May 2018

Algorithms for Tower Placement on Terrain

Binay Dahal

ALGORITHMS FOR TOWER PLACEMENT ON TERRAIN

by

Binay Dahal

Master of Science (M.S.)
University of Nevada, Las Vegas
2018

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2018

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

May 11, 2018

This thesis prepared by

Binay Dahal

entitled

Algorithms for Tower Placement on Terrain

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Laxmi Gewali, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

John Minor, Ph.D.
Examination Committee Member

Kazem Taghva, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Graduate College Faculty Representative

Abstract

We review existing algorithms for the placement of towers for illuminating 1.5D and 2.5D terrains. Finding the minimum number of towers of zero height to illuminate 1.5D terrain is known to be NP-Hard. We present approximation algorithms for solving two variations of the tower placement problem. In the first variation, we consider the placement of a single tower of given height to maximize visibility coverage. In the second variation, we consider the problem of placing reduced number of common height towers to cover the entire terrain. Algorithms for solving both problem variations are based on discretizing the problem domain by carefully identifying feasible placement points. We also present a Java implementation for placing a single tower of minimum height to illuminate a given 1.5D terrain.

Acknowledgements

First of all, I would like to thank my advisor Dr. Laxmi Gewali, for his continuous guidance and help. His enthusiasm and knowledge have been decisive factors on the completion of this thesis. I would also like to thank all of my committee members for their constant suggestions to improve my work.

Next, I would like to thank Dr. Rama Venkat for helping us to prove one of the lemmas, which enabled us to assert that our proposed algorithm is correct and sufficient to determine the solution.

I'm also grateful to one of my friends Mr. Bibek Subedi, for providing an extra pair of eyes to identify the typos in the report. Big thanks to all my friends and my family members for showing concern on my progress throughout.

Binay Dahal

University of Nevada, Las Vegas

May 2018

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Problems dealing with visibility on the surface of terrain have attracted the interest of many researchers in diverse scientific areas that include (i) geographic information systems, (ii) path planning for aerial vehicles, (iii) transportation networks, (iv) emergency response planning and (v) wireless communications. In geographic information system(GIS) the topography of the terrain is modeled by discretizing the surface by placing nodal points. Each nodal point p_i is specified by three integers x_i,y_i,h_i, where x_i and y_i denote x- and y-coordinates of p_i and h_i denotes the elevation. The nodal points are carefully connected by edges to obtain what is called a triangulated irregular network(**TIN**). For the purpose of visibility computation, each triangle of TIN is assigned an index called *visibility index*. Two triangles t_1 and t_2 of TIN are *visible* to each other if representative points of t_1 and t_2 are such that the line segment connecting t_1 to t_2, lies above the terrain i.e. it does not intersect with the terrain. While some triangles are visible from many other triangles, there could be other triangles that are visible only from a very few other triangles. Triangles having high visibility have a high *visibility index*. Such visibility indices are used in planning road networks, locating facility centers, positioning cellular towers, and modeling reconnaissance trajectories for aerial vehicles.

In a telecommunication network it is required to construct towers on the surface of terrain to cover a given region. Just placing towers on triangles having a high visibility index may need a prohibitively large number of towers. This issue has attracted the interest of many researchers from the algorithm community to develop efficient algorithms for covering a given region of TIN with only a small number of towers. Most researchers have adopted the convention of *line-of-sight* communication for developing tower placement algorithms. In line-of-sight communications, two towers can directly communicate with each other if they are in each other's line of sight, i.e. the

line segment connecting the top of the towers does not intersect with the terrain. In rare cases, towers not in line-of-sight, may be able to communicate by exchanging feeble signals. However, most researchers have adopted the line-of-sight model as fairly good, adequate, and intuitive for many applications, and in this thesis we stick with this model.

In this thesis, we examine algorithmic approaches for the placement of towers in terrain. Some versions of tower placement problems are known to be intractable[KK11] and consequently our motivation is in the development of tower placement algorithms that are efficient and easy to implement. In chapter 2, we present a critical review of groundbreaking algorithms reported in publication avenues. In chapter 3, we present the main contribution of the thesis. We first formulate the Tower Placement Problem(TPP) and present an $O(n^2)$ algorithm that finds the location for placing a tower of given height to maximize the visible region in given 1.5D terrain. We also present a greedy heuristic to place a reduced number of common height watch towers that cover the entire 1.5D terrain. In Chapter 4, we present an implementation of two algorithms dealing with the placement of watch towers. The first algorithm we implement is the computation of shortest towers and their placement for covering the entire 1.5D terrain. The second algorithm we implement is the placement of a tower of given height to maximize the coverage area. The implementation is done in the JAVA programming language.

In Chapter 5, we discuss the experimental results of the implemented algorithms and examine approaches for making them robust and reliable. We also propose interesting variations of tower placement problems that can be pursued in the future.

Chapter 2

Review of Terrain Visibility Algorithms

2.1 Preliminaries

A *terrain surface* is usually modeled by a collection of covering triangles. Such a model is extensively used in geographic information system (**GIS**) and finite element analysis. The network of line segments for representing a terrain surface by covering triangles is also known as a triangulated irregular network (*TIN*). The terrain surface satisfies an interesting structural property called the *projection containment property* which can be elaborated as follows. The term *h-crosssection* is used to indicate the intersection between the terrain and a horizontal plane. The area of *h-crosssection* I_{h_i} at height h_i decreases monotonically as height h_i increases. Specifically, consider two h-sections I_{h_1} and I_{h_2} at height h_1 and h_2 (h_1 above h_2). The projection of I_{h_1} on the horizontal plane at h_2 is contained inside I_{h_2}. This *projection containment property* has been used extensively to develop efficient algorithms for solving geometric problems on terrain. Figure 2.1 is an illustration of the *projection containment property*.

Due to the validity of the projection inclusion property, computational geometry investigators often refer to terrain as a geometric shape in two and half dimension or simply **2.5D-terrain**: the dimensionality of a terrain is viewed between two dimensions (2D) and three dimensions (3D). When the terrain is restricted to two dimensions, the surface becomes a monotone polygonal chain, monotone along the x-axis. It is noted that in a x-monotone chain Ch, any vertical line intersects with Ch in at most one point. Consequently, a terrain in two dimensions is viewed as a *1.5D-terrain*. A 1.5D terrain is illustrated in Figure 2.2.

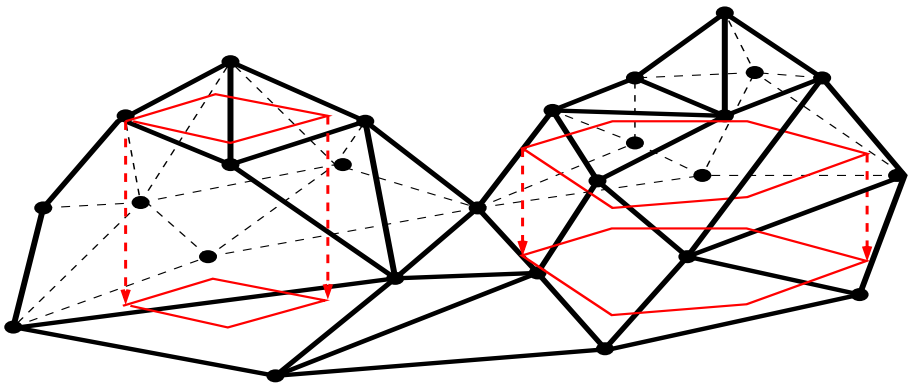


Figure 2.1: Illustrating projection property

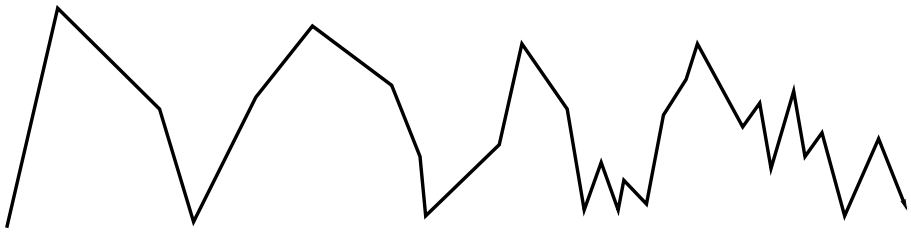


Figure 2.2: A 1.5D terrain

2.2 Placement of Single Tower

One of the extensively investigated problems on terrain visibility is the placement of shortest tower(s) on the surface of 2.5D terrain so that all points on the surface are visible from the top of the tower. The problem can be formally stated as follows:

Shortest Tower Problem (STP)

Given: A 2.5D terrain L.

Question: Find a position $p_0(x, y)$ to place a shortest vertical tower on L so that all points on L are visible from the top of the tower.

It is noted that a point $p_i(x_i, y_i)$ on the surface of L is visible from the top point t_p of tower if the line segment connecting t_p to p_i does not intersect with the surface of L. Details about the concept of visibility can be found in O'Rourke's book[O'R87].

One of the first algorithms for computing shortest tower was reported by Sharir[Sha88]. Sharir's paper establishes that STP reduces to the problem of computing the shortest distance between two

polyhedrons L and S. Polyhedron S is formed by the intersection of half planes formed by the 2D faces of L. It turns out that while L is not a convex polyhedron, S is convex. For the purpose of clarity of presentation, we can illustrate the formation of S in 1.5D as shown in Figure 2.3

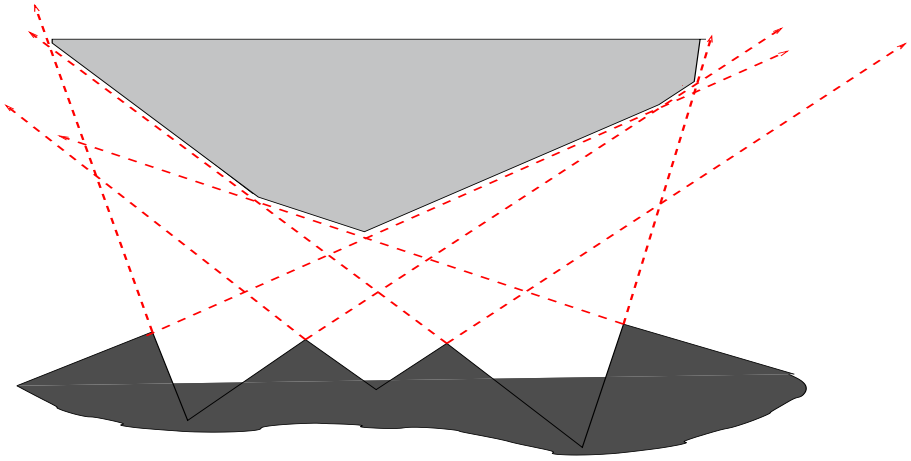


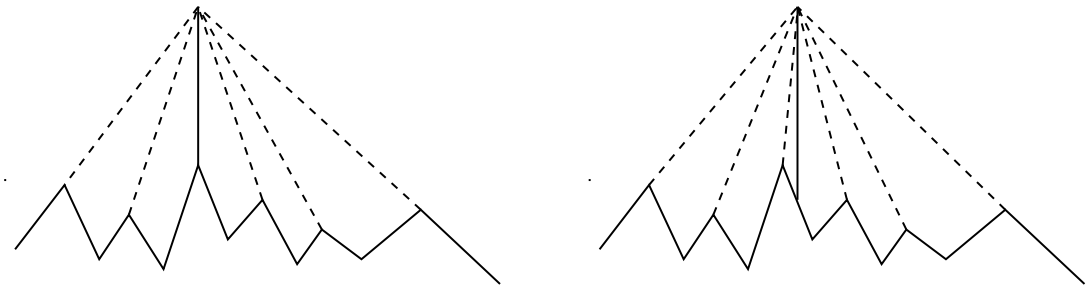
Figure 2.3: Illustrating shapes S and L used in Sharir's algorithm

In the figure, we illustrate Sharir's idea in 1.5D terrain. The area below the terrain can be represented by a simple polygon (not necessarily convex) and the intersections of half planes is represented by a convex polygon which we call the **reference polygon**. In Figure 2.3, the polygon representing terrain is filled with a darker shade and the convex region is filled with a lighter shade. To construct the reference polygon, Sharir[Sha88] used the idea of intersecting rays that originate from segments of the terrain and extend above. Each of these rays defines a half plane (either to the left or to the right as appropriate). Specifically, for a ray proceeding to the north-east direction, the half plane is to the left of the ray. Similarly, for the rays proceeding to the north-west direction the half plane is to the right of the ray. The intersection of these half planes precisely forms the reference convex polygon. It is remarked that the reference convex polygon is unbounded. Sharir proved by geometric analysis that the point on the terrain that minimizes the distance to the reference polygon is the point where the shortest watch tower should be located. To sketch the resulting algorithm, three cases are distinguished. The first case is to find the distance between a vertex of the reference polygon and a line segment of the terrain. The second case is to find the distance between a vertex of the terrain and the reference polygon. Finally, the third case is to find the distance between a line segment of the reference polygon and a line segment of the terrain. While the first two cases can be solved easily in $O(nlogn)$ time by using a standard technique in computational geometry [o'R98]. The third case is slightly complicated and intricate point location

techniques are used in [Sha88] to obtain the distance in $O(nlog^2n)$ time.

It took another nine years to obtain a faster algorithm for solving the shortest watch tower problem. Binhai Zhu[Zhu97] reported a faster algorithm. Binhai used Dobkin-Kirkpatrik's[DK85] hierarchical representation of convex polyhedron to store additional information on the polyhedron. This approach resulted in a faster algorithm which executes in $O(nlogn)$ time.

One of the difficulties in developing efficient algorithms for solving STP is the fact that the shortest tower can potentially be at any point on the surface of the terrain. Figure 2.4 shows the situations where the shortest tower can be either at a vertex or at an interior point on the edge of 1.5D terrain.



(a) Shortest tower at a vertex (b) Shortest tower at an interior point

Figure 2.4: Placement of towers at different positions

2.3 Placement of Two Towers

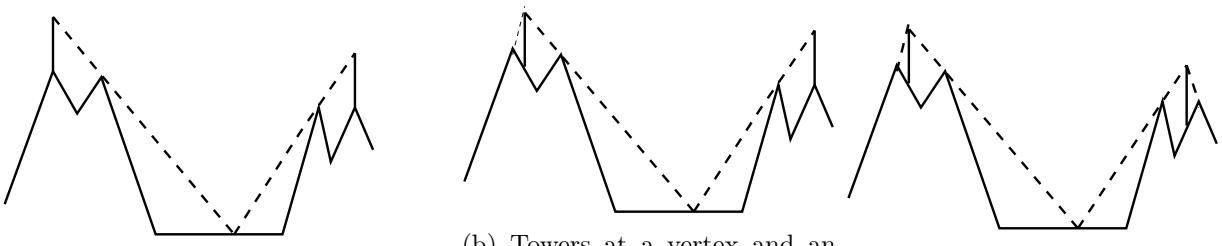
Illuminating terrain by the placement of two towers has been investigated. The problem can be formally stated as follows.

Two Tower Placement Problem (TTPP)

Given: A 2.5D terrain L

Question: Find the placement of two towers of common smallest height to cover L.

This problem can be further stated in two version. In the first version *(the discrete version)*, the base of the tower is restricted to be among the vertices of L. In the second version (called the *continuous version*) the base of the tower could be anywhere on the surface of the tower. As observed in Agarwal et. al.[ABD$^+$05] the optimal solution for the TTPP could be either on vertices or on interior points as shown in the figure below.



(a) Both towers at vertices

(b) Towers at a vertex and an edge

(c) Bothe towers at edges

Figure 2.5: Three version of two-watchtowers

By using a parametric search technique, it is established in[ABD$^+$05] that the discrete two-watchtower position can be determined in $O(n^2 log^4 n)$ time, where n is the number of edges in 1.5D terrain. It is further shown in [ABD$^+$05] that within the same time complexity, the semi-continuous version of the two-watchtower problem can be solved. It is remarked that in the semi-continuous version, one of the towers can be anywhere while the other is required to be placed at one of the vertices. For the continuous version of the two-watchtower problem, it is proved in [ABD$^+$05] that the optimum placement points can be computed in $O(n^3 \alpha(n) log^3 n)$ time, where $\alpha(n)$ is the inverse of the Achermann function.

2.4 Intractability and Approximation

The Tower placement problem is closely related to the well known art gallery problem[O'R87] of computational geometry. In the art gallery problem, it is asked to find the minimum number of point guards inside a simple polygon so that any point in the interior of the polygon is visible to some point guard. It is noted that a point g_i sees a point p_j inside the polygon if the line segment (g_i, p_j) does not intersect with the exterior of the polygon. The standard art gallery problem is known to be NP-Hard[O'R87]. A 1.5D terrain can be viewed as a part of a monotone polygon. The complexity of finding the minimum number of point guards to illuminate a 1.5D terrain, often called the **Terrain Illumination Problem** (TIP) was not settled for quite some time. Finally, in 2010, King and Krohn[KK11] were able to build a relationship between TIP and a variation of the **3-SAT** problem called **planar 3-SAT**. In the standard 3-SAT problem we are given a logical expression E in Conjunctive Normal Form (CNF) where each clause in E contains at most 3 literals[GJ02] and we are asked to determine whether there is an assignment to the variable of E to make it satisfiable. In the planar 3-SAT problem the graph of the logical expression (GLE) is required to be planar.

For example, consider, a logical expression.

$$E_1 = (v_1 \vee v_3 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_2} \vee v_3) \wedge (\overline{v_2} \vee \overline{v_3} \vee \overline{v_4})$$

In GLE, each variable appears as a circular node and each clause represents a square node.

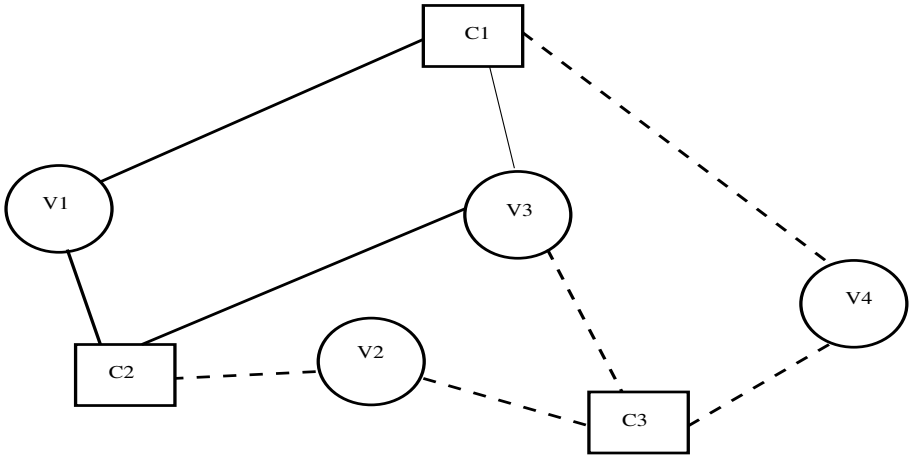


Figure 2.6: Illustrating planar 3-SAT graph

The edges of GLE consists of positive literal edges and negative literal edges. A positive edge connects a clause with a positive literal, and a negative edge connects a clause with a negative literal. In Figure 2.6, positive edges are drawn as solid lines and negative edges are drawn as dashed lines. In a planar 3-SAT it is required that the GLE be planar, i.e no two edges of GLE intersect. King and Krohn reduced the TIP problem to the planar 3-SAT problem. Since, planar 3-SAT is known to be NP-Complete[GJ02], it implies that TIP is also NP-Complete.

The minimum tower placement problem (**minTP**) asks to find the minimum number of towers of a given common height so that all points on the surface of 1.5D terrain is covered. Now, TIP can be viewed as a restricted case of min TP in which the height of the tower is zero. In this sense, the complexity of minTP is also NP-Hard. However, if the common height of the tower is required to be non-zero then the complexity of minTP is still open.

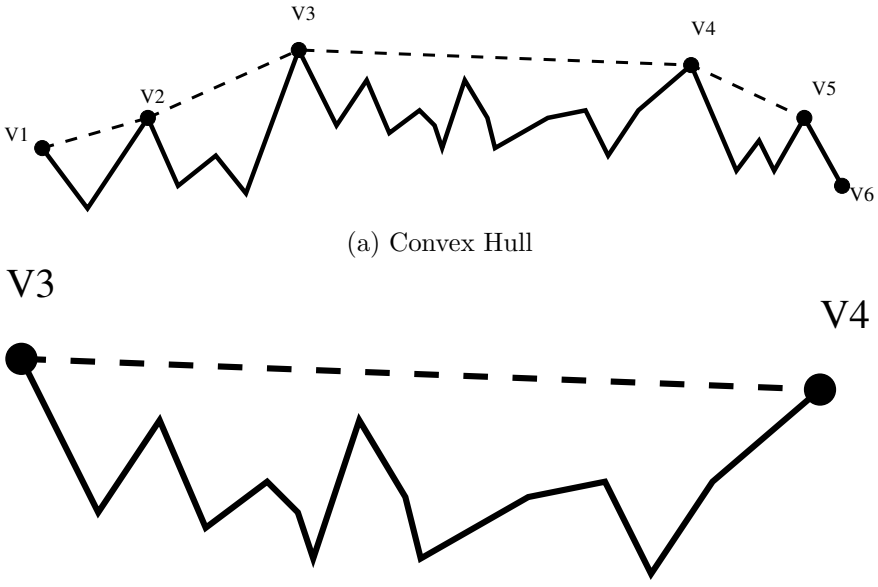
Some interesting approximation algorithms for solving TIP have been proposed.

One of the first such algorithms was reported by Stephen Eidenbenz in [Eid02]. The approach taken in this paper is the development of a relationship between the minimum setcover (*minSet*) problem and minTP. The minimum setcover problem (minSet) is a well known intractable problem [GJ02] and a few approximation algorithms have been reported [GJ02]. Specifically, in the minSet problem, two sets (i) $E = \{e_1, e_2, ..., e_n\}$ and (ii) $S = \{s_1, s_2, ..., s_m\}$ are given, where each s_i is a

subset of E. The minSet problem asks to find a minimum subset S' of S such that every element of E is in at least one member of S'. This problem is known to be NP-Hard [GJ02] and an approximation algorithm with approximation ratio $\log n+1$ is known[GJ02].

In [Eid02] the space above the 2.5D terrain is partitioned into 3D convex cells. These cells can be viewed as set S in the minSet problem. Analyzing this approach, it is established in [Eid02] that an approximation algorithm for solving minTP can be developed. The approximation ratio is also O($\log n$). The time complexity of the algorithm is O(n^6). This algorithm is of theoretical interest and not efficient enough for practical application.

Another approximation algorithm for covering 1.5D terrain is published in [BMKM07]. This algorithm is based on placing point guards(watchtowers of zero height) at (i)the vertices of the convex hull *CH(T)* of terrain T and (ii) at the carefully selected vertices on **sub-terrains** defined by consecutive vertices of *CH(T)*. This is illustrated in Figure 2.7.



(a) Convex Hull

(b) Sub-terrain

Figure 2.7: Convex Hull and Sub-terrain

The dashed chain is the convex hull *CH(T)* of given terrain *T*. The convex hull has six vertices $CH(T) =< v_1, v_2, ...v_6 >$ and there are four sub-terrains. The third sub-terrain induced by *CH(T)* is shown Figure 2.7b. A complicated and intricate case analysis is done in [BMKM07] to select desired vertices for placement in sub-terrains. It is reported in this paper that the resulting algorithm yields a constant factor approximation for placing guards on terrain T. It is however not clear about the value of the constant factor.

Chapter 3

Placement Algorithms

In this chapter we formulate the problem of placing a vertical tower of given height in a 1.5D terrain so that the portion of the terrain visible from the top of the tower is maximized.

3.1 Problem Formulation

We are given a 1.5D terrain T_1 and a watch tower R_1 of height h_1. Find a placement of R_1 so that the portion of T_1 visible from the top of the tower is maximized. The problem is relevant when the height of the tower is not long enough to visibly cover the whole terrain. It was observed in Chapter 2 that the solution to a single tower placement problem need not be in one of the vertices of the terrain. When the solution is one of the interior points on the edge of the terrain it is not clear how to locate such a point. The placement problem can be formally stated as follows:

3.1.1 Tower Placement Problem (TPP)

Given: (i) A 1.5D terrain T_1, (ii) A tower R_1 of height h_1.
Question: Find the location on the terrain to place tower R_1 such that the portion of T_1 visible from the top of the tower is maximized.

If we move the tower from the leftmost points in T_1 to the right then the length of T_1 visible from the top of R_1 changes. At some intervals the change in visible length is gradual (increasing or decreasing), while at some intervals the change is abrupt and discontinuous.

Definition 3.1: (Transition Point) The placement point on the terrain that corresponds to a discontinuity in visible length is called **transition point**. In Figure 3.2, there are 12 transition points.

Definition 3.2: (Critical Points) The set of points on the terrain consisting of transition points and terrain vertices are called **critical points**.

Definition 3.3 (Basic Interval) The interval on the terrain between two consecutive critical points is referred to as a **basic interval**.
A transition point could be any point on the terrain. For a given terrain, the transition points depends on (i) the structural shape of the terrain, and (ii) the height of the tower.

Consider the change in visibility (portions of terrain) from the top of the tower as its placement moves along the basic internal segment. Visible portions of terrain consist of several sub-segments (we refer to them **v-edges**). As the tower moves, some v-edges shrink and other v-edges expand, increasing or decreasing their lengths monotonically. This can be established as stated in the following lemma.

Lemma 3.1: The change in the length of *v-edges* as the tower moves along the basic interval is increasing or decreasing monotonically. (We thank Professor Dr. Rama Venkat's help in the calculus part of this lemma.)

Proof: Without loss of generality, we consider a v-edge whose length increases as the tower moves left to right along the basic interval.(Figure 3.1)

Given constant values of θ_1, θ_2 and h_1, we need to develop a functional relation between l_1 and l_2

As h_1 moves along l_1 (keeping its height h, and θ_1 to the slope constant), θ will change. So does l_1 and l_2. L is the maximum possible visibility length. Actually in this case, θ increases while l_1 and l_2 decrease. Of course, l_2 decreasing means that visibility increases. In other words, $L - l_2$ increases.

Note: $\theta_3 = (180 - \theta_1) - \theta$

$= \phi - \theta$, where $\phi = 180 - \theta_1$ which is a constant.

Law of Sines for ΔABC

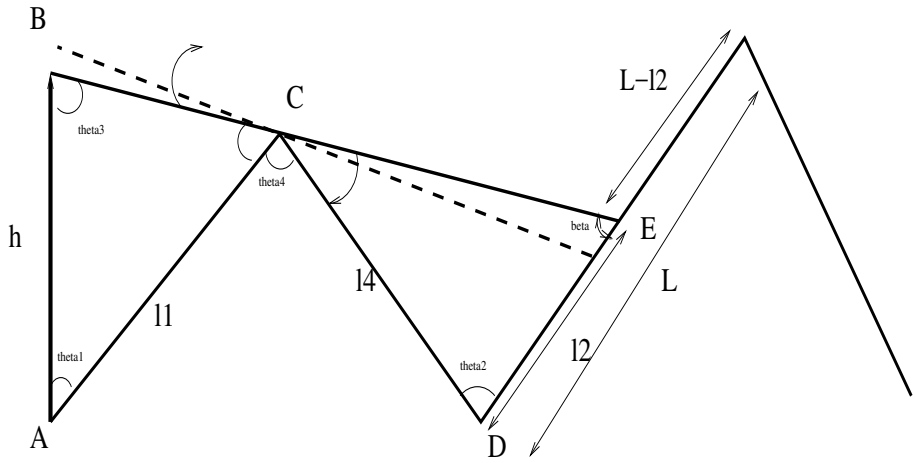


Figure 3.1: Moving tower of h_1 along AC

$$\frac{l_1}{\sin(\theta_3)} = \frac{h}{\sin(\theta)}$$

$$l_1 = h\frac{\sin(\phi - \theta)}{\sin(\theta)}$$

For ΔCDE

$$\gamma = \angle ECD = (180 - \theta_4) - \theta$$

$$\gamma = \alpha - \theta \quad where \quad \alpha = 180 - \theta_4 \quad is \quad a \quad constant$$

$$\beta = \angle CED = 180 - \angle ECD - \theta_2$$

$$= 180 - (180 - \theta_4 - \theta) - \theta_2$$

$$= \theta_4 - \theta_2 + \theta$$

Law of cosines for ΔCDE gives:

$$\frac{l_4}{\sin(\beta)} = \frac{l_2}{\sin(\gamma)}$$

$$l_2 = l_4\frac{\sin(\gamma)}{\sin(\beta)} = l_4\frac{\sin(\alpha - \theta}{\sin(\theta_4 - \theta_2 + \theta)}$$

Note $l_4 = \quad constant$

We have,

$$l_1 = h\frac{\sin(\phi-\theta)}{\sin(\theta)} \quad - \quad - \quad -(i)$$

$$l_2 = l_4\frac{\sin(\alpha-\theta)}{\sin(\theta_4-\theta_2+\theta)} \quad - \quad - \quad -(ii)$$

Differentiating l_1 and l_2 with respect to θ,

$$\frac{dl_1}{d\theta} = h\frac{-\cos(\phi-\theta)\sin(\theta)-\cos(\theta)\sin(\phi-\theta)}{\sin^2(\theta)}$$

$$= (-h)\frac{\sin(\theta+\phi-\theta)}{\sin^2(\theta)}$$

$$\frac{dl_1}{d\theta} = -h\frac{\sin(\phi)}{\sin^2(\theta)}$$

Now for l_2,

$$\frac{dl_2}{d\theta} = l_4\frac{-\cos(\alpha-\theta)\sin(\theta_4-\theta_2+\theta)-\sin(\alpha-\theta)\cos(\theta_4-\theta_2+\theta)}{\sin^2(\theta_4-\theta_2+\theta)}$$

$$= -l_4\frac{\sin(\theta_4-\theta_2+\theta+\alpha-\theta)}{\sin^2(\theta_4-\theta_2+\theta)}$$

$$= -l_4\frac{\sin(\theta_4-\theta_2+\alpha)}{\sin^2(\theta_4-\theta_2+\theta)}$$

$$\frac{dl_2}{dl_1} = \frac{dl_2}{d\theta}.\frac{d\theta}{dl_1} = (\frac{l_4}{h})\frac{\sin(\theta_4-\theta_2+\alpha)\sin^2(\theta)}{\sin(\phi)\sin^2(\theta_4-\theta_2+\theta)}$$

$$\frac{dl_2}{dl_1} = (\frac{l_4}{h})\frac{sin(180-\theta_2)}{\sin(180-\theta_1)}\frac{\sin^2(\theta)}{\sin^2(\theta_4-\theta_2+\theta)}$$

$$\frac{dl_2}{dl_1} = (\frac{l_4}{h})\frac{sin(\theta_2)}{\sin(\theta_1)}\frac{\sin^2(\theta)}{\sin^2(\theta_4-\theta_2+\theta)}$$

The key functional behavior is:

$$\frac{dl_2}{dl_1} \quad \propto \quad \frac{\sin^2(\theta)}{\sin^2(\theta_4-\theta_2+\theta)} = \frac{X}{Y}$$

From the above relation we conclude that X increases monotonically, Y decreases monotonically and $\frac{X}{Y}$ increases monotonically. This means $\frac{X}{Y}$ does not contain extremum (minimum or maximum) in their interior.

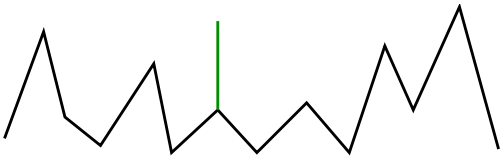
3.2 Computing Transition Points

Consider the image **Im(T_1,h_1)** of terrain T_1, formed by lifting it by height h_1 of the tower. The image is shown in Figure 3.2a drawn in thin segments. From each peak points z_i of the terrain we can construct two **grazing rays** r_{left} and r_{right} that originate at the peak point and extend upward along the terrain edges incident on z_i. In Figure 3.2, grazing rays are drawn as dashed edges. The points of intersections between grazing rays and terrain image Im(T_1,h_1) are referred to as **guiding points**. Guiding points are illustrated in Figure 3.2d drawn as small red circles. We can project guiding points vertically downward on the terrain to obtain the **transition points**, drawn as small blue circles in Figure 3.2e.
A straightforward algorithm for computing transition points is to directly use their constructive definition. Such an algorithm can be described as follows:

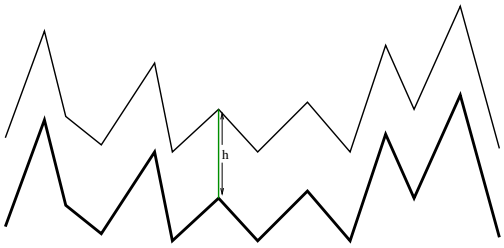
The image chain $Im(T_1, h_1)$ can be constructed by adding height h_1 of tower to the y-coordinates of terrain chain T_1. Specifically, if (x_i, y_i) is the co-ordinate of vertex v_i of T_i then the coordinates of the corresponding image $Im(T_1, h_1)$ is $(x_i, y_i + h_1)$. Grazing rays from each vertex v_i of terrain T_1 can be constructed by using the slope of segments incident on v_i in constant time. We can then check for intersection between grazing rays and segments of image chain $Im(T_1, h_1)$. A formal sketch of our algorithm based on this straightforward approach is listed as Algorithm 1

Algorithm 1 Straightforward Intersection Algorithm for Computing Transition Points

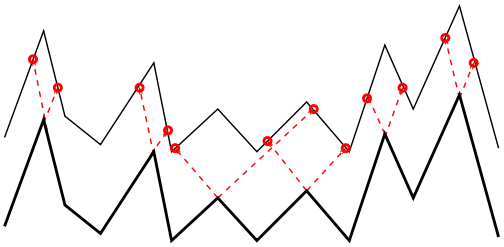
Input: (i) Terrain T_1, (ii) Tower height h_1
Output: Transition points U=$\{u_1,u_2,...,u_k\}$
Construct Im(T_1,h_1) by lifting T_1 by h_1
V= $\varnothing$
for each n in N **do**
    Construct grazing rays r_{left} and r_{right} for z_i
    Let W_i be the intersection points between Im(T_1,h_1) and grazing rays r_{left} and r_{right}
    Add W_i to V
Project points in V vertically downward to T_1 to obtain U
Output U



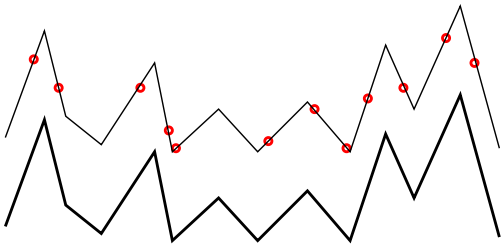
(a) Placement of a tower on 1.5D Terrain



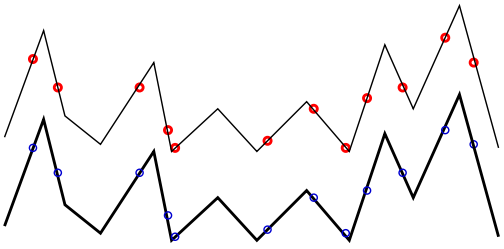
(b) Lifting-up terrain image by h units



(c) Extending rays from top-vertices



(d) Marker points formed by ray extension (red dots)



(e) Projecting marker point to construct transition points

Figure 3.2: Different stages of finding transition points

Observation 3.1: One peak point can potentially trigger *O(n)* guiding points. This happens when a grazing ray originating from a peak point intersects with almost all edges of the terrain's image This is shown in Figure 3.3

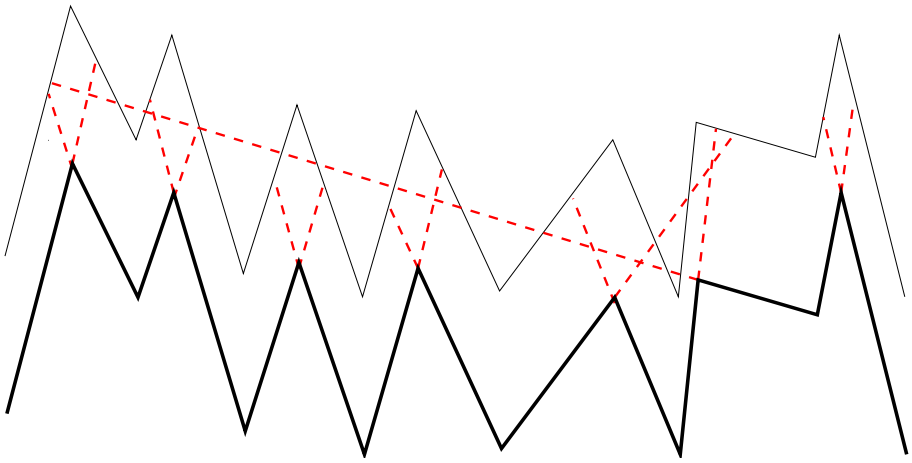


Figure 3.3: Example of a grazing ray inducing O(n) transition segments

Computing guiding points by using Algorithm 3.1 is rather slow. Observation 3.1 leads to the conclusion that Step 7 in Algorithm 3.1 can take $O(n)$ time for computing intersection points corresponding to one pair of grazing segments. Since there are $O(n)$ peaks, a straightforward approach for computing guiding points can take $O(n^2)$ time.

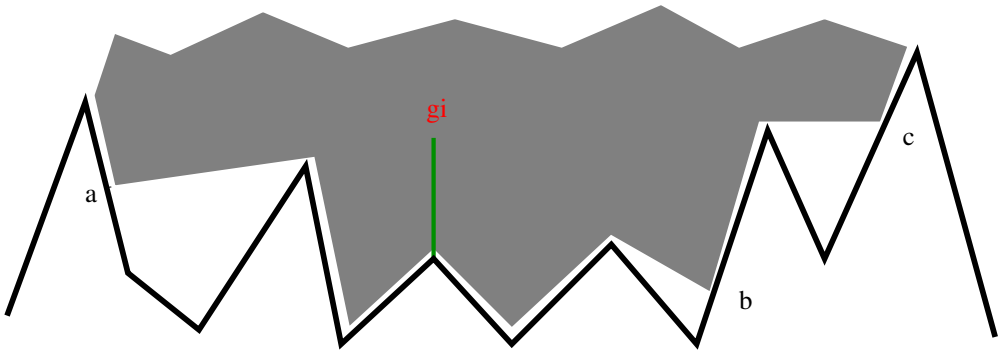
By using the plane sweep technique of computational geometry [o'R98], all guiding points can be computed more efficiently. The approach is to sweep a vertical line from left to right and maintain two data structures (i) a height balanced tree **Tr** to maintain the segments intersected by the sweep line and (ii) a priority queue **Q** to store segment endpoints and candidate intersection points to the right of the sweep line. When the sweep line is on an endpoint p_i, all intersection points to the left of the sweep line are discovered together with some implied intersections points to the right of the sweep line. When the sweep line is at the right end point of a segment e_i (**event 1**), it is removed from the tree **Tr**. Similarly, when the sweep line is at the left end of segment e_i (**event 2**), it is inserted into the tree **Tr**. During **event 1** and **event 2**, when possible intersection point p_j is indeed found, then p_j is inserted into queue **Q**. Whenever the sweep line is on an intersection point p_j (**event 3**), the order of the corresponding segments (intersecting at p_j) are interchanged. The algorithm finds all intersection points when all edges are processed by a left to right sweep. A formal description of this algorithm is listed as Algorithm 2.

Algorithm 2 Plane Sweep Algorithm for Computing Transition Points

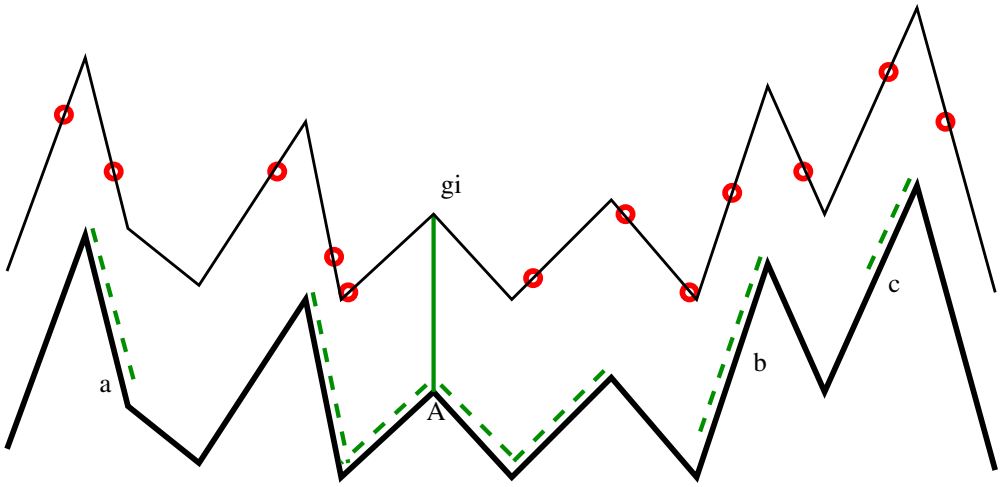
1: **Tr**=$\varnothing$ Initialize search tree **Tr**
2: (i) $Q = Q \cup$ Endpoints of edges of $\mathbf{Im(T_1, h_1)}$ Initialize priority queue Q to endpoints of edges of image chain
3: (ii) $Q = Q \cup$ Endpoints of grazing segments Initialize priority queue Q to endpoints of edges of grazing segments
4: **while** Q is not empty **do**
5: Let p be the point with minimum x-coordinate in Q
6: Delete p from Q
7: **if** p is a left endpoint of edge e_j **then**
8: Insert e_j into **Tr**
9: Let e_i, e_k be two neighbors of e_j in **Tr**
10: Insert $\cap(e_i, e_j)$ and $\cap(e_j, e_k)$ into Q
11: **if** p is a right endpoint of edge e_j **then**
12: Let e_i,e_k be neighbors of e_j in **Tr**
13: Delete e_j from **Tr**
14: Insert $\cap(e_i, e_k)$ into Q if the intersection is to the right of sweep line
15: **if** p is $\cap(e_i, e_j)$ **then**
16: e_i and e_j are necessarily adjacent in **Tr**
17: Interchange e_i, e_j in **Tr**
18: Let e_h, e_k be the neighbors of e_i and e_j in **Tr**
19: Insert $\cap(e_h, e_i)$ and $\cap(e_j, e_k)$ into Q if they are to the right of the sweep line
20: Output p

3.3 Maximizing Tower's Coverage

Once we have the critical points, we are ready to describe an algorithm for placing a tower T_1 of given length h_1 to maximize coverage. Consider the **visibility polygon** $VP(g_i)$ from a critical point g_i as shown in Figure 3.4a. The interior of $VP(g_i)$ is shaded in the figure. The visibility polygon from a point inside a simple polygon can be computed in *O(n)* time [DBVKOS00]. The portion of T_1 visible from g_i, denoted by $L(T_1, g_i)$, can be extracted from $VP(g_i)$ is straightforward manner. $L(T_1, g_i)$ is indicated in Figure 3.4b where the visible portions of terrain edges are indicated by dashed edges. The Visible Portion $L(T_1, g_i)$'s from all critical vertices are computed and we select the one that maximizes the length of the visible portions. A formal sketch of the algorithm is listed as Algorithm 3.



(a) Visibility polygon VP(g_i) from point g_i



(b) Illustrating visible portions from position A

Figure 3.4: Visibility polygon and visible edge portions

Algorithm 3 Placement to maximize coverage

Input: (i) Terrain T_1, (ii)Height h_1
Output: Placement point $t^{'}$ on T_1 that maximizes coverage

1: Compute critical points g_1,g_2, ... ,g_m using Algorithm 2
2: **for** each point g_i **do**
3: Compute Visibility polygon $VP(g_i)$
4: Extract $L(T_1, g_i)$ from $VP(g_i)$
5: Set $g^{'}$ to g_i that maximizes $L(T_1, g_i)$
6: Project down $g^{'}$ to T_1 to obtain $t^{'}$
7: Output $t^{'}$

3.4 Covering the Entire Terrain

If the length of the tower is not long enough it would be interesting to develop an algorithm for covering the whole terrain by placing only a small number of towers. If the length of the tower is zero then the problem of finding the minimum number of tower placement is NP-Hard [KK11]. This motivates us to develop a good heuristic to place a reduced number of towers of common height h_1 to cover the entire 1.5D terrain T_1.

The algorithm we present is a greedy algorithm that determines placement points, incrementally, one placement at a time, by scanning the terrain left to right.

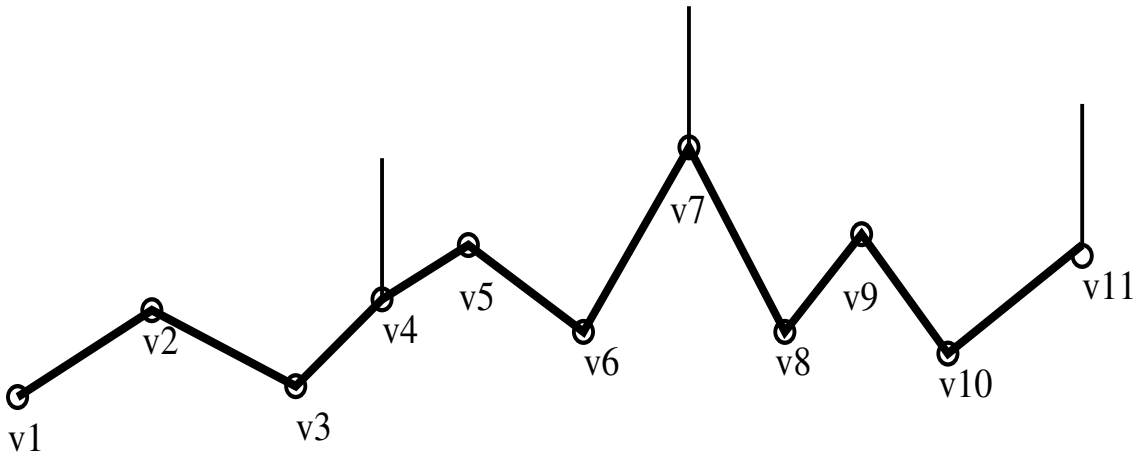


Figure 3.5: Illustrating placement by RT-Algorithm

We start with a few definitions needed to describe the greedy heuristic. Consider a 1.5D terrain whose left to right vertices are in the order v_1,v_2,...,v_n as shown in Figure 3.5.

Definition 3.4 (**RL-vertex**) For a given height h_1, let **RL(**T_1**,**h_1**)**=v_j be the *rightmost left covering* vertex such that by placing the tower at v_j, all edges of T_1 to the left of v_j are visible from v_j. In Figure 3.5, **RL(**T_1**,**h_1**)** is given by v_4.

Definition 3.5 (**LM-Chain**) Let $LM(T_1, h_1)$ be the maximal leftmost sub-terrain containing no sub-edges of T_1 invisible from the tower placed at vertex $RL(T_1, h_1)$. In Figure 3.5, $LM(T_1, h_1)$ is given by the chain $< v_1, v_2, v_3, v_4, v_5 >$.

The greedy heuristic we propose, which we call **Left Covering RightMost** heuristic (**LCRM-heuristic**), essentially identifies **RL-vertices** and **LM-chains**, one at a time, by performing a right to left scan of the terrain. Once the first RL-vertex and corresponding LM-chain is identified, terrain T_1 is updated by deleting the LM-chain from T_1. In our running example, the updated

terrain T_1 is $< v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11} >$. In the second greedy iteration, the RL-vertex is v_7 and the corresponding LM-chain is $< v_5, v_6, v_7, v_8 v_9 >$. In the third greedy iteration, RL-vertex is given by v_{11}. The placement of the corresponding towers to cover the entire terrain are at vertices v_4,v_7 and v_{11}. A formal stepwise sketch of the resulting LCRM-heuristic is listed as Algorithm 4.

Algorithm 4 LCRM-heuristic

Input: (i) Terrain T_1, (ii) Tower height h_1
Output: Placement vertices W=v_{i_1},v_{i_2},...v_{i_m}

1: $W = \phi$
2: **while** $T_1 \neq \phi$ **do**
3: Determine RL-vertex u_{i_1} by checking visibility from the top of the tower placed at rightmost vertex starting at v_n
4: $W = W \cup \{u_{i_1}\}$
5: T_1= sub-chain of T_1 to the right of T_1
6: Output W

Chapter 4

Implementation and Experimental Results

In this section, we present the implementation details of our proposed algorithms for various visibility problems in 1.5D terrain. These include determining the position of a shortest watchtower to completely guard the terrain, placement of a watchtower of specific height, placement of two watchtowers to completely guard the terrain etc. The algorithms are implemented in Java while the GUI was developed using Java Swing API.

Classes for various elementary geometric objects like point, segment, line and ray are used from the custom BasicGeometry library. This library provides all the basic functionality like length of a segment, col-linearity of points, intersection among line segments, etc, that we require to implement the algorithms. Details of these used methods will be presented later.

4.1 Interface Description

The frontend interface of the application is a GUI window developed using Java Swing API. The layout used to place the elements in the window is the GridBagLayout which works by creating a dynamic grid where each GUI element is placed in each cell. The application window is just a JFrame object in Java Swing. The GUI comprises of three main panels: the top panel, the left panel and the right panel. (Figure 4.1)

The top panel is for the menu bar. It contains a File Menu which has in turn Open, Save and Exit options. It can be used to (i) open a saved terrain file, (ii) save the current file, and (iii) exit the application. The left panel contains all the control elements that can be used to draw, edit or

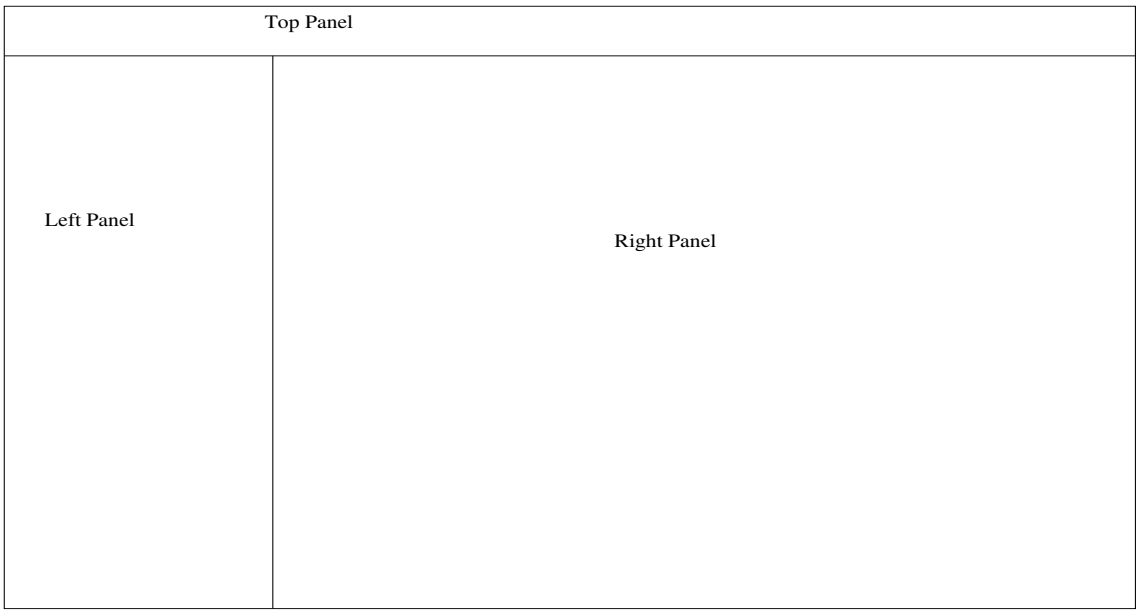


Figure 4.1: GUI layout of the application

clear the terrain. This panel contains buttons, checkboxes and text areas to display the coordinates of generated points. Each of these will be discussed in detail shortly. The right panel is used to draw the terrain. It is the main display area of the application.

The interface allows (i) drawing the vertices of a terrain, (ii) loading the previously saved points from a file,(iii) saving the current terrain vertices,(iv) drawing the terrain from vertices, (v) editing the terrain structure, and (vi) finding the shortest watchtower for that terrain. Each of the UI elements are as indicated in the following table (Table 4.1).

S.N.	**File Menu Item**	**Function**
1	Open	Open a previously saved terrain point set
2	Save	Save the current terrain points to a file
3	Exit	Exit the application

Table 4.1: Description of File Menu Items

The file menu consists of three options: Open, Save and Exit. Through these, users can open a previously saved terrain point set and draw a terrain from it. The Save option can be used to save a current terrain to a file. As the name implies, exit is used to quit the application. Similarly, the left panel of the GUI consists of several buttons and checkboxes that facilitates the drawing and editing of the terrain, displaying the kernel of the terrain and drawing the shortest tower. The

Draw button displays the terrain from plotted points, the Clear button clears the currently drawn terrain. There are also options for adding or removing the points in the terrain and for editing the existing points. These features are enabled when the respective boxes are checked. There are three textareas for different purposes. The first one prints the coordinates of terrain points. The second text box displays the coordinates of the shortest tower, while the third one displays the coordinates of the terrain kernel. Each of these elements are described in tables 4.2-4.4.

A snapshot of the application is shown in figure 4.2 below:

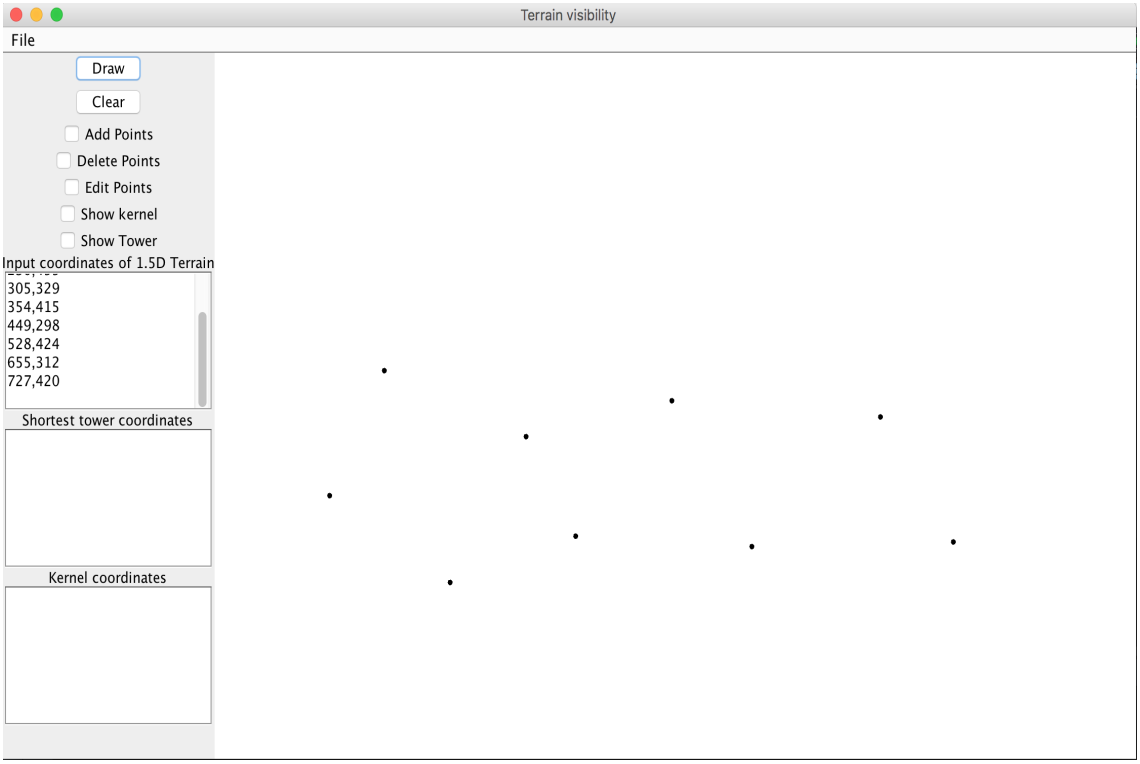


Figure 4.2: Snapshot of the GUI application

4.2 Component Functionality

S.N.	Buttons	Function
1	Draw	Draw the terrain joining the points plotted
2	Clear	Clear the drawing panel

Table 4.2: Description of Buttons Functionality

S.N.	CheckBoxes	Function
1	Add points	Add a new point to a terrain
2	Delete Points	Delete any point from a terrain
3	Edit Points	Edit the terrain point by moving it.
4	Show Kernel	Display the kernel of the currently drawn terrain.
5	Show tower	Display the shortest tower from which the terrain is completely visible

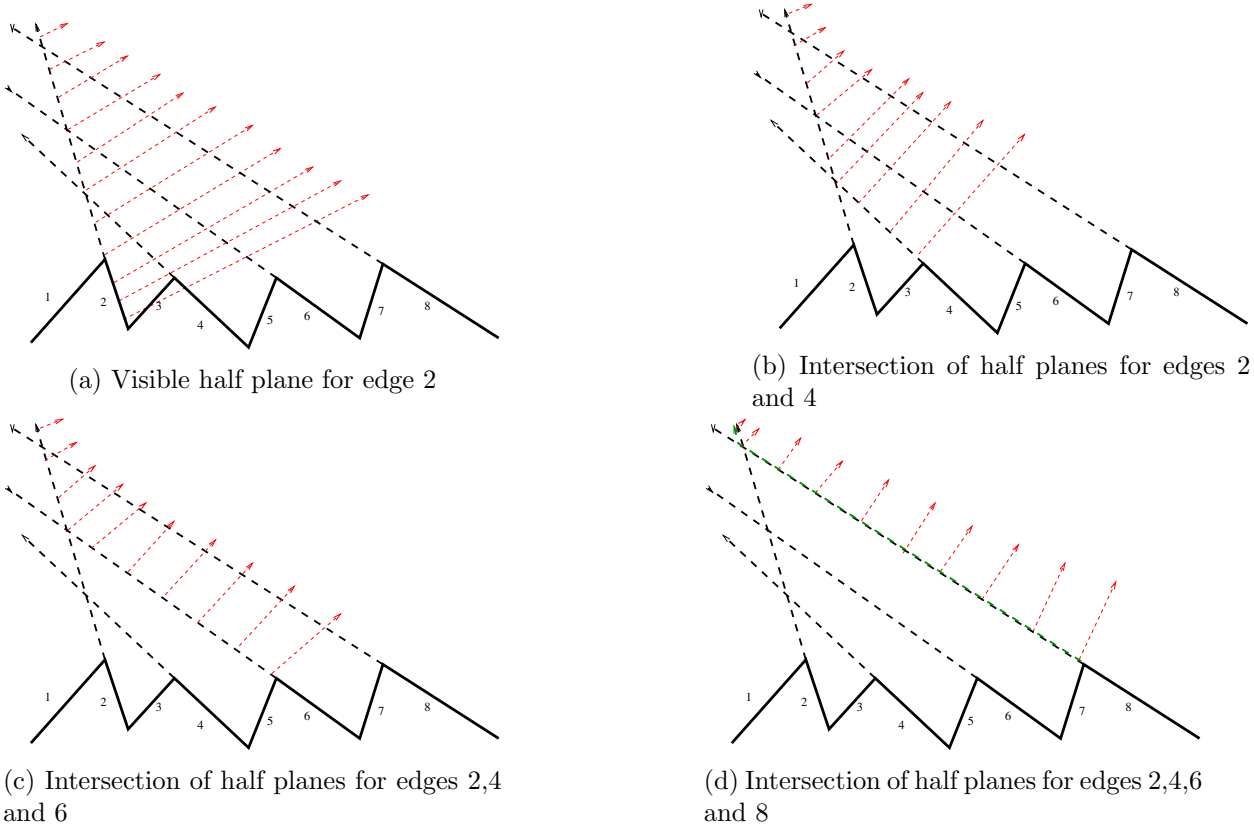
Table 4.3: Description of CheckBoxes Functionality

S.N.	TextArea	Function
1	Input Coordinates	Display the input coordinates of a terrain
2	Shortest Watchtower	Display the coordinates of shortest watchtower
2	Kernel Coordinates	Display the coordinates of a kernel

Table 4.4: Description of TextAreas Functionalities

4.3 Implementation of the Single Tower Placement Algorithm

The first algorithm we implemented was the algorithm to single tower placement problem. This algorithm is reported in [Sha88]. Here, the objective is to place the tower on a terrain such that all of the terrain is visible from the top of the tower. Finding the position for the shortest watchtower involves finding the convex region above the terrain. Any point inside this convex region may be the solution of our problem. That means, all of the terrain can be seen from any point on this convex region. We call this convex region a *kernel*". Finding the point on a kernel which is closest to the terrain is straightforward. Hence, the main sub-task while implementing the solution is determining the segments of the kernel.



(a) Visible half plane for edge 2

(b) Intersection of half planes for edges 2 and 4

(c) Intersection of half planes for edges 2,4 and 6

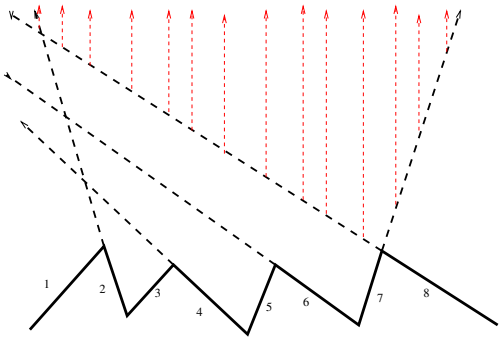
(d) Intersection of half planes for edges 2,4,6 and 8

Figure 4.3: Illustrating half plane of the edges in forward pass

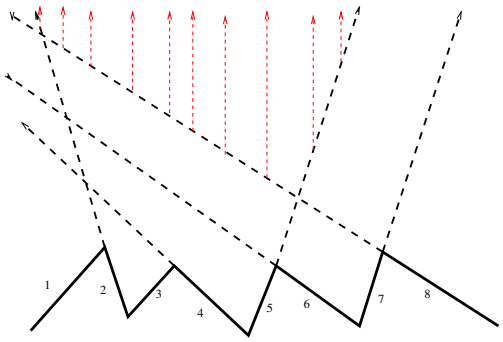
We begin by extrapolating the edges of a terrain. In the first (forward) pass, we take only the backward edges of the terrain and find the visible half plane for those edges. The idea is illustrated in figure 4.3. In the Figure, edges numbered 2,4,6 and 8 are considered in the first pass. In Figure 4.3a, the red dashed arrow lines depict the half plane for edge 2. Similarly, we compute half planes for the edges 4,6 and 8. Then we compute the intersection of these half planes which forms part of

the kernel.

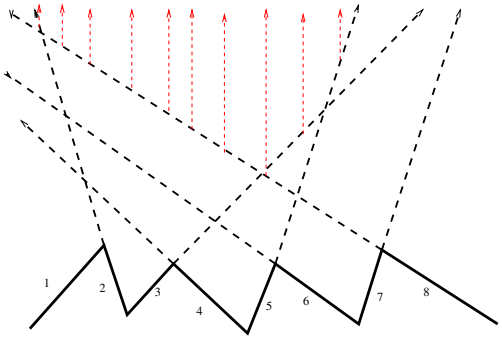
The segments of a kernel as the forward pass completes is shown as a green dashed line in Figure 4.3d. Now, we begin the backward pass where we start from edge 7 through 1 in backward fashion. In the backward pass, we compute the intersection of visible planes resulted from the forward pass with the visible half plane of each edge. At the end of the backward pass, we get the desired convex region which we call a kernel of the terrain. The backward pass is illustrated in Figure 4.4. In Figure 4.4d, the segments of the kernel are shown in green dashed lines.



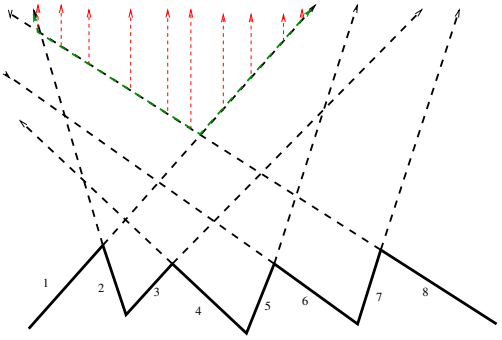
(a) Visible unbounded region until edge 7



(b) Visible unbounded region until edge 5



(c) Visible unbounded region until edge 3



(d) Visible unbounded region until edge 1

Figure 4.4: Illustrating the formation of kernel at the end of backward pass

After we obtain the segments for the kernel, all we have to do is find the point lying on the kernel whose perpendicular distance to the terrain is the lowest. A Screenshot of our program finding the shortest tower is given in Figure 4.5.

In our implementation, we store the edges of a kernel as forward feasible segments and backward feasible segments computed from forward pass and backward pass. Portion of the method *ComputeFeasiblePoints* which store these segments is as shown in code listing below:

```
1  List<segment> feasibleSegments = new ArrayList<>();
2  for (int k = 0; k <= i; k++) {
3      for (segment s : feasibleSegmentsPool) {
```

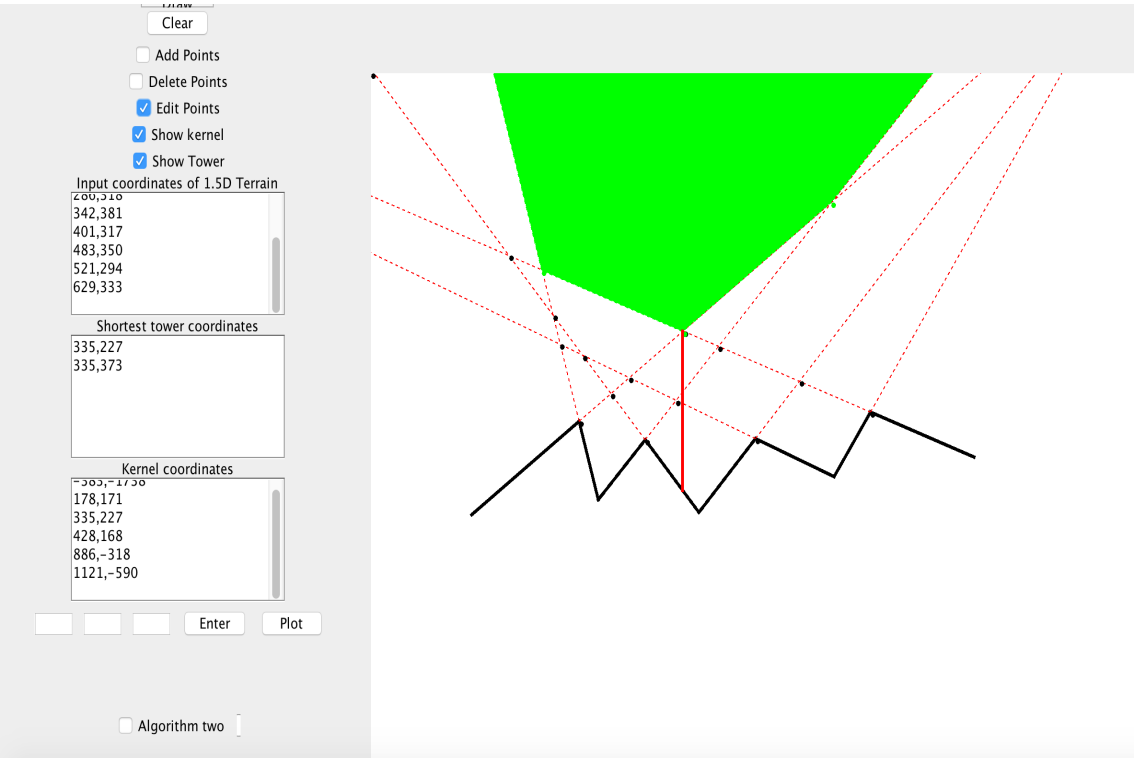


Figure 4.5: Finding the shortest tower

```
        line fwl = new line(forwardSegments.get(k));

        if(fwl.PerpDistanceToPoint(s.source())<collinearTol &&
        fwl.PerpDistanceToPoint(s.target())<collinearTol) {
        feasibleSegments.add(s);
        }
    }
}


for (int k = 0; k <= i; k++) {
    for (segment s : feasibleSegmentsPool) {
        line bwl = new line(backwardSegments.get(k));
        if(bwl.PerpDistanceToPoint(s.source())<collinearTol &&
        bwl.PerpDistanceToPoint(s.target())<collinearTol)
        {
            feasibleSegments.add(s);
        }
    }
}
```

4.4 Implementation of the Maximum Visibility Problem

In this sub-section, we describe the implementation details of the Maximum Visibility problem to illuminate the terrain. The theoretical ingredients of maximum visibility were presented in Chapter 3. Here, we describe how we implemented these ingredients. As with the previous implementation, we used the same interface for drawing the terrain. The GUI has an option (check box) to select the display of the placement of the tower for maximum visibility . The user can also input the height of the tower in number of pixel units.

The implementation displays both the 1.5D terrain and its generated image lifted up to the height of the tower. A snapshot of the display is shown in Figure 4.6.

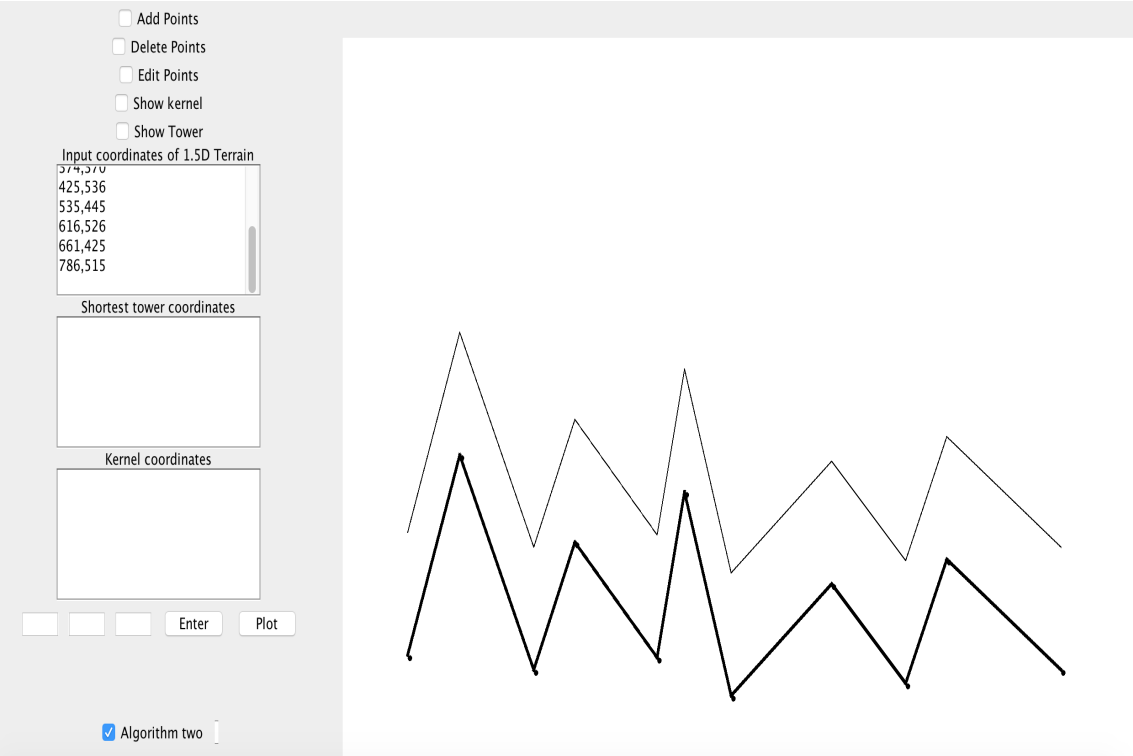


Figure 4.6: Image of the terrain

The edges of the terrain incident on convex vertices are extended to determine the guiding points. For our implementation, we adopted the straightforward method of checking the intersection between *grazing rays* formed by extending terrain edges incident on convex vertices and the terrain image. In Figure 4.7, grazing rays are drawn as dotted lines. The intersection points between grazing rays and the image of the terrain give guiding points, which are drawn as red dots in Figure 4.7.

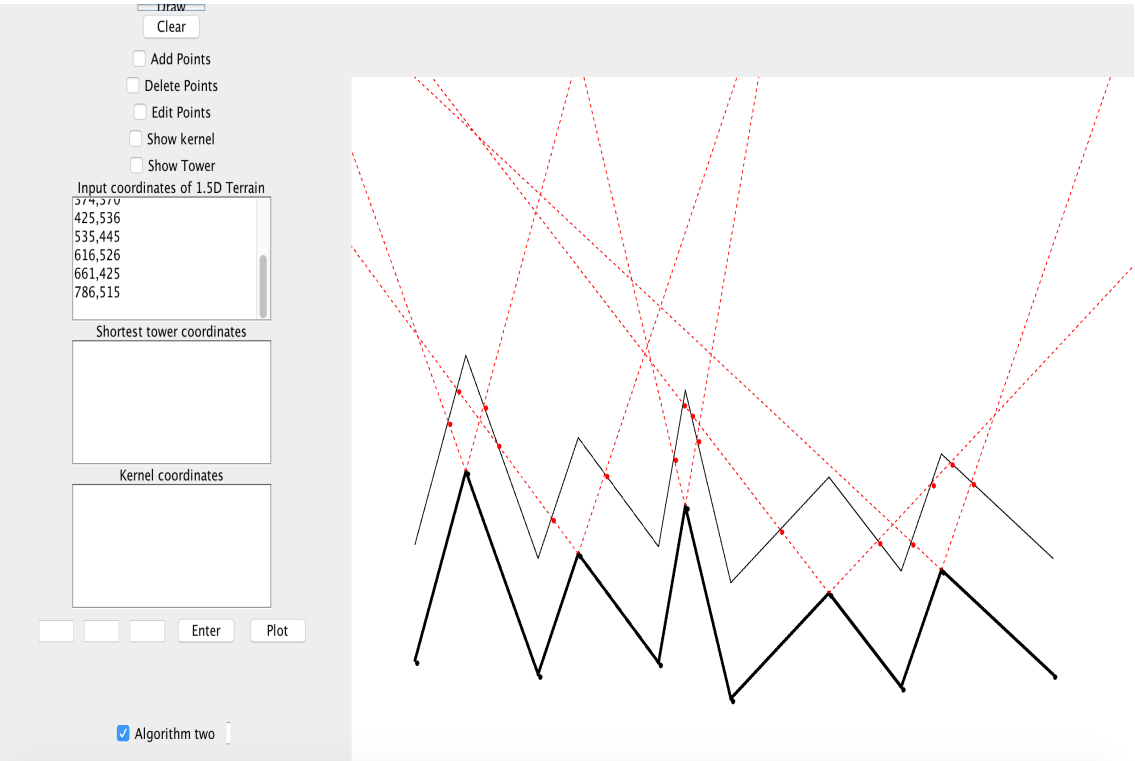


Figure 4.7: Our program finding the guiding points

In our implementation, we have defined a Java method called *DrawImageAndIntersection* that computes the line segments for terrain image and guiding points. The portion of the code that does this task is listed below:

```
for(line l:lines)
{
        my_point start=new my_point(l.getStart().get_x(),
                l.getStart().get_y()-limitedHeightTower);
        my_point end=new my_point(l.getEnd().get_x(),
                 l.getEnd().get_y()-limitedHeightTower);
         line il=new line(start,end);
         imageLines.add(il);
}

for(line il:imageLines)
 {
         for(segment s:extrapolatedLines)
        {
                segment s1=new segment(il.getStart(),
                il.getEnd());
                 if(s.Intersect(s1))
                           intersectionPointsWithImage.add
```

```
19                                      (s.Compute_Intersect(s1));
20              }
21      }
```

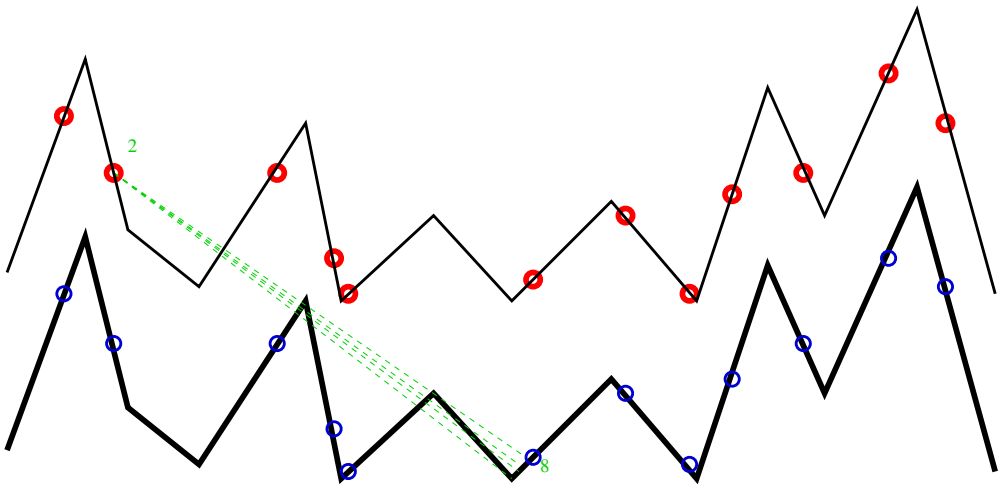
Here, the first **for** loop iterates over each edge of the terrain, retrieves the start and end coordinates of the edge and then finds the end coordinates of the image by simple subtracting the y-coordinates of the terrain edges from the height of the tower. This is how the image of the terrain is formed. After computing the edges of the image terrain, the second **for** loop iterates to check the intersection between grazing rays and image edges. The grazing rays are stored in a **List** data structure called **extrapolatedLines**. For computing the intersection points, the method **Compute_Intersect** provided by the class *segment* is used. The computed guiding points are stored in a **List** data structure named **intersectionPointsWithImage**.

The y co-ordinates of the guiding points are modified by adding the height of the tower to obtain the transition points on the terrain. We need to compute the **visible portions** of the terrain from each guiding point. What we exactly need is the **visibility segments** emanating from the guiding points to the terrain vertices. To compute a visibility segment, we treat image edges as transparent and terrain edges as opaque.

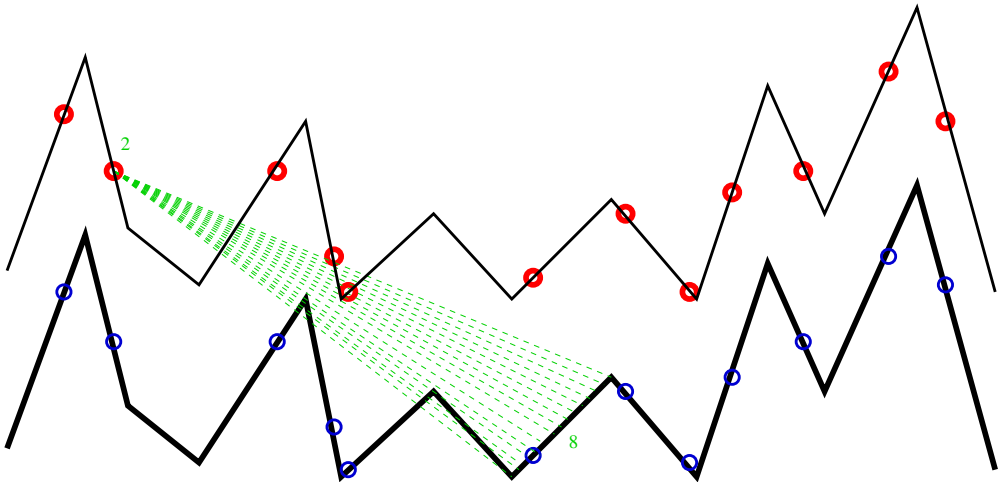
To find the visible portion of the terrain image, we need to establish the concept of whether the line of sight from the top of the tower to any specific point on the edges are blocked. If the line of sight is blocked, we say that point on the terrain edge is not visible from the top of the tower, else it is visible. After we computationally establish this concept, we just go through all the points on the edges and calculate the portion of the edges that are visible. This concept can be best illustrated through Figure 4.8.

As we see in the figure, if we place the tower on the second guiding point, line of sight from that point to the eighth edge is blocked by other edges until some higher point. After we gradually go upwards on the eighth edge, there comes a point when the line of sight starts reaching that edge unblocked. And it will remain so until the end of the edge. Therefore, placement of the tower at this guiding point means the visible portion of the eighth edge is from the point when the line of sight starts getting unblocked to the end of the edge. So for each of the transition points (guiding points), we compute the length of the visible portion for every terrain edge and sum them.

Another issue of this implementation is how to computationally determine whether the line of sight is being blocked. For this, we use a simple idea: check the intersection between the line of sight and all of the edges of the terrain except the edge we are trying to illuminate. If any one



(a) Line of Sight being blocked



(b) Line of sight starts getting unblocked

Figure 4.8: Illustrating how line of sight is used to determine visibility

of those edges has an intersection with the line of sight, then it is blocked; otherwise it is not. In our code, we have implemented a method *ComputeVisibleEdgeDistance* which returns the length of an edge that is visible from any particular point. To gradually change the line of sight as we progress through the edge, we compute the slope and intercept of the edge, and find the next point that lies on the edge which connects to the guiding points. The excerpt of the code that does this is given below:

```
1  public double ComputeVisibleEdgeDistance(my_point p,int lineInd)
2  {
3              int height=600;
4              line l=lines.get(lineInd);
5              p=new my_point(p.get_x(),height-p.get_y());
6
```

```
            my_point start=new my_point(l.getStart().get_x(),
                    height-l.getStart().get_y());

            my_point end=new my_point(l.getEnd().get_x(),
                    height-l.getEnd().get_y());

            double slope=(double)(end.get_y()-start.get_y())/
                    (double)(end.get_x()-start.get_x());

            double intercept=(double)(end.get_x()*start.get_y()-
                    start.get_x()*end.get_y())/
                (double)(end.get_x()-start.get_x());

          for(int i=start.get_x()+2;i<end.get_x();i=i+2)
          {
                currentX = i;
                newY = (int) ((slope * i) + intercept);
                lineOfSight = new segment(p, new my_point(i, newY));
                -  -
                -  -
                -  -
           }
           -  -
           -  -
           -  -

}
```

As per our description above, we have created a method to check if a line of sight is blocked or not by an edge which is called repeatedly from method $ComputeVisibleEdgeDistance$. Method $LineOfSightBlocked$ is listed below:

```
private boolean LineOfSightBlocked(segment s,int height,int li)
{
            int no=0;
            for (int i=0;i<lines.size();i++)
             {
                line edge=lines.get(i);

                my_point sStart=new my_point(edge.getStart().get_x(),
                                height-edge.getStart().get_y());

                my_point sEnd=new my_point(edge.getEnd().get_x(),
                                height-edge.getEnd().get_y());
```

```
13
14                  segment sEdge = new segment(sStart,sEnd);
15                  if(i!=li)
16                   {
17                              if (sEdge.Intersect(lineOfSight))
18                                          no++;
19                   }
20
21              }
22              //return no;
23              if(no==0)
24                  return false;
25              else
26                  return true;
27  }
```

We can see from the code, the *for* loop iterates over all the edges and checks if it intersects with lineOfSight. If it does, we increment the variable *no* by 1. At the end, if *no* is 0, i.e. no edges intersects with lineOfSight, it returns false(meaning it is not blocked) else it returns true(meaning it is blocked). Finally, our program identifies the position where the visibility of the terrain is maximum. It is shown in the following Figure:

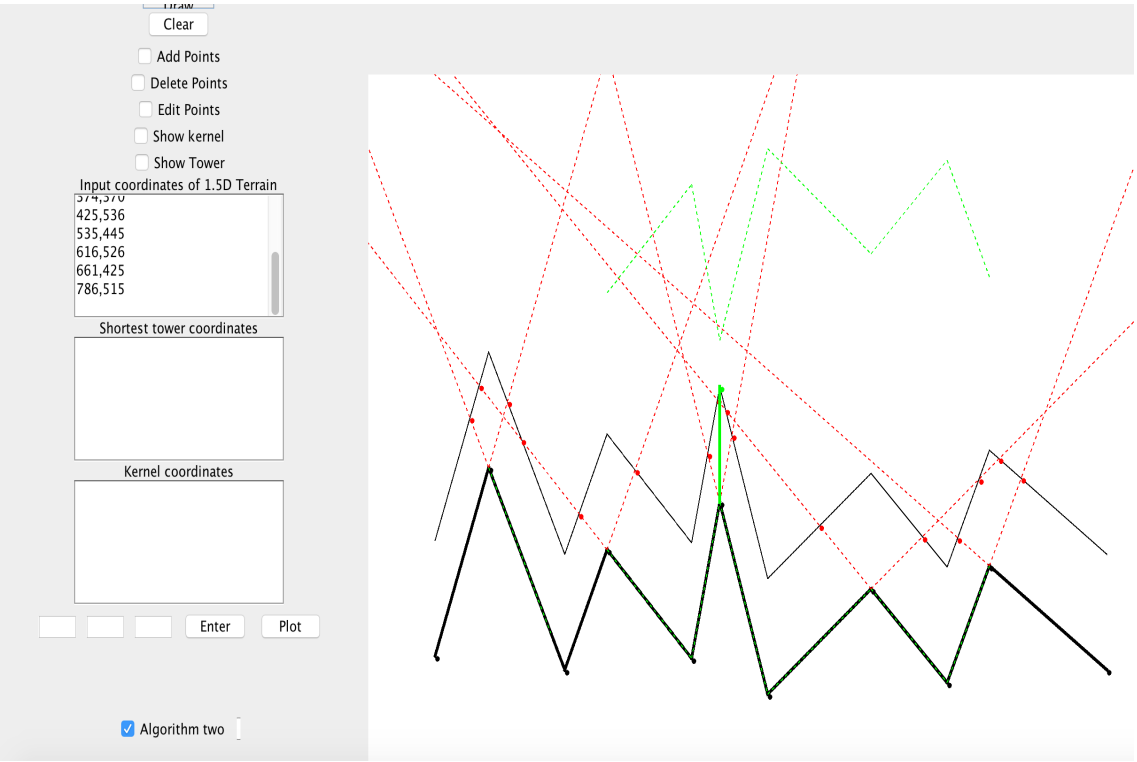


Figure 4.9: Our program finding the solution for maximum visibility

Chapter 5

Conclusion

We presented a cursory review of existing algorithms for placing towers in 1.5D and 2.5D terrain. We proposed two variations of tower placement algorithms.The first version of the problem asks for placing a single tower of a given length to maximize the visibility coverage. In the second version of the problem, we proposed the placement of a reduced number of guards to cover the entire 1.5D terrain. While the first problem is an optimization problem, the second one is a heuristic for obtaining an approximate solution for a NP-Hard problem. Algorithms for solving both problems are based on discretizing the placement points on the terrain by computing transition points.

It was observed in Chapter 3 that the number of transition points could be quadratic in the number of vertices in the terrain(Figure 3.3). Not all transition points, as mentioned in Chapter 3, are necessary to search for the optimum placement. So, it would be interesting to reduce the number of transition points to make the proposed algorithms efficient. It would also be interesting to characterize 1.5D terrain for which the number of transition points is linear in the number of vertices of the terrain. The proposed algorithms are for 1.5D terrain. It would be a valuable exercise to extend our proposed algorithms to 2.5D terrain.

In our experimental investigation, we constructed the input 1.5D terrain manually by using mouse clicks in the interface of the prototype program. In order to evaluate the performance of the proposed algorithms rigorously, it would be appropriate to generate the 1.5D terrain randomly. The proposed algorithm could then be tested on several randomly generated 1.5D terrains.

Another variation of the tower placement problem is the positioning of two towers of common height to maximize the coverage. We can call this problem **2T-Max**. A solution for 2T-Max can be obtained by exploiting the structure of transition points formulated in Chapter 3. What we need is to check the coverage for all pairs of transition points and pick the one that maximizes the cover.

It would be interesting to solve 2T-Max efficiently without using all pairs of transition points.

Bibliography

[ABD+05] Pankaj K Agarwal, Sergey Bereg, Ovidiu Daescu, Haim Kaplan, Simeon Ntafos, and Binhai Zhu. Guarding a terrain by two watchtowers. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 346–355. ACM, 2005.

[BMKM07] Boaz Ben-Moshe, Matthew J Katz, and Joseph SB Mitchell. A constant-factor approximation algorithm for optimal 1.5 d terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[DBVKOS00] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[DK85] David P Dobkin and David G Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6(3):381–392, 1985.

[Eid02] Stephan Eidenbenz. Approximation algorithms for terrain guarding. *Information Processing Letters*, 82(2):99–105, 2002.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[KK11] James King and Erik Krohn. Terrain guarding is np-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

[O'R87] Joseph O'Rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[o'R98] Joseph o'Rourke. *Computational geometry in C*. Cambridge university press, 1998.

[Sha88] Micha Sharir. The shortest watchtower and related problems for polyhedral terrains. *Information Processing Letters*, 29(5):265–270, 1988.

[Zhu97] Binhai Zhu. Computing the shortest watchtower of a polyhedral terrain in o (nlogn) time. *Computational Geometry*, 8(4):181–193, 1997.

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Binay Dahal

dabinay@gmail.com

Degrees:

Master of Science in Computer Science 2018

University of Nevada Las Vegas

Thesis Title: Algorithms for Tower Placement on Terrain

Thesis Examination Committee:

Chairperson, Dr. Laxmi Gewali, Ph.D.

Committee Member, Dr. John Minor, Ph.D.

Committee Member, Dr. Kazem Taghva, Ph.D.

Graduate Faculty Representative, Dr. Henry Selvaraj, Ph.D.