $\mathcal{T}(p_i, q)$ lies on or below the polygonal line $p_i s q$, which, except for its endpoints, lies below the segment $p_i q$. Hence, q is visible from p_i. We now apply Corollary 1 and conclude. □

For a fixed x-coordinate, each viewpoint can have at most one shadow ray as in Corollary 2:

Observation 1 *Given a point $h \in \mathcal{T}$ and a viewpoint p_i, there is at most one vertex v_k such that $\mathcal{T}(v_k, h)$ is below $\rho(p_i, v_k)$.*

Therefore for simplicity we will use r_i to denote such ray for viewpoint p_i, assuming it exists.

2.2 Description of the algorithm

The algorithm sweeps the terrain from left to right while maintaining some information. Most notably, we maintain a set of rays L corresponding to a (sub)set of viewpoints that are not visible at the moment[1] (possibly, $L = \emptyset$). More precisely, the set L at $x = x(h)$ (for $h \in \mathcal{T}$) is defined as

$L = \{\rho(p_i, v_k)$ s.t. $T(v_k, h)$ lies below $\rho(p_i, v_k)$ and p_i is non-dominated at $x = x(h)\}$.

The general idea is: If while sweeping through point $h \in \mathcal{T}$ we have $r_i = \rho(p_i, v_k) \in L$, then $\mathcal{T}(v_k, h)$ lies below r_i. Then if the terrain crosses r_i at some point to the right of h, it will become visible from p_i. Note that the definition of L also requires the viewpoints associated to the rays in L to be non-dominated, which allows to decrease the size of L.

We also maintain some special viewpoints. If the terrain is currently visible, we keep the leftmost visible viewpoint, which we call *primary* viewpoint and denote by p_a. If the terrain is not visible, we maintain another viewpoint, which we call *secondary* viewpoint and denote by p_b (see Figure 3). The secondary viewpoint belongs to the set $\{p_i \,|\, r_i \in L\}$, and it is the viewpoint that is *more likely* to become visible around the portion of the terrain that we are examining. More precisely, if we are sweeping through point $h \in \mathcal{T}$, then r_b is defined as the lowest ray in L at $x = x(h)$. When the terrain is not visible, we define $L' = L \backslash \{r_b\}$. Otherwise, we set $L' = L$. At any moment of the sweep, we will know the lowermost ray in L'.

Finally, we maintain a boolean $\nu \in \{0, 1\}$ that indicates whether the terrain is currently visible or not.

[1] Often in this section we use *visible* to refer to visibility from the intersection of the sweep line and the terrain.

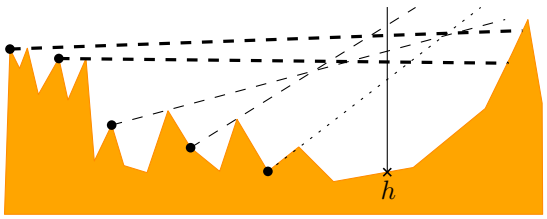

Figure 3: When sweeping point h, the secondary viewpoint is the one whose ray is dotted. L' contains only the thicker dashed rays (since the viewpoints of the other ones are dominated).

The algorithm begins at the leftmost vertex, and starts sweeping the terrain as explained below. For the sake of simplicity, in the following description we assume that we know at any time which is the lowermost ray in L' and, if $\nu = 0$, which is the secondary viewpoint. We will explain how to keep this information updated later.

The most important events are terrain vertices: the sweep stops at all of them. Other relevant events are when the secondary viewpoint changes or when the lowermost ray in L' changes. However, we can treat those easily: Suppose that we are about to process edge wv and we detect, say, that the secondary viewpoint changes at $x = \alpha$, where $x(w) < \alpha < x(v)$. Then we simply subdivide wv into $w\mathcal{T}[\alpha]$ and $T[\alpha]v$ and run two iterations of the algorithm below, each with the appropriate secondary viewpoint.

We now explain an iteration of the algorithm. Let w be the vertex preceding v in $\mathcal{T}$. We treat the interior of the edge wv and the vertex v separately.

Detecting visibility changes in the interior of the edge wv. We distinguish several cases:

(i.1) $\nu = 0$ and $L = \emptyset$. We do nothing.

(i.2) $\nu = 0$ and $L \neq \emptyset$. We check whether the ray r_b intersects the edge wv. In the affirmative, we compute the point of intersection and we set $\nu = 1$ at that point. The viewpoint that was secondary, p_b, becomes the primary viewpoint, and r_b is removed from L. We continue as in (i.3) or (i.4), depending on L being empty or not.

(i.3) $\nu = 1$ and $L = \emptyset$. We do nothing.

(i.4) $\nu = 1$ and $L \neq \emptyset$. We check whether the lowermost ray r_j of L' at $[x(w), x(v)]$ intersects the edge wv. If it does, we remove r_j from L' and find the new lowermost ray of L'. Additionally, if p_j is to the left of p_a, we set $p_a = p_j$. We continue as in (i.3) or (i.4), depending on L being empty or not.

Dealing with the vertex v. Let us first suppose that no viewpoint lies on v. We distinguish the following cases:

(ii.1) $\nu = 0$. We do nothing.

(ii.2) $\nu = 1$ and p_a continues being visible right after v. We do nothing.

(ii.3) $\nu = 1$ and p_a stops being visible right after v. We set $\nu = 0$, and we add $\rho(p_a, v)$ to L. Additionally, p_a becomes the secondary viewpoint.

Next, we treat the case where a viewpoint p_i lies on v:

(ii.4) $\nu = 0$. We set $\nu = 1$ and $p_a = p_i$. If $L \neq \emptyset$, then r_b is added to L' and the lowermost ray of L' becomes r_b. There is no longer a secondary viewpoint.

(ii.5) $\nu = 1$ and p_a continues being visible right after v. We do nothing.

(ii.6) $\nu = 1$ and p_a stops being visible right after v. We add $\rho(p_a, v)$ to L', and update the lowermost ray of L'. We set $p_a = p_i$.

It remains to explain the way we maintain the lowermost ray in L' and, if $\nu = 0$, the secondary viewpoint.

Maintaining the lowermost ray in L'. Knowing the lowermost ray in L' during the whole sweep is equivalent to maintaining the lower envelope of L'. We use a modification of Bentley-Ottmann's algorithm for line-segment intersections [3], run on the set of rays that at some point will be in L'. The algorithm essentially computes the intersections between the rays in L' as the terrain is swept. The sweep line data structure allows to retrieve the lowest ray in L' at any time.

The sweep line and the event queue are implemented using the standard data structures (i.e. a binary search tree and a priority queue).[2] Next we argue that the overall running time of the sweep is only $O(m \log m)$.

First note that, by Observation 1, at any moment of the sweep, L' contains $O(m)$ rays. Moreover, by Corollary 2, every time the sweep line goes through the intersection of two rays, the one corresponding to the viewpoint more to the right becomes dominated by the other one, so that ray will not be in L' from that moment on, and what is even more important, no ray from that viewpoint will. Thus the total number of intersections considered by the algorithm is $O(m)$.

The other types of events are insertions and deletions of rays. In total, we make at most m insertions to L': Indeed, we only add a ray in cases (ii.4) or (ii.6), and we can charge the insertion to viewpoint p_i, which is traversed at that point by the sweep line. Analogously, the number of deletions is $O(m)$ as well.

Each insertion or deletion operation in the event queue has cost $O(\log m)$, since the queue only contains intersection events about rays that are consecutive along the sweep line, and there can be at most m rays intersected by the sweep line at a given time. Since the total number of events processed is $O(m)$, it follows

[2] In our case the sweep line could be represented with a simpler structure, like a doubly-linked list, but this would not affect the overall running time.

that the total time spent on maintaining the lower envelope of L' is $O(m \log m)$.

Note that, even though we have presented this sweep line algorithm separately, it should be interleaved with the main sweep line algorithm described previously.

Maintaining the secondary viewpoint. On top of the updates caused by cases (i.2), (ii.3) and (ii.4), we do the following: Every time that there is a new secondary viewpoint or a new lowermost ray in L', we check whether r_b intersects this lowermost ray r_j. In the affirmative, we add an event at the x-coordinate of the intersection point. When this point is swept by the general algorithm, if the secondary viewpoint and the lowermost ray in L' have not changed, then p_j becomes the new secondary viewpoint. Thus, r_j is removed from L', and the lowermost ray in L' is updated. Notice that the ray corresponding to the old secondary viewpoint is not added to L' because it corresponds to a viewpoint that is now dominated.

These operations are performed every time that there is a new secondary viewpoint or a new lowermost ray in L'. If there is a new secondary viewpoint p_b caused by event (ii.3), we associate it to the vertex v such that $r_b = \rho(p_b, v)$. If there is a new secondary viewpoint because r_b intersects the lowermost ray r_j in L', we associate it to the old secondary viewpoint, which becomes dominated. This shows that the secondary viewpoint changes $O(n+m)$ times throughout the whole algorithm. On the other hand, we already know that the lowermost ray in L' changes at most $O(m)$ times. Thus, the operations described in the paragraph above are globally done in $O(n+m)$ time.

2.3 Correctness and running time

The correctness of the method follows from the fact that all changes in the terrain between visible and invisible are detected. Corollary 1 guarantees that it is enough to keep track of only the leftmost visible viewpoint, which we use in (i.4) and (ii.5). Finally, Corollary 2 shows that, whenever two rays in L cross, one of them stops being relevant for the algorithm. We use this property in the definition of L'.

We next analyze the running time. As seen before, maintaining the lowermost ray in L' at any time can be done in $O(m \log m)$ time. Notice that the number of insertions to L can be $\Theta(n)$, so if we instead maintained the lowermost ray of L, our algorithm would take $O(n \log m)$ time. For this reason we keep the secondary viewpoint separately from the remaining rays in L.

Other than that, we spend constant time per iteration. Recall that the number of iterations is bounded by the sum of: (i) the number of vertices in $\mathcal{T}$, (ii) the number of times that there is a new non-empty secondary viewpoint, (iii) the number of times that there is a new non-empty lowermost ray of L', (iv) the number of times that we are in event (i.2) and r_b intersects wv, (v) the number of times that we are in event (i.4) and the lowermost ray r_j of L' intersects wv. It is not difficult to see that this adds up to $O(n+m)$.

To conclude, we observe that the right-visibility map can be computed analogously. We finally merge the two maps in $O(n)$ time and obtain the visibility map. Note that the algorithm can be modified to output, for each visible region, a set of viewpoints that cover that region. We obtain the following theorem:

Theorem 2 *Given a 1.5D terrain $\mathcal{T}$, the visibility map of $\mathcal{P}$ can be constructed in $O(n + m \log m)$ time.*

Acknowledgments. We thank Michael Hoffmann for interesting discussions on the topic of this paper. We also thank our coauthors from [5], Ferran Hurtado, Inês Matos, Vera Sacristán, and Frank Staals, for helpful discussions related to the problem studied. M. L. is supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. M. S. is supported by the project NEXLIZ - CZ.1.07/2.3.00/30.0038, which is co-financed by the European Social Fund and the state budget of the Czech Republic. R. S. was funded by Portuguese funds through CIDMA (Center for Research and Development in Mathematics and Applications) and FCT (Fundação para a Ciência e a Tecnologia), within project PEst-OE/MAT/UI4106/2014, and by FCT grant SFRH/BPD/88455/2012. In addition, R. S. was partially supported by projects MINECO MTM2012-30951/FEDER, Gen. Cat. DGR2009SGR1040, and by ESF EUROCORES program EuroGIGA-ComPoSe IP04-MICINN project EUI-EURC-2011-4306.

References

[1] B. Ben-Moshe, O. Hall-Holt, M. J. Katz, and J. S. B. Mitchell. Computing the visibility graph of points within a polygon. In *Proc. 20th Symposium on Computational Geometry*, pages 27–35, 2004.

[2] B. Ben-Moshe, M. Katz, and J. Mitchell. A constant-factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM J. Comput.*, 36(6):1631–1647, 2007.

[3] J. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *Computers, IEEE Transactions on*, C-28(9):643–647, 1979.

[4] M. Gibson, G. Kanade, E. Krohn, and K. Varadarajan. An approximation scheme for terrain guarding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 5687 of *LNCS*, pages 140–148. Springer, 2009.

[5] F. Hurtado, M. Löffler, I. Matos, V. Sacristán, M. Saumell, R. Silveira, and F. Staals. Terrain visibility with multiple viewpoints. In *Algorithms and Computation*, volume 8283 of *LNCS*, pages 317–327. Springer Berlin Heidelberg, 2013.

[6] J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011.