





The VC-Dimension of Limited Visibility Terrains

Matt Gibson-Lopez   

The University of Texas at San Antonio, TX, USA

Zhongxiu Yang  

The University of Texas at San Antonio, TX, USA

Abstract

Visibility problems are fundamental to computational geometry, and many versions of geometric set cover where coverage is based on visibility have been considered. In most settings, points can see “infinitely far” so long as visibility is not “blocked” by some obstacle. In many applications, this may be an unreasonable assumption. In this paper, we consider a new model of visibility where no point can see any other point beyond a sight radius ρ . In particular, we consider this visibility model in the context of terrains. We show that the VC-dimension of limited visibility terrains is exactly 7. We give lower bound construction that shatters a set of 7 points, and we prove that shattering 8 points is not possible.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases VC-dimension, Terrain Guarding, Limited Visibility

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.5

Supplementary Material

Software (Source Code for Lower Bound): <https://github.com/utsa-saga/vc7proof>
archived at `swh:1:dir:33dfe57e9e3cb7a1fad1a49f6bd65a0fa1e32a67`

Software (Source Code for Upper Bound): <https://github.com/utsa-saga/vc8proof>
archived at `swh:1:dir:bffd67e203db12523de2cc9f76baa84d3b6046d0`

Funding Supported by the National Science Foundation under Grant No. 1733874.

1 Introduction

Visibility is fundamental to computational geometry, and in particular, variants of geometric set cover where coverage is determined by what a point “sees” have been widely considered in the literature. One of the most famous such applications is the *art gallery problem* where we are given a simple polygon P and we wish to place the minimum number of “guard” points G such that each point inside P is seen by some guard in G . Usually, the way that visibility is defined, is that two points p and q see each other if the line segment \overline{pq} does not contain any point outside of P . The intuition is that P is modelling some room (or art gallery), and if the line segment connecting the two points exits P , then there is a wall of the polygon that “blocks” their vision. The geometric set cover problem with respect to this definition of visibility has been considered in many different contexts, for example simple polygons [9, 15, 8, 1, 13], monotone polygons [20, 22, 12, 21, 14], rectilinear polygons [21, 22], staircase polygons [4, 23, 25], and terrains [18, 17, 2, 16, 6, 11, 19].

A *terrain* T is an x -monotone polygonal chain defined by a sequence of n vertices p_1, \dots, p_n . We let $x(p_i)$ and $y(p_i)$ denote the coordinates of p_i . The x -monotone property implies that $x(p_i) < x(p_{i+1})$ for each $i \in \{1, \dots, n-1\}$. Note that given this definition, a vertical line intersects T in at most one point. Consider three vertices of T , p_i, p_j , and p_k such that $i < j < k$. We say p_j *pierces* $\overline{p_i p_k}$ if p_j is strictly above $\overline{p_i p_k}$. Similarly, if there is some p_j that pierces $\overline{p_i p_k}$ then we say that $\overline{p_i p_k}$ is *pierced*. Generally visibility in terrains is defined in the following way: p_i and p_k see each other if and only if $\overline{p_i p_k}$ is not pierced.



© Matt Gibson-Lopez and Zhongxiu Yang;

licensed under Creative Commons License CC-BY 4.0

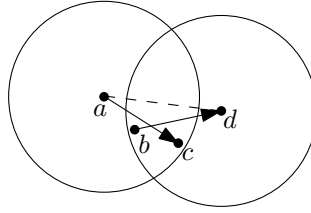
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 5; pp. 5:1–5:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** a sees c . b sees d . In an LVT, a may be too far from d to see it.

1.1 A Restricted Model of Visibility

An issue with the usual visibility model for various applications is that points are assumed to be able to see “infinitely far”. That is, a guard that is very far from p is assumed to see p just as clearly as another guard that is very close to p . Distance does not matter so long as the line segment connecting the points is not “blocked”. In many applications, this assumption is not true, and a model that considers the distance between a guard and the points it sees may be desirable. To address this undesirability, there has been some research that has considered restricted visibility models for exploring/guarding polygons or polygonal environments. Much of this past work is motivated by robotics, often times in an online setting where a robot is trying to learn about its surroundings but has restrictions on how far it can see, e.g. [3, 10, 24]. Other past work has considered restricted visibility models when computing an optimal “watchman tour” where a single guard patrols a known polygon, e.g. [26, 7].

In this paper we consider a *limited visibility* model of the terrain guarding problem, where each point has a sight radius ρ and cannot see any points that are outside of its sight radius. So to contrast with most of the past work with limited visibility models, we are assuming that we have full knowledge of the terrain we wish to guard, we can use multiple guards, and the guards are static. We call an instance of this problem a *limited visibility terrain* (LVT). An LVT can be defined by a set of n , x -monotone vertices and a sight radius ρ . We say that two points p and q see each other if and only if (1) \overline{pq} is not pierced and (2) $d(p, q) \leq \rho$, where $d(p, q)$ is the Euclidean distance between p and q . In this paper, we assume each point has the same sight radius, although it would be interesting to also consider the scenario where each point has its own radius. For any point p , we denote $disc(p)$ to be a disc of radius ρ centered at p . Then clearly p cannot see any points that are not inside $disc(p)$ in this model.

1.2 Order Claim Generalization

One of the key properties of (regular) terrains is the order claim.

▷ **Claim 1.** Let a, b, c, d be four points on a regular terrain such that $x(a) < x(b) < x(c) < x(d)$. If (1) a sees c and (2) b sees d , then a sees d .

The intuition here is pretty simple. Since a sees c , it must be that \overline{ac} is not pierced (so b is “below” \overline{ac}). Similarly \overline{bd} is not pierced (so c is below \overline{bd}). This implies that \overline{ad} is “above” \overline{ac} and \overline{bd} , and therefore \overline{ad} is not pierced (implying that a and d see each other).

For LVTs the order claim does not hold if $a \notin disc(d)$. See Figure 1 for an illustration. We can however apply a generalization of the order claim.

▷ **Claim 2.** Let a, b, c, d be four points on an LVT such that $x(a) < x(b) < x(c) < x(d)$. If (1) \overline{ac} is not pierced and (2) \overline{bd} is not pierced, then \overline{ad} is not pierced.

1.3 VC-Dimension

An interesting measure of the complexity of a set system is the notion of VC-dimension. To define this, we say that a finite set of points G in T is *shattered* if for every subset of $G' \subseteq G$ there exists some point $v \in T$ such that v sees every point in G' and does not see any point in $G \setminus G'$. In this context, we call v a *viewpoint*. The VC-dimension is the largest d such that there exists some LVT T and point set G of size d that can be shattered.

Brönnimann and Goodrich give a polynomial-time $O(\log OPT)$ -approximation algorithm for any set system with constant VC-dimension [5] where OPT is the size of an optimal cover. For regular terrains, King showed that the VC-dimension is 4 [17]. This result relies heavily on the order claim, which as we mentioned above does not hold in the limited visibility model. One would certainly expect the VC-dimension to increase in this model.

1.4 Our Results

We prove the following theorem.

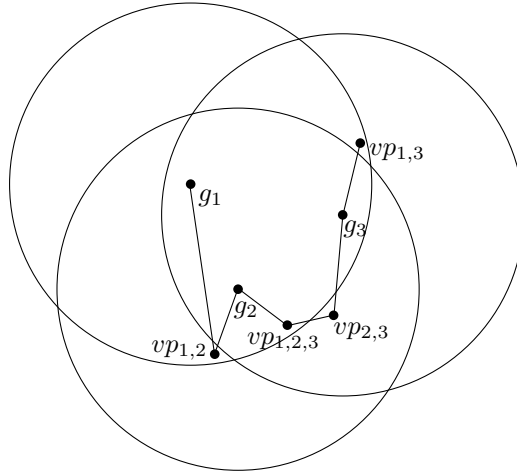
► **Theorem 3.** *The VC-Dimension of limited visibility terrains is 7.*

We show that for any set of 8 points on a LVT, it is not possible to shatter the set, and we show that it is possible to shatter a set of 7 points (thereby proving that our upper bound analysis is tight). We show several key structural lemmas about visibility in a LVT, and then use a computer program to show that there is no permutation of the 8 points and the 2^8 needed viewpoints such that there is a LVT with the points in left-to-right order according to the permutation that satisfies all of the visibility requirements. As is often the case with VC-dimension proofs, a direct proof often involves a tedious case analysis, so we use a computer program that is much easier to verify is correct rather than having to analyze each case “manually”. Our proof of Theorem 3 then implies that the algorithm of Brönnimann and Goodrich [5] is a $O(\log OPT)$ -approximation algorithm for guarding the vertices of a LVT.

2 Upper Bound

In this section, we show that the VC-dimension of LVTs is at most 7. We will do this by showing that any set of 8 points cannot be shattered. Suppose that it is possible to shatter 8 points, that is there exists a LVT that shatters a set $G = \{g_1, \dots, g_8\}$ which are indexed according to increasing x-coordinate without loss of generality. Motivated by the connection to geometric set cover, we sometimes call the points of G *guards*. For any subset $S \subseteq \{1, \dots, 8\}$, let vp_S denote a point that should see g_i for each $i \in S$ and does not see g_j for any $j \notin S$. For example, $vp_{1,6,8}$ should see g_1, g_6 , and g_8 and should not see any other point of G . Let V be the set of all such viewpoints. Clearly $|V| = 2^8 = 256$.

We begin with a high-level overview of our proof strategy for showing that G cannot be shattered. Suppose the contrary, and let T denote a LVT that does shatter G . We say the *ordering* of T is the permutation of $G \cup V$ when sorting $G \cup V$ by their x-coordinates (note T may have vertices that do not correspond with G or V). We say any subsequence of the ordering is a *partial ordering* of T . The intuition is that an ordering describes the left-to-right order of the points of V and G with respect to T , and a partial ordering describes the left-to-right ordering of *some* of the points of V and G . See Figure 2 for an illustration.



■ **Figure 2** The ordering of T is $(g_1, vp_{1,2}, g_2, vp_{1,2,3}, vp_{2,3}, g_3, vp_{1,3})$. Partial orderings of T include $(g_1, g_2, vp_{2,3}, g_3)$, $(vp_{1,2}, vp_{1,2,3}, g_3, vp_{1,3})$, etc.

Now we view the problem in the “opposite direction”; instead of starting with a LVT and considering its order, we start with an order and consider whether there is a LVT that has this order. We call any permutation O of $V \cup G$ a *candidate ordering* (CO) of $V \cup G$. We say O is *realizable* if there is a LVT T that shatters G such that O is the ordering of T . For every subset $V' \subseteq V$, we call a permutation of $V' \cup G$ a *candidate partial ordering* (CPO), and we say it is realizable if it is a partial ordering of any LVT T that shatters G .

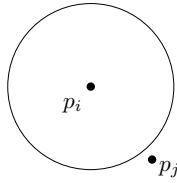
We obtain our result by showing that there is no realizable CPO. We do this by first proving a set of structural lemmas. Given a CPO O , the lemmas will imply some properties that must be satisfied by *any* realization of O . For example, we might be able to argue that any realization of O must satisfy $y(g_i) < y(g_j)$, or we might be able to argue that a point $v \in V$ that is not supposed to see g_i must be outside the disc of g_i or $\overline{vg_i}$ must be pierced in any realization. If the lemmas contradict each other then we can determine that O is not realizable.

The proofs of VC-dimension upper bounds are often tedious case analyses. We instead use a computer program that helps to automate this case analysis. Our program, given a CPO O , determines if our lemmas can be used to show that O is not realizable. The CPOs we give as inputs to the program are all of the CPOs based off G and four points of V . Each of the CPOs can be analyzed in parallel, and therefore picking four points of V provides a nice tradeoff between the number of CPOs that must be considered and the amount of time it takes to evaluate a single CPO (the time to evaluate all cases on a 3.7 GHz CPU with 6 cores is less than 24 hours).

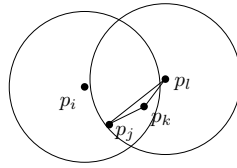
2.1 Structural Lemmas

We now give a set of structural lemmas that will help us argue when a CPO is not realizable. We use the following definitions. Suppose that p_i and p_j are two points on an LVT such that $d(p_i, p_j) \geq \rho$ and $|x(p_i) - x(p_j)| < \rho$ (note that this implies that $y(p_i) \neq y(p_j)$). Then if $y(p_i) > y(p_j)$ we say that p_i is *high outside disc*(p_j), and similarly we say that p_j is *low outside disc*(p_i). See Figure 3.

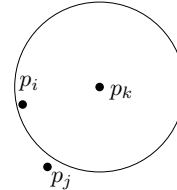
► **Lemma 4.** *Let p_i, p_j, p_k , and p_l be four points on a LVT such that $x(p_i) < x(p_j) < x(p_k) < x(p_l)$. If (1) p_i sees p_k and (2) p_j sees p_l , then $d(p_j, p_k) \leq \rho$.*



■ **Figure 3** p_j is low outside of $disc(p_i)$.



■ **Figure 4** $d(p_j, p_k) \leq \rho$.



■ **Figure 5** $y(p_j) < y(p_i)$ and $y(p_j) < y(p_k)$.

Proof. Without loss of generality, assume that $y(p_j) \leq y(p_k)$. Then it must be that $y(p_k) \leq y(p_l)$, or else p_k would pierce $\overline{p_j p_l}$. Consider the triangle $\triangle p_j p_k p_l$. We have $\angle p_j p_k p_l > \frac{\pi}{2}$, thus $\overline{p_j p_l}$ is the longest side of the triangle. Then we have $d(p_j, p_k) < d(p_j, p_l) \leq \rho$. See Figure 4 for illustration. ◀

► **Lemma 5.** Let p_i, p_j , and p_k be three points on a LVT such that $x(p_i) < x(p_j) < x(p_k)$. If (1) p_k sees p_i but does not see p_j and (2) $\overline{p_j p_k}$ is not pierced, then p_j is low outside $disc(p_k)$. Moreover, we also have $y(p_j) < y(p_i)$.

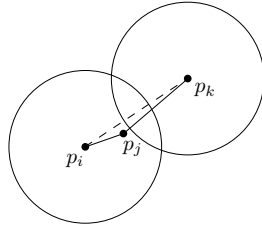
Proof. See Figure 5 for illustration. Since $\overline{p_j p_k}$ is not pierced and they do not see each other, then clearly $d(p_j, p_k) > \rho$. Since $p_i \in disc(p_k)$, we must have $x(p_k) - x(p_i) \leq \rho$, and since $x(p_j)$ is “between” $x(p_i)$ and $x(p_k)$ it therefore must be that $x(p_k) - x(p_j) < \rho$. So it remains to show that $y(p_j) < y(p_k)$ and $y(p_j) < y(p_i)$. For the sake of contradiction, suppose that $y(p_j) \geq y(p_i)$, clearly $y(p_k) > y(p_j)$, otherwise p_j will pierce $\overline{p_i p_k}$. Consider the $\triangle p_i p_j p_k$. We have $\angle p_i p_j p_k$ is greater than $\frac{\pi}{2}$, see Figure 6 for illustration. Therefore $d(p_i, p_k) > d(p_j, p_k) > \rho$, contradicting that p_i sees p_k . Thus the initial assumption $y(p_j) \geq y(p_i)$ is wrong. We can derive a contradiction for $y(p_j) \geq y(p_k)$ similarly. ◀

► **Lemma 6.** Let p_i, p_j , and p_k be three points on a LVT such that $x(p_i) < x(p_j) < x(p_k)$. If (1) $y(p_i) < y(p_j)$ and $d(p_i, p_j) > \rho$, and (2) $\overline{p_i p_k}$ is not pierced, then $y(p_j) < y(p_k)$ and $d(p_i, p_k) > \rho$.

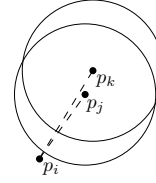
Proof. Clearly we must have $y(p_j) < y(p_k)$ or else p_j would pierce $\overline{p_i p_k}$. See Figure 7 for illustration. In $\triangle p_i p_j p_k$, clearly the longest edge is $\overline{p_i p_k}$, thus $d(p_i, p_k) > d(p_i, p_j) > \rho$. ◀

► **Lemma 7.** Let p_i and p_k be two points on a LVT that that do not see each other, but $d(p_i, p_k) \leq \rho$. Then there must be some point $p_j \in (p_i, p_k)$ that pierces $\overline{p_i p_k}$. If there is some point p_l such that $d(p_i, p_l) \leq \rho$ and $d(p_k, p_l) \leq \rho$, then either $d(p_j, p_l) \leq \rho$ or p_j is high outside $disc(p_l)$.

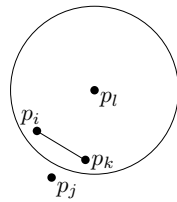
Proof. It must be that $x(p_l) - x(p_i) \leq \rho$, and since p_j is “between” p_i and p_k then we have $x(p_l) - x(p_j) < \rho$. We then complete the lemma by showing that if p_j is such that $d(p_j, p_l) > \rho$ and $y(p_j) < y(p_l)$ then it cannot pierce $\overline{p_i p_k}$. In this case, p_j must be low outside $disc(p_l)$. By convexity, we have that $\overline{p_i p_k}$ is contained inside of $disc(p_l)$, and therefore it cannot be that p_j pierces $\overline{p_i p_k}$, see Figure 8 for illustration. ◀



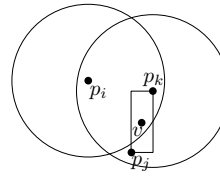
■ **Figure 6** Contradiction with p_i seeing p_k .



■ **Figure 7** $y(p_i) < y(p_k)$.



■ **Figure 8** Contradiction with $d(p_j, p_l) > \rho$ and $y(p_j) < y(p_l)$.



■ **Figure 9** Contradiction with $d(v, p_k) \leq \rho$.

► **Lemma 8.** *Let p_i, p_j , and p_k be three points on an LVT such that $x(p_i) < x(p_j) < x(p_k)$ such that p_j is low outside $\text{disc}(p_i)$ and $d(p_j, p_k) \leq \rho$. Then there is no point $v : x(p_j) < x(v) < x(p_k)$ such that v is low outside $\text{disc}(p_k)$ and $d(p_i, v) \leq \rho$.*

Proof. If the lemma is incorrect, then there would be a v that satisfies the following three properties: (1) $y(v) < y(p_k)$, (2) $d(v, p_k) > \rho$, and (3) $d(p_i, v) \leq \rho$. We will prove the lemma by showing that any point that satisfies conditions (1) and (3) cannot also satisfy condition (2). To that end, let v be any point satisfying $y(v) < y(p_k)$ and $d(p_i, v) \leq \rho$. We will show that it must be such that $d(v, p_k) \leq \rho$. See Figure 9 for illustration. Given $x(p_i) < x(p_j) < x(v)$, $d(p_i, p_j) > \rho$, $d(p_i, v) \leq \rho$, from Lemma 5 clearly $y(v) > y(p_j)$. Since $y(v) < y(p_k)$, then we have $x(p_j) < x(v) < x(p_k)$ and $y(p_j) < y(v) < y(p_k)$. This implies that v is inside of the axis-parallel rectangle with p_j as lower-left corner and p_k as upper-right corner. If a and b are any two points in this rectangle, then we have $d(a, b) \leq d(p_j, p_k) \leq \rho$, and therefore it must be that $d(v, p_k) \leq \rho$. ◀

2.2 Procedure

We use a computer program to show that there does not exist a realizable CO of $G \cup V$. Our strategy is to consider a starting set of viewpoints $V_{start} := \{vp_{1,3,5,7,8}, vp_{1,2,4,6,8}, vp_{1,3,6,8}, vp_{1,2,4,5,7,8}\}$, and to show that every CPO of $G \cup V_{start}$ is not realizable. This implies there is no realizable CO of $G \cup V$. If not, then consider the subsequence of a realizable order when considering the vertices in $G \cup V_{start}$. This would be a realizable CPO of $G \cup V_{start}$.

Suppose O is some CPO of $G \cup V_{start}$ that we are considering. It may be that our lemmas can “directly” show that O is not realizable, that is we can apply the lemmas to the points in O to derive a contradiction. In fact, this is true for roughly half of the CPOs of $G \cup V_{start}$ (which is why we pick this set as a starting point). If we cannot directly show O is not

realizable, we consider *extensions* of O . That is, we consider a viewpoint v that is not in O , and we consider adding v to O in one of the $|O| + 1$ “gaps” in O . Intuitively, we add one more viewpoint to O to obtain a new CPO O' while ensuring that the order of the points in O is the same in O' . Then we say that O' is an extension of O . Using this terminology, note that O is realizable if and only if there is at least one realizable extension of O . Similarly, if we are considering a CPO O of $G \cup V'$ and there is a viewpoint $v \in V \setminus V'$ such that our lemmas imply a contradiction for *every* extension of O that includes v , then we can in turn say that O is not realizable. When considering viewpoints for extensions, we restrict ourselves to the set V_{test} which is the set of all 96 viewpoints such satisfies either (1) they see both g_1 and g_8 , or (2) see g_2 and g_7 and exactly one of g_1 and g_8 . The intuition is that V_{test} consist of the more “difficult” viewpoints to find a spot in an order, so by only considering extensions from V_{test} we can focus the search to “difficult” viewpoints only, speeding up our program. And indeed it is sufficient to consider only these viewpoints given that our program verifies that no CPO of $G \cup V_{start}$ is realizable, even if we only needed to extend to $G \cup V_{test}$.

Our program consists of two main functions: *isNotRealizable()* (Algorithm 1 and 2 in the paper due to the length of the algorithm) and *candidateOrderingIsExtendable()* (Algorithm 3 in the paper). *isNotRealizable()* takes a CPO as input and returns true if our lemmas can directly show that the order is not realizable, and returns false otherwise. *candidateOrderingIsExtendable()* takes a CPO as input and returns true if there is an extension such that the lemmas cannot directly show it is not realizable, and otherwise returns false. In this paper, we give color-coded pseudocode that explains what our program does. The actual C++ source code that we use as well as a color-coded rich text version of the source code is provided at <https://github.com/utsa-saga/vc8proof> (the colors are used to help with the readability of the code, pairing the “sections” of the source code with the “sections” of the pseudocode from the paper).

Given 4 fixed starting viewpoints V_{start} and guard set $G = \{g_1, g_2, \dots, g_8\}$, we consider all 11,880 CPOs of $G \cup V_{start}$. Our program handles each of these 11,880 orderings independently (cases can be tested simultaneously in parallel). Let O_1 denote one such ordering. We pass O_1 to *candidateOrderingIsExtendable()*. This function calls *isNotRealizable()* on O_1 . If we determine that it is not realizable, then *candidateOrderingIsExtendable()* returns false immediately. If we cannot determine that O_1 is not realizable, then we add a new viewpoint from V_{test} to O_1 to obtain a new candidate ordering and repeat until we find a candidate ordering of $G \cup V_{test}$ for which *isNotRealizable()* returns false (indicating that it might be realizable) or determine that every candidate ordering of $G \cup V_{test}$ extending from O_1 is not realizable. In our actual program, we use some heuristics for computing a “good” ordering of the points in $V_{test} \setminus O_i$ that are not shown in *candidateOrderingIsExtendable()*.

It is not too difficult to see that *candidateOrderingIsExtendable()* is correct given that *isNotRealizable()* is correct. We will formally show that *isNotRealizable()* is correct. That is, we show that if *isNotRealizable()* returns true for some candidate ordering O_i , then a contradiction about any realization can be derived from our lemmas. To help show the correctness of this function, we define some notation. We use three main variables to detect lemma contradictions: `cannotBlockWithTerrain[{u, v}]`, `tooFarAway[{u, v}]`, and `higherPoint[{u, v}]`.

The first variable is set to true if it must be that every realization must satisfy that \overline{uv} is not pierced (following from the order claim generalization that we presented in Section 1.2. We consider this variable for all pairs $u, v \in O$. The second variable is set to be true when every realization (according to our lemmas) must satisfy that $d(u, v) > \rho$, it is set to false if $d(u, v) \leq \rho$ in every realization, and it is set to unknown if our lemmas do not imply this one

way or the other. We consider this variable for u, v pairs such that at least one of u and v is in G . As for the third variable, it will be set to either u , v , or unknown depending on if we can determine which point must have the larger y-coordinate in any realization according to our lemmas. This again is defined for all pairs u, v pairs such that at least one of u and v is in G .

In order to determine the values of variables 2 and 3, we define some extra notation and variables that are used as “helpers”. Let O be any candidate ordering of $G \cup V'$, where V' is any subset of V . For ease of description, we add to O a “dummy” point $vp_{-\infty}$ to the left of every point of O and a “dummy” point vp_{∞} to the right of every point of O . Since we have a fixed ordering of the points, we can determine for any two points of $u, v \in O$ whether or not u and v are can be pierced by applying the order claim generalization we presented in Section 1.2. For any guard $g \in G$, let $L(g)$ denote all viewpoints to the left of g in O , and similarly let $R(g)$ denote all viewpoints to the right of g in O . Let $leftMostPointGuardSees(g)$ denote the leftmost point of $L(g)$ that sees g . If there is no such point, then we define it to be g . Similarly we let $rightMostPointGuardSees(g)$ denote the rightmost point in $R(g)$ that sees g , and if no such point exists we set it to be g . Let $closestHighOutsideLeft(g)$ denote the rightmost point $v \in L(G)$ such that g does not see v , $cannotBlockWithTerrain[\{g, v\}]$ is true, and v sees some $g' \in G$ right of g (our lemmas imply this view point must be high outside $disc(g)$). If no such point exists, we let $closestHighOutsideLeft(g)$ be $vp_{-\infty}$. We similarly define $closestHighOutsideRight(g)$ to be the leftmost point $v \in R(g)$ such that g does not see v , $cannotBlockWithTerrain[\{g, v\}]$ is true, and v sees some $g' \in G$ left of g . If no such point exists, we define $closestHighOutsideRight(g)$ to be vp_{∞} . For any two points $u, v \in O$, we say $u < v$ if u is to the left of v in O .

► **Lemma 9.** *If our function $isNotRealizable(O)$ returns true, then O is not realizable.*

Proof. We will show a contradiction to realization for each line of $isNotRealizable()$ that returns true. To this end, we will also show that the assignments for our variables $cannotBlockWithTerrain$, $tooFarAway$, and $highestPoint$ are in fact correct.

Gold Block. In the gold block (Line 1), we set the values for the variable $cannotBlockWithTerrain$. If for points a and d we have $cannotBlockWithTerrain[\{a, d\}]$ set to true, then it is true that in any realization of O that \overline{ad} is not pierced as either they see each other or there are points b and c such that \overline{ac} is not pierced and \overline{ad} is not pierced (second Claim from Section 1.2). If $cannotBlockWithTerrain[\{a, d\}]$ is false then we make no assumptions about whether \overline{ad} is pierced.

Green Block. Now consider the green block (Lines 2–3), and in particular suppose we return true in line 3. Without loss of generality assume that it returned true because $leftMostPointGuardSees(g) < closestHighOutsideLeft(g)$. Let p denote $leftMostPointGuardSees(g)$, and let q denote $closestHighOutsideLeft(g)$. Then g does not see q and \overline{qg} cannot be pierced which implies that in any realization we must have $d(q, g) > \rho$. But we can apply Lemma 4 with $p_i, p_j, p_k,$ and p_l as p, q, g, g' respectively, where g' is a guard that q sees to the right of g (which exists by the definition of $closestHighOutsideLeft(g)$). Then in any realization, $d(q, g) \leq \rho$ by Lemma 4, a contradiction.

We also remark that $closestHighOutsideRight(g)$ must indeed be high outside $disc(g)$ in any realization. This follows from Lemma 5 by letting p_k be $closestHighOutsideRight(g)$, p_j be g , and p_i be any guard that $closestHighOutsideRight(g)$ sees to the left of g . Similarly, $closestHighOutsideLeft(g)$ must be high outside of $disc(g)$.

Blue Block. Now consider the blue block of code (Lines 4–23) where we set the `tooFarAway` and `higherPoint` variables for each pair of guards. We claim that if `tooFarAway[{gi, gj}]` is set to true, then the lemmas imply that any realization must have $d(g_i, g_j) > \rho$, and if it is set to false then every realization must have $d(g_i, g_j) \leq \rho$. Similarly we claim that if we set `higherPoint[{gi, gj}]` to be g_i , then any realization must have $y(g_i) > y(g_j)$ and if it is set to be g_j then any realization must have $y(g_i) < y(g_j)$. We initially set both variables to unknown, and then we change them if we detect a reason to change them.

Suppose we set `tooFarAway[{gi, gj}]` to be false in Line 7. Then let p be `leftMostPointGuardSees(gj)` and let q be `rightMostPointGuardSees(gi)`. Then from the if-statement, we have that $p < g_i < g_j < q$ and applying Lemma 4 we have that $d(g_i, g_j) \leq \rho$ in any realization, therefore our variable setting is correct.

Now suppose that we set `higherPoint[{gi, gj}]` to be g_i in Line 9. If `closestHighOutsideRight(gj)` and `rightMostPointGuardSees(gi)` are the same point, then we have that $y(g_i) > y(g_j)$ in any realization by Lemma 5. If they are not the same point, then we have $g_i < g_j < p < q$ where p is `closestHighOutsideRight(gj)` and q is `rightMostPointGuardSees(gi)`. If we had $y(g_i) \leq y(g_j)$ then it would have to be that $d(g_i, p) > \rho$ and $y(g_i) < y(p)$ and therefore we can apply Lemma 6 to g_i , p , and q to get that $d(g_i, q) > \rho$, a contradiction. Therefore it must be that $y(g_i) > y(g_j)$ in any realization. A symmetric argument holds for Line 13.

Now suppose that we set `higherPoint[{gi, gj}]` to be g_j in Line 17 (and therefore also set `tooFarAway[{gi, gj}]` to be true in the next line). Then from the if-statement, we can apply Lemma 6 to g_i , `closestHighOutsideRight(gi)`, and g_j to see that both assignments must be true in any realization. The argument is symmetric in Line 22.

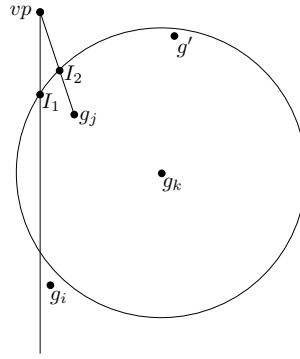
Note that each of the “return true” statements in this block occur if we get contradictory statements about which of g_i and g_j has the larger y-coordinate, and therefore if we return true in this block then O is indeed not realizable.

Light Green Block. Now consider the light green block (Lines 24–26). If we return true in line 26, then we have that $\overline{g_i g_j}$ cannot be pierced in any realization. But at the same time we would need a g_k between g_i and g_j such that $y(g_k) > y(g_i)$ and $y(g_k) > y(g_j)$. Clearly the latter implies that g_k pierces $\overline{g_i g_j}$, and therefore O cannot be realized.

Red Block. Now consider the red block of code (Lines 27–41). In this block we set the `tooFarAway` and `higherPoint` variables for a guard g and a viewpoint p . We will argue that if we set these variables to something other than unknown then any realization must satisfy that assignment. If we set `tooFarAway[{g, p}]` to be false in Line 30 then its either because g sees p (in which case they obviously cannot be too far away) or it’s because `leftMostPointGuardSees(g) < p < g < g'` where g' is a guard that p sees. In this case we can apply Lemma 4 to see that it must be that $d(p, g) \leq \rho$ in any realization.

Now suppose we enter the else statement starting at Line 34. In this case we have that g does not see p and we have `cannotBlockWithTerrain[{g, p}]` is true. Therefore clearly the reason why g does not see p is because $d(g, p) > \rho$, and therefore our assignment of `tooFarAway[{g, p}]` to be true is correct. Then further suppose we set `higherPoint[{g, p}]` to be g in Line 36. Then we can apply to Lemma 5 to `leftMostPointGuardSees(g)`, p , and g to see that indeed it must be $y(p) < y(g)$. The argument for Line 38 where we set `higherPoint[{g, p}]` to be p follows symmetrically.

Yellow Block. Now consider the yellow block (Lines 42–51). Consider any viewpoint vp that satisfies the for-loop condition in Line 43. Then we have that vp does not see g_j , but $d(vp, g_j) \leq \rho$. Therefore any realization must have some point that pierces $\overline{vp g_j}$. Moreover



■ **Figure 10** $x(g') > x(g_k)$ and vp sees g' .

we have $\text{cannotBlockWithTerrain}[\{g_i, g_j\}]$ is true from Line 42. Then maybe g_i is a point that pierces $\overline{vp g_j}$, but if it does not pierce it then certainly no point between g_i and g_j can pierce $\overline{vp g_j}$ (as it would also pierce $\overline{g_i g_j}$). This means that if g_i does not pierce $\overline{vp g_j}$, then we would need to find a point between vp and g_i (possibly a point not in O) to pierce $\overline{vp g_j}$. The yellow block first considers if it is possible for g_i to pierce, and if it cannot, then we consider the possibility of a “blocker” between vp and g_i .

First consider the if-statement in Line 44. If this condition is true, then we have that there is some guard g_k such that vp and g_j are inside of $\text{disc}(g_k)$ but g_i is low outside $\text{disc}(g_k)$. Lemma 7 implies that any point that pierces $\overline{vp g_j}$ cannot be low outside $\text{disc}(g_k)$, and therefore g_i cannot pierce $\overline{vp g_j}$.

Next consider the if-statement in Line 46. If this condition is true, then we have that there is some guard g_k to the right of g_j such that vp is high outside $\text{disc}(g_k)$, g_i is low outside $\text{disc}(g_k)$, and g_j is inside $\text{disc}(g_k)$. Since vp is high outside $\text{disc}(g_k)$, we have that a ray shot straight down from vp will first intersect $\text{disc}(g_k)$ at a point I_1 such that $y(I_1) > y(g_k)$ (recall that this is true since vp must see a guard to the right of g_k in order for $\text{higherPoint}[\{vp, g_k\}]$ to be vp). Now consider walking along $\overline{vp g_j}$ starting at g_j and walking towards vp . Let I_2 denote the point that this walk exits $\text{disc}(g_k)$. Then we have that $y(I_2) > y(I_1)$. See Figure 10. Therefore any point that is low outside $\text{disc}(g_k)$ cannot pierce $\overline{vp g_j}$, and therefore g_i cannot pierce $\overline{vp g_j}$.

So now suppose that $g_i \text{CanBeBlocker}$ was set to false by one of the two cases above. Then indeed g_i cannot pierce $\overline{vp g_j}$ and we now need to consider a blocker strictly between vp and g_i . There are two potential issues this might cause. The first is that if g_i does not pierce $\overline{vp g_j}$ but some point b between vp and g_i does pierce $\overline{vp g_j}$, then it must be that b also pierces $\overline{vp, g_i}$. So if we have $\text{cannotBlockWithTerrain}[\{vp, g_i\}]$ set to true, then we cannot use such a blocker b . But moreover, using a blocker b in this range may cause us to have to flip $\text{cannotBlockWithTerrain}[\{vp_2 g_j\}]$ from false to true if we use such a blocker b . Indeed, for any point vp_2 to the left of vp , if we have $\text{cannotBlockWithTerrain}[\{vp_2, g_i\}]$ is true, then the use of this blocker b will force us to have $\text{cannotBlockWithTerrain}[\{vp_2, g_j\}]$ to be true by the second claim in Section 1.2. If we previously determined that vp_2 cannot be too far away from g_j , then this would imply that vp_2 sees g_j , a contradiction. Therefore in this case we would not be able to use a new blocker b .

So if we determine that g_i cannot be a blocker and a new point between vp and g_i cannot be the blocker, then it is not possible to pierce $\overline{vp g_j}$ without changing other visibilities and therefore O cannot be realized, and the algorithm returns true accordingly.

Purple Block. Now consider the purple block (Lines 52–54). If there are guards g_i and g_j and viewpoints vp_1 and vp_2 that satisfy the if-statement condition then we have vp_2 sees g_j and is low outside $disc(g_i)$, and we have vp_1 sees g_i and is low outside $disc(g_j)$. The left-to-right order of $g_i < vp_2 < vp_1 < g_j$ then contradicts Lemma 8, and therefore O is not realizable and we return true accordingly. ◀

3 Lower Bound

In this section, we show that the VC-Dimension of LVTs is at least 7. See Figure 12 for the coordinates of the vertices of our LVT where a set of 7 vertices is shattered. We remark that the main value of this construction is that it shows that our upper bound proof we give in Section 2 is tight. We do not claim that this LVT is particularly interesting outside of the fact that it proves that the upper bound cannot be improved.

Here, we assume that the radius ρ that each vertex can see is 1. The vertices that are shattered are the vertices $G = \{g_1, \dots, g_7\}$. The viewpoints are labeled vp where the subscript denotes which vertices of G the viewpoint sees. Some of the vertices are neither a viewpoint nor a vertex in G and are labeled “-” (their role is simply to block a viewpoint from seeing a vertex in G). The table does not contain a viewpoint for the empty set or for the subsets of size 1. It is trivial to add these points: the empty set point can be placed to the right of all points a distance greater than 1 to each vertex in G , and a viewpoint that should see only one vertex g_i can be placed in a steep “canyon” just to the left of g_i so that every other g_j is blocked from seeing it.

To formally verify that our coordinates are correct, we wrote a computer program that ensures each viewpoint sees exactly which guards they are supposed to see. Our program to verify the problem is available at <https://github.com/utsa-saga/vc7proof>.

The aspect ratio of our construction is very large which makes it difficult to produce a static figure that is helpful in visualizing the construction. Nonetheless, see Figure 11 for a static figure of the LVT. Note that in this figure, we use letters A through G to represent the guards instead of $g_1 - g_7$ (e.g., B represents g_2 and BEF represents $vp_{2,5,6}$). See Figure 11 for the overall the boundary of the LVT, where the viewpoints are listed in blue (many of the labels are not visible because the points are so close together) and the guards in G are red. There is a dynamic version of the figure that allows zooming in-and-out here: <https://www.geogebra.org/calculator/ybvdrjag>. Finally we have dynamic figures for each guard. They show the disc of the guard so it is possible to see which portions of the LVT are too far away from certain guards. In these figures, green line segments indicate that the guard and viewpoint see each other (their line segment is not pierced, and they are close enough to see each other); red line segments indicate their line segment is pierced; purple viewpoints indicate the viewpoints that are outside of the disc of the guard.

- g_1 : <https://www.geogebra.org/calculator/vcjuryvhmm>
- g_2 : <https://www.geogebra.org/calculator/hjnqeuyqy>
- g_3 : <https://www.geogebra.org/calculator/nktasnvc>
- g_4 : <https://www.geogebra.org/calculator/pgb5tzj2>
- g_5 : <https://www.geogebra.org/calculator/bptpyugq>
- g_6 : <https://www.geogebra.org/calculator/cx8khxkc>
- g_7 : <https://www.geogebra.org/calculator/t83pypcq>

■ **Algorithm 1** boolean `isNotRealizable(O)` – Part 1 of 2.

```

1: For each pair of points  $u, v$ , set cannotBlockWithTerrain[ $\{u, v\}$ ] to true if  $u$  sees  $v$  or the
   order claim generalization implies that  $\overline{uv}$  cannot be pierced, otherwise set it to be false.
2: if there is a  $g \in G$  such that
    $leftMostPointGuardSees(g) < closestHighOutsideLeft(g)$  or
    $closestHighOutsideRight(g) < rightMostPointGuardSees(g)$  then
3:   return true
4: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
5:   tooFarAway[ $\{g_i, g_j\}$ ] = unknown, higherPoint[ $\{g_i, g_j\}$ ] = unknown
6:   if  $leftMostPointGuardSees(g_j) < g_i$  and
    $g_j < rightMostPointGuardSees(g_i)$  then
7:     tooFarAway[ $\{g_i, g_j\}$ ] = false
8:     if  $closestHighOutsideRight(g_j) \leq rightMostPointGuardSees(g_i)$  then
9:       higherPoint[ $\{g_i, g_j\}$ ] =  $g_i$ 
10:    if  $leftMostPointGuardSees(g_j) \leq closestHighOutsideLeft(g_i)$  then
11:      if higherPoint[ $\{g_i, g_j\}$ ] is  $g_i$  then
12:        return true
13:      higherPoint[ $\{g_i, g_j\}$ ] =  $g_j$ 
14:      if  $closestHighOutsideRight(g_i) < g_j$  and
   cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true then
15:        if higherPoint[ $\{g_i, g_j\}$ ] is  $g_i$  then
16:          return true
17:        higherPoint[ $\{g_i, g_j\}$ ] =  $g_j$ 
18:        tooFarAway[ $\{g_i, g_j\}$ ] = true
19:      if  $g_i < closestHighOutsideLeft(g_j)$  and
   cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true then
20:        if higherPoint[ $\{g_i, g_j\}$ ] is  $g_j$  then
21:          return true
22:        higherPoint[ $\{g_i, g_j\}$ ] =  $g_i$ 
23:        tooFarAway[ $\{g_i, g_j\}$ ] = true
24: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
25:   if cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true and there is a  $k \in (i, j)$  such that
   higherPoint[ $\{g_i, g_k\}$ ] and higherPoint[ $\{g_k, g_j\}$ ] are both  $g_k$  then
26:     return true
27: for all  $g \in G$  do
28:   for all viewpoints  $p \in L(g)$  do
29:     if  $g$  sees  $p$  or ( $leftMostPointGuardSees(g) < p$  and
    $p$  sees a guard right of  $g$ ) then
30:       tooFarAway[ $\{g, p\}$ ] = false, higherPoint[ $\{g, p\}$ ] = unknown
31:     else if cannotBlockWithTerrain[ $\{g, p\}$ ] is false then
32:       tooFarAway[ $\{g, p\}$ ] = unknown, higherPoint[ $\{g, p\}$ ] = unknown
33:     else
34:       tooFarAway[ $\{g, p\}$ ] = true
35:       if  $leftMostPointGuardSees(g) < p$  then
36:         higherPoint[ $\{g, p\}$ ] =  $g$ 
37:       else if  $p$  sees a guard to the right of  $g$  then
38:         higherPoint[ $\{g, p\}$ ] =  $p$ 
39:       else
40:         higherPoint[ $\{g, p\}$ ] = unknown
41:   Symmetrically repeat steps 28 - 40 for viewpoints  $p \in R(g)$ .

```

■ **Algorithm 2** boolean `isNotRealizable(O)` – Part 2 of 2.

```

42: for every pair  $g_i, g_j \in G$  such that  $i < j$  and cannotBlockWithTerrain[\{ $g_i, g_j$ \}] is true do
43:   for each viewpoint  $vp$  such that leftMostPointGuardSees( $g_j$ ) <  $vp$  <  $g_i$  and  $vp$  does
      not see  $g_j$  and tooFarAway[\{ $vp, g_j$ \}] is false do
44:     if there is a  $g_k \in G$  such that tooFarAway[\{ $vp, g_k$ \}] is false and tooFarAway[\{ $g_i, g_k$ \}]
      is true and higherPoint[\{ $g_k, g_i$ \}] is  $g_k$  and tooFarAway[\{ $g_j, g_k$ \}] is false then
45:        $g_i$ CanBeBlocker = false
46:     else if there is a  $g_k \in G$  such that  $g_j < g_k$  and tooFarAway[\{ $vp, g_k$ \}] is true and
      higherPoint[\{ $vp, g_k$ \}] is  $vp$  and tooFarAway[\{ $g_i, g_k$ \}] is true and higherPoint[\{ $g_k, g_i$ \}]
      is  $g_k$  and tooFarAway[\{ $g_j, g_k$ \}] is false then
47:        $g_i$ CanBeBlocker = false
48:     if  $g_i$ CanBeBlocker is false then
49:       if cannotBlockWithTerrain[\{ $vp, g_i$ \}] is true or (there is a  $vp_2$  such that
      leftMostPointGuardSees( $g_j$ ) <  $vp_2$  <  $vp$  and  $vp_2$  sees  $g_i$  and  $vp_2$  does not see  $g_j$ 
      and tooFarAway[\{ $vp_2, g_j$ \}] is false) then
50:         return true
51:   Repeat Lines 43 - 50 symmetrically, looking for a  $vp$  to the right of  $g_j$  such that  $g_i$  needs
      a blocker to prevent it from seeing  $vp$ .
52: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
53:   if there are viewpoints  $vp_1$  and  $vp_2$  such that  $g_i < vp_2 < vp_1 < g_j$  and  $vp_1$  sees  $g_i$ 
      and tooFarAway[\{ $vp_1, g_j$ \}] is true and higherPoint[\{ $vp_1, g_j$ \}] is  $g_j$  and  $vp_2$  sees  $g_j$  and
      tooFarAway[\{ $g_i, vp_2$ \}] is true and higherPoint[\{ $g_i, vp_2$ \}] is  $g_i$  then
54:     return true
55: return false

```

■ **Algorithm 3** boolean `candidateOrderingIsExtendable(O_i)`.

```

1: if isNotRealizable( $O_i$ ) then
2:   return false
3: else if  $V_{test} \setminus O_i = \emptyset$  then
4:   return true
5: for all  $v \in V_{test} \setminus O_i$  do
6:   for  $j = 0$  to  $|O_i|$  do
7:     Let  $O_{i+1}$  be a candidate ordering by inserting  $v$  into position  $j$  in  $O_i$ .
8:     if candidateOrderingIsExtendable( $O_{i+1}$ ) then
9:       return true
10: return false

```

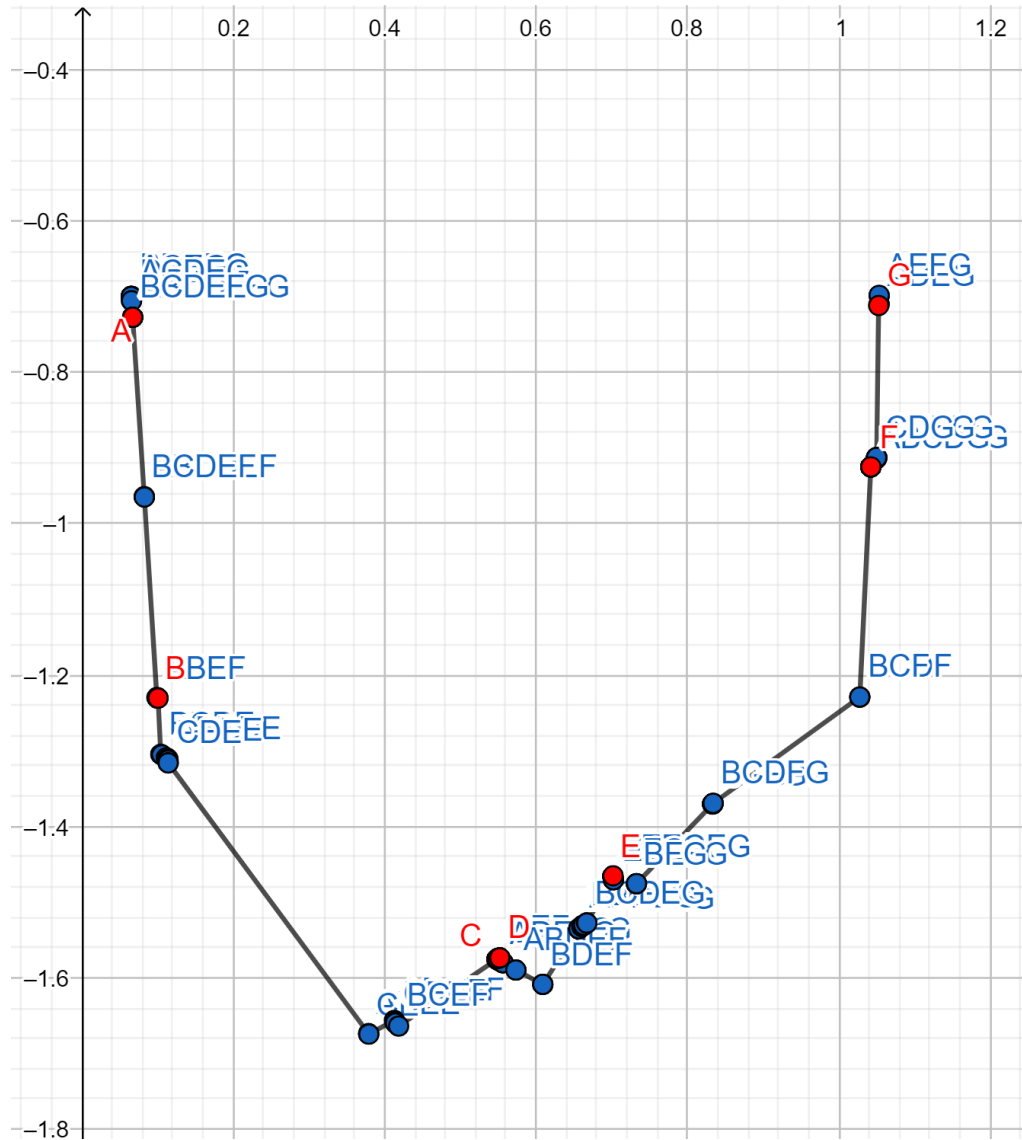


Figure 11 Terrain Boundary.

References

- 1 J. Abello, H. Lin, and S. Pisupati. On visibility graphs of simple polygons. *Congressus Numerantium*, 90:119–128, 1992.
- 2 Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '05*, pages 515–524, USA, 2005. Society for Industrial and Applied Mathematics.
- 3 Amitava Bhattacharya, Subir Kumar Ghosh, and Sudeep Sarkar. Exploring an unknown polygonal environment with bounded visibility. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner, and C. J. Kenneth Tan, editors, *Computational Science — ICCS 2001*, pages 640–648, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- 4 Therese Biedl and Saeed Mehrabi. Grid-obstacle representations with connections to staircase guarding. In Fabrizio Frati and Kwan-Liu Ma, editors, *Graph Drawing and Network Visualization*, pages 81–87, Cham, 2018. Springer International Publishing.
- 5 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.
- 6 Alon Efrat and Sarel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006. doi:10.1016/j.ipl.2006.05.014.
- 7 Sándor P Fekete, Joseph SB Mitchell, and Christiane Schmidt. Minimum covering with travel cost. *Journal of combinatorial optimization*, 24(1):32–51, 2012.
- 8 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. In *SWAT*, pages 96–104, 1988.
- 9 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997. doi:10.1007/BF02770871.
- 10 Subir Kumar Ghosh, Joel Wakeman Burdick, Amitava Bhattacharya, and Sudeep Sarkar. Online algorithms with discrete visibility - exploring unknown polygonal environments. *IEEE Robotics Automation Magazine*, 15(2):67–76, 2008. doi:10.1109/MRA.2008.921542.
- 11 Matt Gibson, Gaurav Kanade, Erik Krohn, and Kasturi Varadarajan. An approximation scheme for terrain guarding. In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 140–148, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 12 Matt Gibson, Erik Krohn, and Matthew Rayford. Guarding monotone polygons with half-guards. In Joachim Gudmundsson and Michiel H. M. Smid, editors, *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada*, pages 168–173, 2017.
- 13 Matt Gibson, Erik Krohn, and Qing Wang. The vc-dimension of visibility on the boundary of a simple polygon. In Khaled Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation*, pages 541–551, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- 14 Matt Gibson, Erik Krohn, and Qing Wang. The vc-dimension of visibility on the boundary of monotone polygons. *Computational Geometry*, 77:62–72, 2019. Canadian Conference on Computational Geometry. doi:10.1016/j.comgeo.2018.10.006.
- 15 Alexander Gilbers and Rolf Klein. New results on visibility in simple polygons. In Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures*, pages 327–338, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 16 James King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640. Springer, 2006. doi:10.1007/11682462_58.
- 17 James King. Vc-dimension of visibility on terrains. In *In Proc. 20th Canadian Conference on Comput. Geom.*, pages 27–30, 2008.
- 18 James King and Erik Krohn. Terrain guarding is np-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011. doi:10.1137/100791506.

- 19 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. Guarding terrains via local search. *JoCG*, 5:168–178, 2014.
- 20 Erik Krohn and Bengt J. Nilsson. The complexity of guarding monotone polygons. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG 2012, Charlottetown, Prince Edward Island, Canada, August 8-10, 2012*, pages 167–172, 2012. URL: <http://2012.cccg.ca/papers/paper26.pdf>.
- 21 Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013. doi:10.1007/s00453-012-9653-3.
- 22 Erik A. Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, pages 1–31, 2012.
- 23 T. Prellberg. Uniform q-series asymptotics for staircase polygons. *Journal of Physics A: Mathematical and General*, 28(5):1289–1304, March 1995. doi:10.1088/0305-4470/28/5/016.
- 24 Abdulmuttalib Turkey Rashid, Abduladhem Abdulkareem Ali, Mattia Frasca, and Luigi Fortuna. Path planning with obstacle avoidance based on visibility binary tree algorithm. *Robotics and Autonomous Systems*, 61(12):1440–1449, 2013. doi:10.1016/j.robot.2013.07.010.
- 25 Sven Schuierer, Gregory J. E. Rawlins, and Derick Wood. A generalization of staircase visibility. In H. Bieri and H. Noltemeier, editors, *Computational Geometry-Methods, Algorithms and Applications*, pages 277–287, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 26 Israel A. Wagner, Michael Lindenbaum, and Alfred M. Bruckstein. Mac versus pc: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *The International Journal of Robotics Research*, 19(1):12–31, 2000. doi:10.1177/02783640022066716.