5-1-2019

Approximation Algorithms for Illuminating 1.5D Terrain

Jiwan Khatiwada
nawijj@gmail.com

APPROXIMATION ALGORITHMS FOR ILLUMINATING 1.5D TERRAIN

By

Jiwan Khatiwada

Bachelor's Degree in Computer Engineering
Tribhuvan University
2014

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 2019

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 18, 2019

This thesis prepared by

Jiwan Khatiwada

entitled

Approximation Algorithms for Illuminating 1.5D 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

Kazem Taghva, Ph.D.
Examination Committee Member

John Minor, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Dean

Abstract

We review important algorithmic results for the coverage of 1.5D terrain by point guards. Finding the minimum number of point guards for covering 1.5D terrain is known to be NP-hard. We propose two approximation algorithms for covering 1.5D terrain by a fewer number of point guards. The first algorithm (Greedy Ranking Algorithm) is based on ranking vertices in term of number of visible edges from them. The second algorithm (Greedy Forward Marching Algorithm) works in greedy manner by scanning the terrain from left to right. Both algorithms are implemented in Python 2.7 programming language.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor Dr. Laxmi Gewali for his continuous guidance and help. His guidance helped me in all the time of research and writing of this thesis. 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 insightful comments and encouragement to improve my work.

I would like to thank my friends and family members for supporting me and for showing concern on my progress throughout writing this thesis.

Jiwan Khatiwada

University of Nevada, Las Vegas

May 2019

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Problems related to visibility on terrain surface have numerous applications such as (i) geographic data frameworks, (ii) route management for aerial vehicles, (iii) transportation systems, (iv) crisis reaction arranging, and (v) remote communications system (Wired and Wireless).

Terrain visibility problems can be viewed as a restricted instance of the well-known Art Gallery Problem [O'R98] in computational geometry. In the art gallery problem, the domain is a simple polygon and it is required to find the set S of a minimum number of point guards inside the polygon so that any point inside it is visible from some point in S. It is remarked that two points p_i and p_j inside a polygon are visible to each other if the line segment having p_i and p_j as endpoints does not intersect with the exterior of the polygon. The standard art gallery problem is known as NP-hard [O'R98]. This intractability result has motivated many researchers to look for approximation algorithms for art gallery problem [O'R98]. Some variation of the standard art gallery problem has been considered. Such variations include an alternative notion of visibility and having input polygon restricted to monotone polygons and orthogonal polygons. In visibility variations, the notion of staircase visibility [Gew95, VP12] . In the staircase visibility model, two points p_i and p_j inside the polygon are visible if there is a staircase path connecting p_i and p_j that lies completely inside the polygon. It is noted that in a staircase path the edges are parallel to x-axis and y-axis and the path itself is monotone.

In this thesis, we use the standard notion of visibility and restrict the polygon domain as a monotone polygon in which one chain is a monotone chain and the other chain is a line segment. A monotone polygon with one chain as line segment is precisely a 1.5D terrain. The standard terrain is a 2.5D structure which means that a terrain is a structure which is between two dimensions and three dimensions. This view can be further elaborated in term of the cross-section of terrain with a horizontal plane. If we consider the cross-section of a terrain with a horizontal plane then the cross-section area become progressively smaller as the height of the horizontal plane increases.

The thesis is organized as follows. In Chapter 2, we review important existing algorithms dealing with visibility property of simple polygon and 1.5D terrain. In particular, we examine existing algorithmic results for placing guards to cover 1.5D terrains. We also review intractability results and approximation algorithms for placing point guards in a monotone polygon and 1.5D terrain. In Chapter 3, we present the principle contributions of this thesis. We design, describe and sketch two approximation algorithms for finding a reduced number of point guards to cover (or illuminate) a 1.5D terrain. The first algorithm which we call "Greedy Ranking" is based on the ranking of vertices on visibility measures. The nodes are then processed in a greedy manner by placing the first point guard at the node with the largest visibility and other guards are progressively placed by re-ranking the uncovered nodes. The time complexity of the algorithm is $0(|E|log|V|)$ where $|E|$ is the number of edges and $|V|$ is the number of vertices in visibility graph induced by the terrain. The second algorithm (Greedy Forward Marching Algorithm) places guards in greedy manner, by placing candidate guard g_i as far right as possible to illuminate uncovered terrain to the left of g_i. The time complexity of this algorithm is $O(|E| + |V|)$.

In Chapter 4, we describe in detail the implementation of the proposed algorithms. The implementation is done by using Python 2.7 and the prototype program contains a friendly user interface. We also describe the process of generating input data for terrain, both randomly and by user-input. We then discuss the results obtained for many sets of data (more than 100 sets). Finally, in Chapter 5, we discuss (i) possible extension of the proposed algorithms and (ii) interesting variation of terrain illumination problem for future research.

Chapter 2

Visibility Algorithms on Polygon and 1.5D Terrain

In this part, we presented a brief review of some problem in computational geometry that is closely related to our research topic and reviewed some algorithms for illuminating polygon and 1.5D terrain by placing point guards.

2.1 Preliminaries

Problem dealing with visibility in the presence of polygons has been investigated by several researchers since last 40 years [O'R87, DBVKOS00]. In defining the notion of the visibility, the boundary of a polygon is considered as an opaque object. Two points inside the polygon are visible if the line segment connecting them do not intersect with the boundary. This is illustrated in Figure 2.1.

In Figure 2.1, point p is visible to the point v as the line segment connecting p to v does not intersect with any boundary edge of the polygon. On the other hand, point p is not visible to point q as the line segment connecting them intersect with the boundary edges. This notion of the internal visibility carries over naturally to the exterior of the polygon. In the figure, three exterior points (r, t, s) are shown. It is clear that r is visible to t but not visible to s.

One of the widely investigated visibility problems on the simple polygon is to illuminate the entire interior of the polygon by placing a minimum number of point guards inside the polygon boundary. This is often known as the *Art Gallery* problem. An Art Gallery problem instance is shown in Figure 2.2. For this polygon, two vertex guards are necessary and enough to guard the

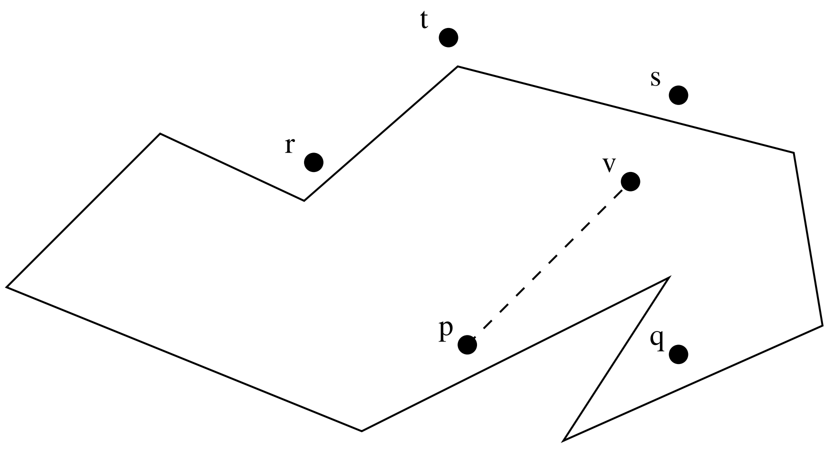


Figure 2.1: A Polygon

entire polygon. Interested readers can find such problem in [O'R87, DBVKOS00].

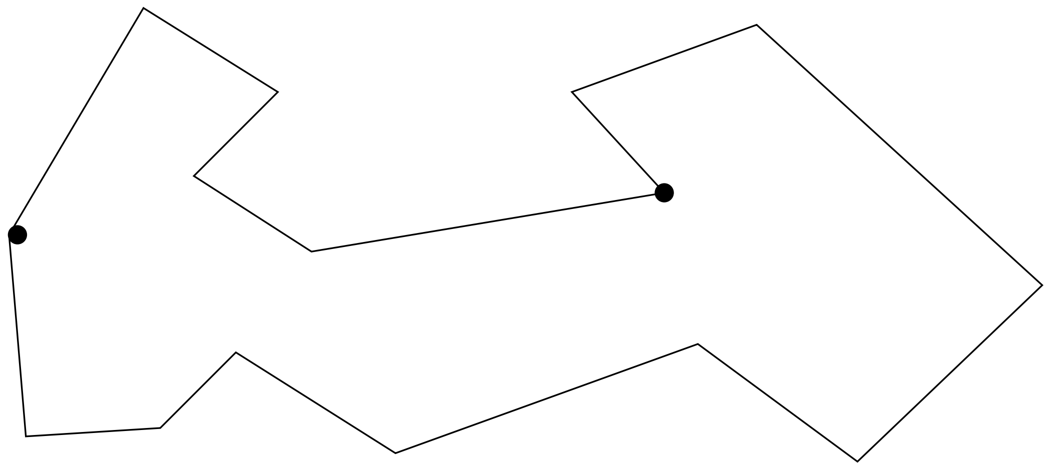


Figure 2.2: An instance of Art Gallery Problem

The problem of placing minimum number of guards inside a simple polygon is known to be intractable [LL86]. This problem remains NP-hard even for some restricted classes of polygons. One of the widely studied restricted class of simple polygons are monotone polygons. A simple polygon is called monotone if its boundary can be partitioned into two chains, each of which are monotone with respect to a given direction. Monotone polygons are used to model two dimensional terrain. Finding the minimum number of point guards to cover terrain has applications in telecommunication tower placement and geographic information system. An instance of the placement of point guards on a 1.5D terrain is shown in Figure 2.3. In this problem instance, five point guards are needed.The point guards are drawn as small circles. Readers can easily verify that the terrain cannot be illuminated (or covered) with less than 5 point guards. The problem of finding the minimum number of point guards in terrain was a long standing open problem, which was settled by James King and Erik Krohn in 2009. They proved [KK11] this problem to be NP-hard.

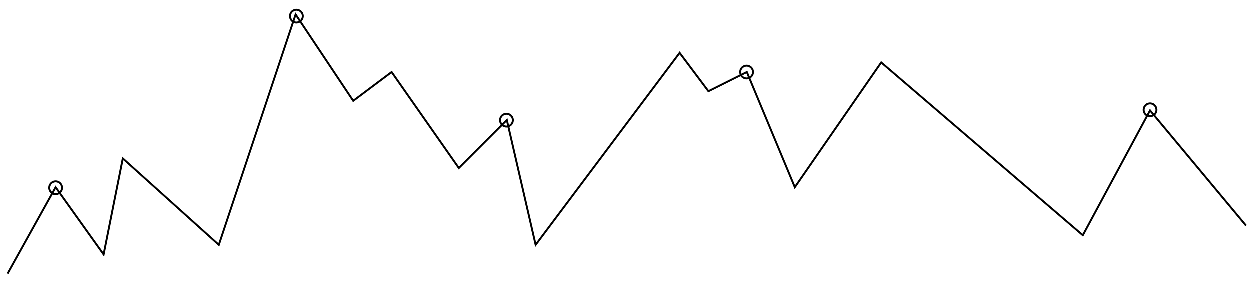


Figure 2.3: An instance of Terrain Illumination

They reduced an instance of PLANAR 3-SAT problem to an instance of minimum guard placement problem in the monotone polygon. PLANAR 3-SAT problem is a restricted version of the standard 3-SAT problem [GJ02]. In a planar 3-SAT, the graph implied by the satisfiability expression has to be a planar graph. Both standard 3-SAT and PLANAR 3-SAT problem are known to be NP-Hard [GJ02].

2.2 One-sided versus two-sided guarding

The standard terrain or terrain guarding problem is the one-sided guarding problem. In the definition of one-sided guarding problem, a point p_i in the domain is said to be guarded if p_i is visible from any guard.

Very recently [LH18], the notion of two-sided guarding problem has been introduced in the context of guarding 1.5D terrain. In this definition, a point p_i in the terrain is said to be two-sided guarded if p_i is visible to at least one guard in the left and at least one guard in the right. The distinction of one-sided guarding and two sided guarding is shown in Figure 2.4.

An examination of the terrain in Figure 2.4 shows that it needs 5 guards (X) to cover it under

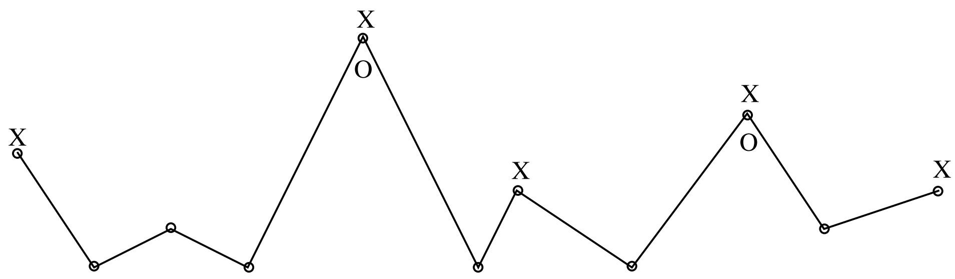


Figure 2.4: Distinguishing one-sided and two-sided guarding

two-sided notion of visibility. This terrain can be guarded by 2 guards (shown by o) under normal (one-sided) notion of visibility. While finding the minimum number of guards to cover 1.5D terrain is NP-Hard [KK11], the problem can be solved in linear time [LH18] under the notion of two-sided visibility.

2.3 Approximate Algorithm

The intractability of the art gallery problem has motivated many authors to develop approximation algorithms. One of the first such algorithm was proposed by S.K Ghosh [Gho10]. This paper contains approximation algorithms for both simple polygons and polygon with holes. It is established in [Gho10] that a simple polygon with n vertices can be guarded with numbers of guards m such that m is no more than O(log n) time the optimal solution. Their algorithm is based on partitioning the polygon into convex components. The convex components are views as sets and approximation set covering algorithm is used to obtain approximation solution for art gallery problem. The time complexity of the algorithm is $O(n^4)$ for the simple polygon and $O(n^5)$ for the polygon with holes.

Ben-Moshe et. al. [BMKM05] have reported an approximation algorithm for guarding 1.5D terrain by point guards. The idea is to process **pockets** of the terrain separately by evaluating the visibility of convex vertices in each packet. The **pockets** are formed when 1.5D terrain is enclosed by the convex hull bounded as shown in the Figure 2.5.

In the figure, convex hull boundary (Shown by dashed edges) and 1.5D terrain induce three pockets. The authors [BMKM05] made complicated case analysis to develop an approximation algorithm of constant factor for covering the terrain. The time complexity of the algorithm is $O(n^2)$ where n is the number of vertices in the terrain. In this algorithm it is not clear what is the

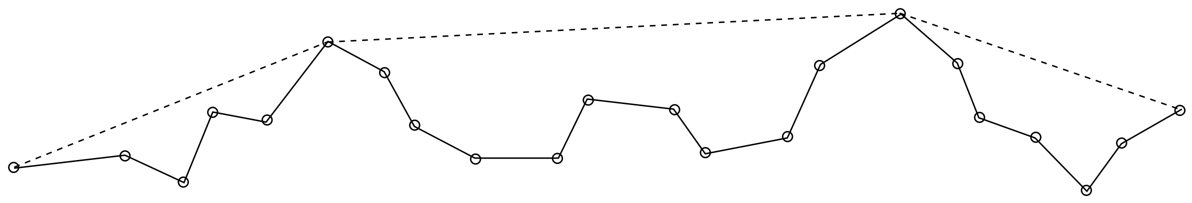


Figure 2.5: Formation of pockets

exact value or bound of the constant factor

Chapter 3

Fast Heuristic For Covering 1.5D Terrain

In this chapter we propose two heuristic algorithms for placing a reduced number of point guards to cover a given 1.5D terrain.

3.1 Preliminaries

We start with the description of the properties and characterization of 1.5D terrain needed for developing fast heuristics. Since the problem of placing a minimum number of point guards in 1.5D terrain is NP-Hard [KK11] it is motivating to come up with good heuristic methods that execute relatively fast in practical applications. Due to simpler structural properties of 1.5D terrain, guard placement is relatively easier compared to guard placement in a simple polygon.

Observation 3.1

For simple polygons, it is known that visibility of all vertices does not imply that all of the boundary is visible [O'R98]. This fact applies to 1.5D terrain as shown in Figure 3.1.

In the figure, two guards are placed at v_1 and v_5 shown by the solid circles. All the remaining vertices v_2, v_3, v_4, v_6 and v_7 are visible from these two guards but the edge $< v_3, v_4 >$ is not visible.

Observation 3.2

A guard placed at a non-convex vertex can be moved to the adjacent convex vertex without losing any coverage.

Definition 3.1: (Zig-Zag Terrain) A 1.5D terrain in which no two consecutive vertices are convex.

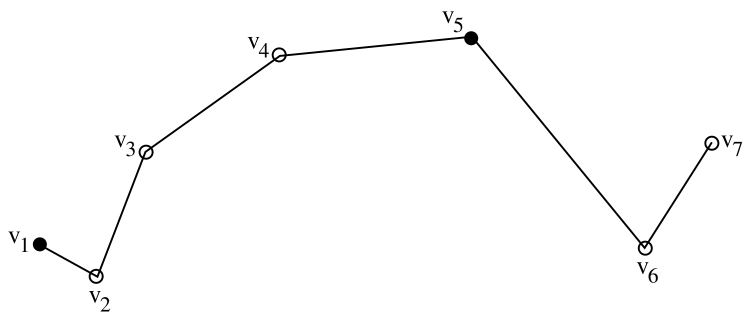


Figure 3.1: Illustrating Observation 3.1

Lemma 3.1 *In a Zig-Zag 1.5D terrain, covering all vertices implies the covering of all boundary points.*

Proof: Suppose there is an edge $e_i = (v_{i-1}, v_i)$ which is not completely visible. If a point guard is at v_{i-1}, v_i, or v_{i+1} then e_i is clearly visible. If v_i is visible from vertex v_j (other than v_{i-1},v_i,v_{i+1}) then v_j must be in the sector R_j formed by v_{i-1}, v_i, v_{i+1} as shown by dashed rays in Figure 3.2. Consequently, points of e_i are visible from that guard due to the convexity of the sector R_i.

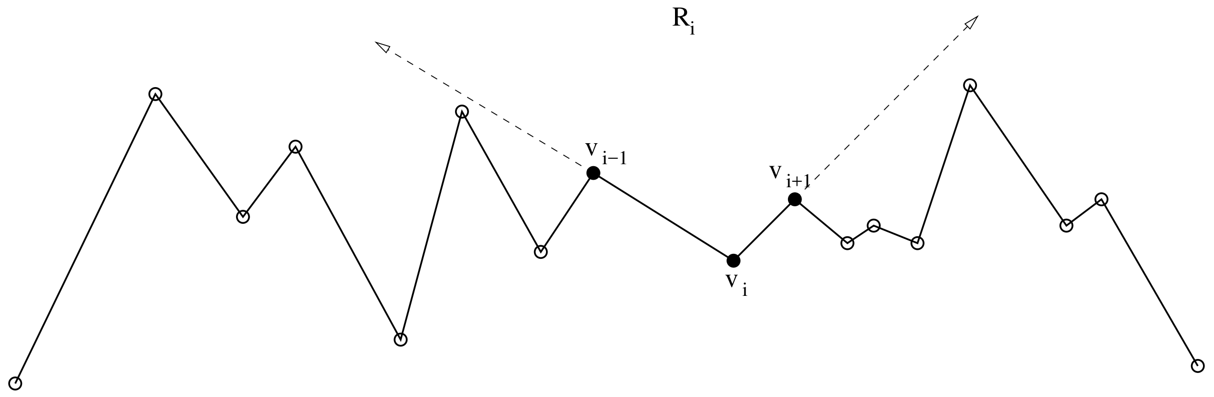
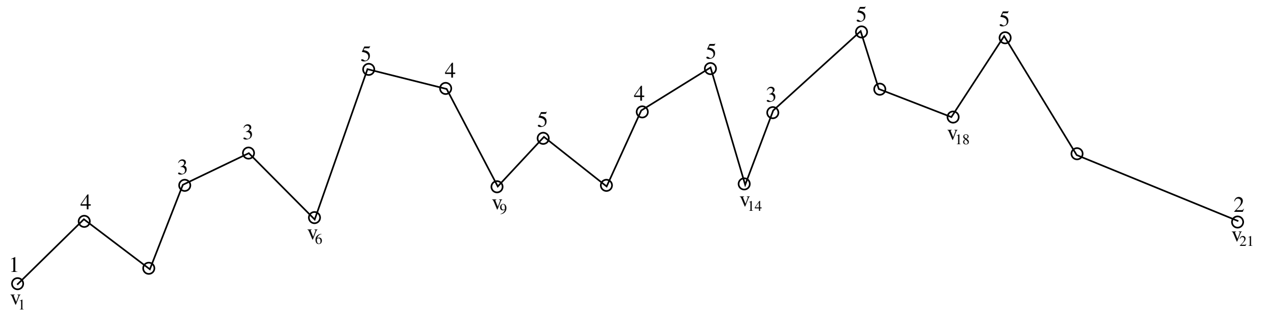


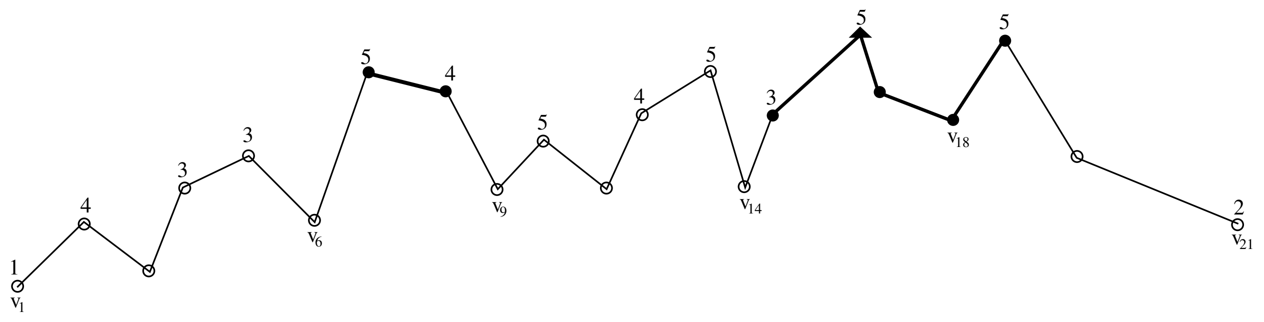
Figure 3.2: Illustrating proof of Lemma 3.1

3.2 Greedy Ranking Algorithm

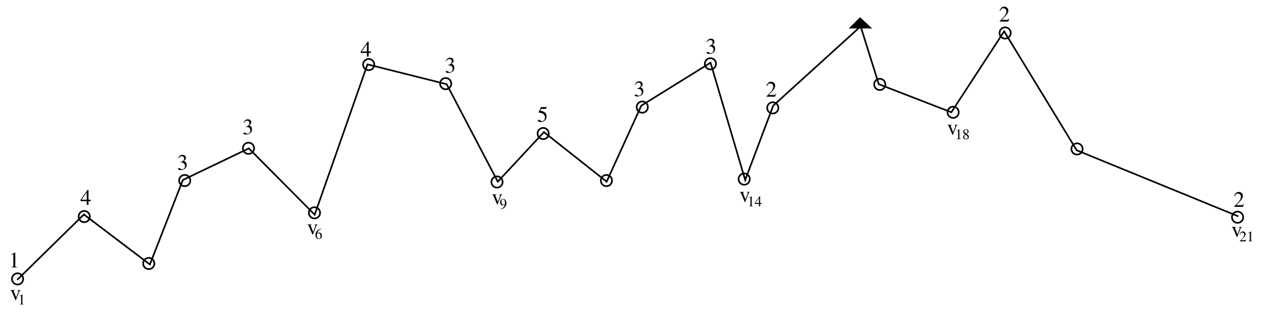
This algorithm works in two stages: (i) greedy placement stage and (ii) redundancy removal stage. In the first stage, the next point guard is placed on the vertex that covers the maximum number of non-illuminated edges.



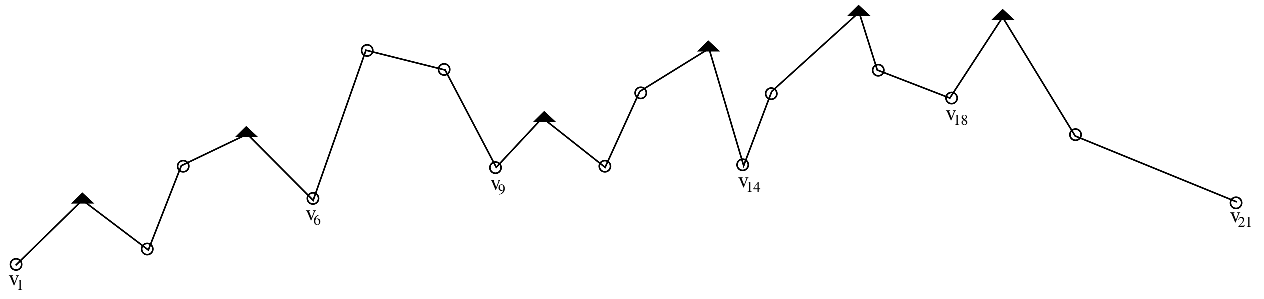
(a) Illustrating Initial Node Ranking.



(b) Illumination state after placing the first guard



(c) Ranking of vertices after placing first guard



(d) Guard placement after completing first stage

Figure 3.3: Illustration of Greedy Ranking Algorithm

The location of vertex guards are determined based on their rank. The **rank** of a node v_i is the number of non-covered edges that can be covered by placing a point guard at v_i. Initially all edges are not covered and the rank of each node is the number of edges visible from them. The initial rank of nodes are illustrated in Figure 3.3a.

We describe the working of the algorithm with this running example. Since vertex v_{10}, v_{13}, v_{16}, v_{19} have the highest rank (5), we pick the vertex v_{16} arbitrarily and place the first guard at v_{16} (shown by a triangle symbol). Note that if more than one vertex have the same rank then we pick the vertex arbitrarily.

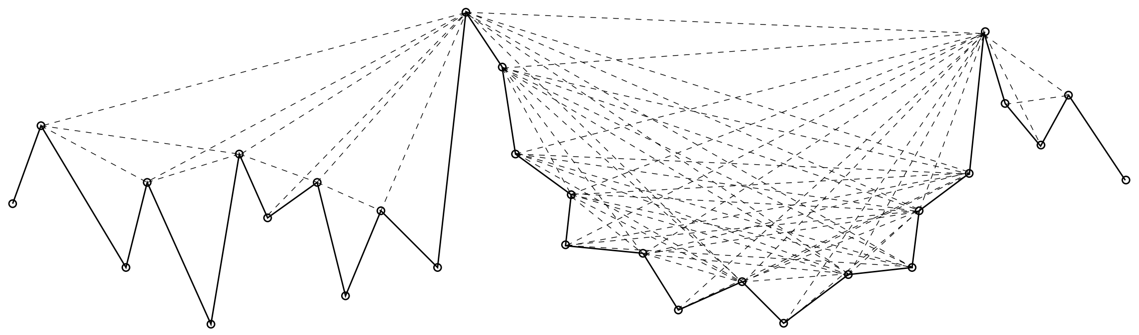
After the placement of the guard at v_{16}, all the edges visible from v_{16} are determined (visible vertices are drawn with filled circle and visible edges are drawn with thick line in Figure 3.3b). Based on this placement, the ranking of the nodes are recomputed. In recomputing the rank, only the uncovered edges are considered. The new ranks are shown in figure Figure 3.3c.

It is noted that for the vertices where guards are already placed, the rank is not needed and not shown in Figure 3.3c. In the second round of ranking, vertex $v_1 0$ has the highest rank and a guard is placed there. The process of re-ranking and guard placement is continued until all edges are covered. In our running example, the first stage is completed after 6 rounds of ranking. The placement of guards after completion of the first stage (greedy ranking) is shown in Figure 3.3d.

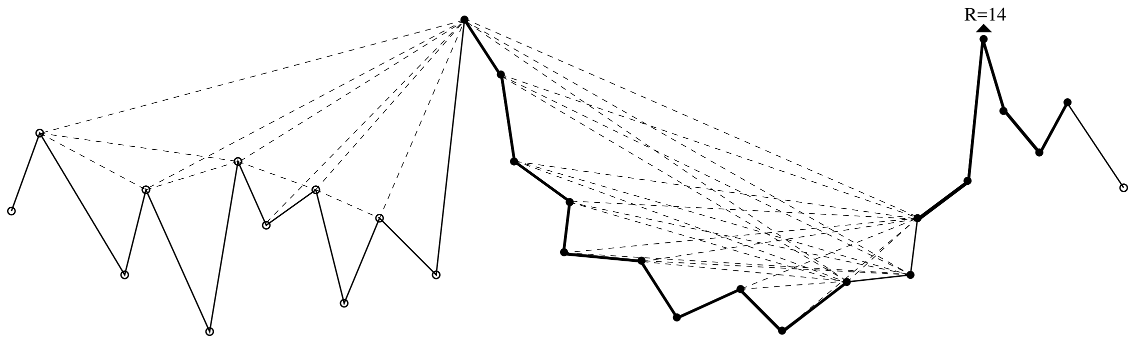
The greedy ranking algorithm places guards incrementally (one at a time) by identifying the vertex with highest rank. After each guard placement, the rank of the vertices are recomputed (updated) so that edges already covered are excluded in the ranking accumulation. The algorithm stops when all the edges are covered. A formal sketch of "Greedy Ranking Algorithm" is shown in Algorithm 3.1.

A straight forward implementation of the Algorithm 1 can take O(n^3) time in the worst case. Step 4 (ranking step) can be implemented by checking the intersection of possible edges with terrain edges which can take O(n^2) time in the worst case. To implement Step 7 (marking covered edges), we can similarly check the intersection of candidate visibility edges with edges of terrain, which again takes O(n^2) time. If the while loop repeats **n** times then the total time complexity of the algorithm is O(n^3). This time complexity is rather high. An improved implementation of Algorithm 1 based on the visibility graph is described next.

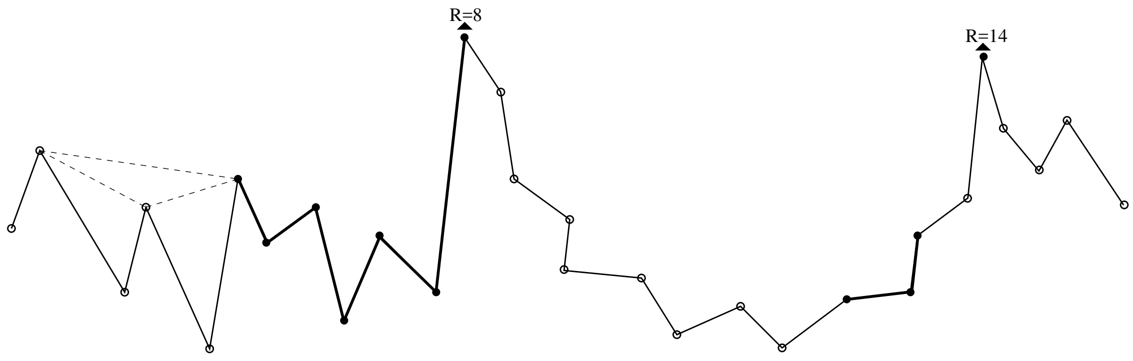
The visibility graph in the presence of a polygonal object is defined by considering the polygonal boundary as obstacles. The visibility graph in the presence of 1.5D terrain can be defined similarly and an example is shown in Figure 3.4a. A visibility graph can be computed in time proportional to



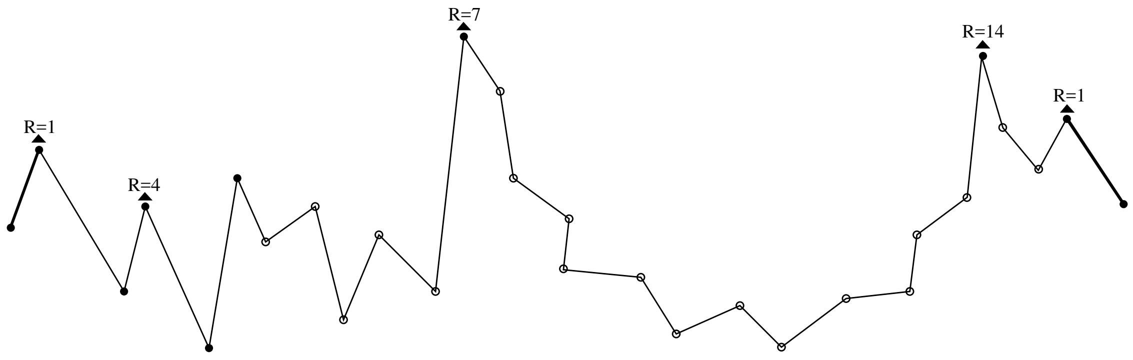
(a) Illustration of visibility graph.



(b) Illustration of visibility graph for uncovered edge after placing 1st guard



(c) Illustration of visibility graph for uncovered edge after placing 1st guard



(d) Illustration of visibility graph for uncovered edge after placing 1st guard

Figure 3.4: Illustration of Visibility graph

Algorithm 1 Greedy Ranking Algorithm

```
1: Input: Terrain chain $Ch_1$ $\{v_0,v_1,.....v_{n-1}\}$
2: Output: Subset $S_g$ of $Ch_1$ where guards are placed
3: while all edges are not covered do
4:     RankVertices($Ch_1$ $\{v_0,v_1,.....v_{n-1}\}$)
5:     u = getHighestRankedVertex($Ch_1$)
6:     u.guard = True
7:     Mark edges covered by u
8: $S_g = \{v \mid v.guard = True\}$
9: Output $S_g$
```

```
function RankVertices(chain $Ch_1$ $\{v_0,v_1,.....v_{n-1}\}$)
    for all $v$ in $Ch_1$ do
        if $v$ is convex and no guard placed at $v$ then
            $v$.rank = Number of newly covered edges
```

its size by using an algorithm given in [GM87]. When the visibility graph is computed in [GM87], visibility edges emanating from a vertex are available in angularly sorted order around it. This structure of the output of the algorithm in [GM87] can be used to implement the Greedy Ranking Algorithm efficiently. Initial ranking of vertices can be obtained by simply reading off the number of visibility edges emanating from each vertex. When a guard is placed at a vertex we can define the notion of Uncovered Visibility Graph (UVG) by considering only those visibility edges that are connected to the uncover edges of the terrain. When the first guard is placed (indicated by filled triangle sign above the terrain vertex), the resulting visibility graph is shown in Figure 3.4b .

The efficient version of Algorithm 1 is based on updating UVG as guards are placed. Initially, UVG is given by the standard visibility graph. When a guard is placed at vertex v_i, visibility edges emanating from v_i and its incident vertices are deleted to update UVG. The updated rank of vertices are the counts of visibility edges corresponding to the updated UVG. In order to retrieve and update the ranks of nodes, the vertices of **UVG** are maintained in a priority queue Q, using the priority of the number of visibility edges emanating vertices. A formal sketch of the algorithm is listed as Algorithm 2.

The time complexity of Algorithm 2 can be done as follows. Step 3 takes $O(|E| + |V|)$ [GM87]. One delete operation and decrease key operation can be done in $O(log|V|)$ time. Each execution of *while* loop removes at least one visibility edge to update UVG. Hence the whole loop executes at most $O|E|$ time. Thus the total time complexity is $O(|E|log|V|)$

Algorithm 2 Improved Greedy Ranking Algorithm

```
1: Input: Ordered list of vertices chain $V$ $\{v_0,v_1,.....v_{n-1}\}$ of terrain T
2: Output: Subset $S_g$ of $V$ where guards are placed
3: Compute Visibility Graph (VG) for T
4: Store Vertices of VG in max priority Queue Q in the priority of vertex degree
5: Set uncovered visibility graph UVG to VG
6: $S_g=\varnothing$
7: while UVG contains edges do
8:     v=Q.getMax()
9:     $S_g = S_g \cup \{v\}$
10:    Q.deleteMax()
11:    for all vertices u adj to v do
12:        decrease key of u by 1
13:        remove edge (u,v) from UVG
14: Output $S_g$
```

3.3 Removal of Redundant Guards

Once the terrain is fully guarded using the 'Greedy Ranking Algorithm', the next step is the identification and removal of redundant guards. Given three consecutive guards g_{i-1}, g_i and g_{i+1}, guard g_i is said to be **redundant** if the edges seen by g_i is a subset of the union of the edges seen by g_{i-1} and g_{i+1}. In our running example (Figure 3.3d obtained after stage1 processing) guard at v_{16} is redundant. An inspection of placement reveals that all the edges visible to v_{16} are also visible jointly by guards at v_{13} and v_{19}. The result of guard placement after removing redundant guards is shown in Figure 3.5. Incidentally, this is also the minimum number of guards needed to cover the running example 1.5D terrain.

3.4 Greedy Forward Marching Algorithm

The previous algorithm (the one presented in Section 3.2) attempts to place guards based on the visibility ranking of the nodes. In the Greedy Forward Marching Algorithm (GFM - Algorithm), we attempt to place the guards by scanning the terrain from left to right and identifying the nodes where a guard must be placed. This approach has been used for placement of towers in 1.5D terrain [GD18]. We adopt this approach for covering terrain by point guards. It is like covering terrain by zero height towers. We call such node a **Next Candidate Node**. Such a node can be defined as follows.

Definition 3.2 : (Next Candidate Node (NC)) The first guard is placed at the rightmost

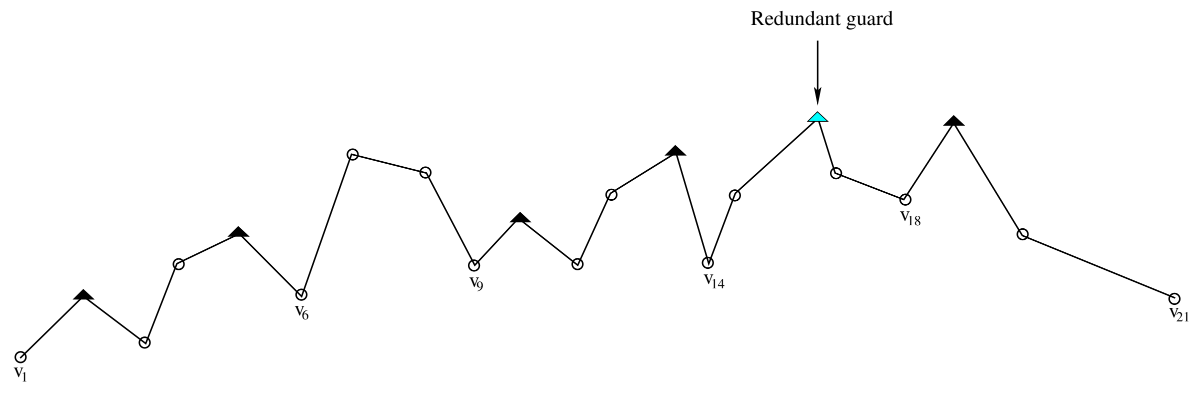


Figure 3.5: Final guard placement after applying redundancy removal.

vertex v_{i_1} such that all edges to the right of v_{i_1} are covered by the guard at v_{i_1}. Let v_{i_1}, v_{i_2},.....v_{i_k} denotes the guards placed so far. Then NC(v_{i_k}) denote the next candidate node such that it is the rightmost node that covers all the vertices (terrain surface) between v_i and NC(v_i). This is illustrated in the Figure 3.6. In this figure, two guards are currently placed which are at v_3 and v_7 (denoted by filled triangles). If we inspect the figure, we find that the next candidate node is v_{11}. The algorithm proceeds by marching left to right by identifying next candidate nodes in a greedy manner.

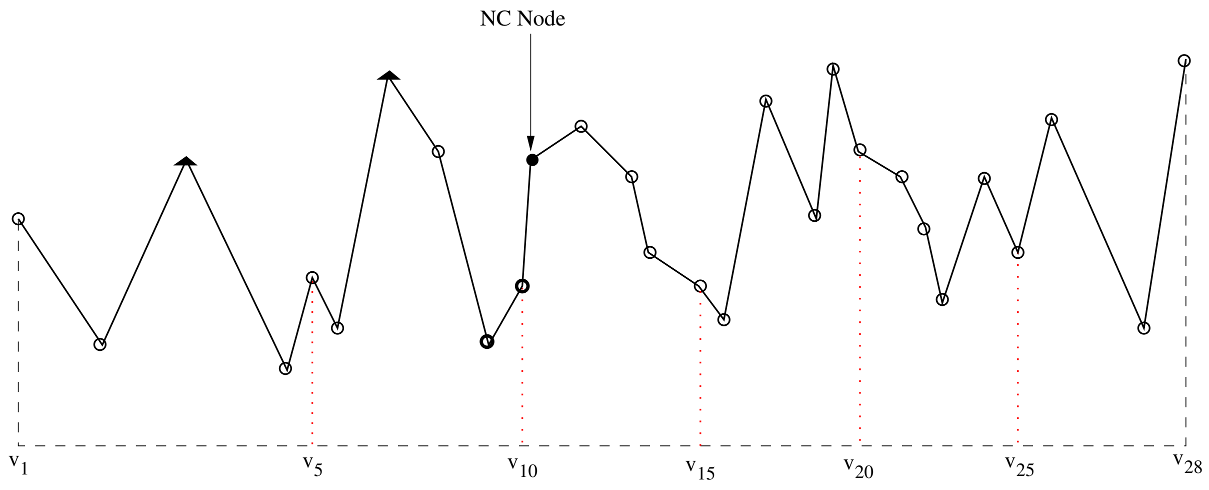


Figure 3.6: Illustration of Next Candidate Node (NC)

Algorithm 3 Greedy Forward Marching Algorithm

```
1: Input: Ordered list of vertices chain $V$ $\{v_0,v_1,.....v_{n-1}\}$ of terrain T
2: Output: Subset $S_g$ of $V$ where guards are placed
3: Compute Visibility Graph (VG) for T
4: j = 1; $S_g$=$\varnothing$
5: Place the first guard at rightmost vertex $v_{i_1}$ that covers all the edges to the left of $v_{i_1}$
6: Delete all visibility edges originating from consecutive vertices covered by $v_{i_1}$
7: Let $T_c$ be consecutive edges covered by $v_{i_1}$
8: $T = T$ - $T_c$
9: while T $\neq \varnothing$ do
10:     Find $v_{i_{j+1}}$=NC($v_{i_j}$)
11:     Let $T_c$ be consecutive edges covered by $v_{i_{j+1}}$
12:     $T = T$ - $T_c$
13:     $S_g = S_g \cup \{v_{i_{j+1}}\}$
14: Output $S_g$
```

The formal sketch of the algorithm is listed as Algorithm 3. The time complexity of Algorithm 3 can be done as follows. Step 3 (computing visibility graph $VG(V, E)$) can be done in $O(|E|+|V|)$ by using algorithm given in [GM87]. Observe that as a guard is placed at vertex v_i, all visibility edges originating from consecutive vertices covered by the guard are removed. Furthermore, each visibility edge is examined at most a constant number of times. Thus the total time taken by the *while* loop is bounded by $|E| + |V|$. Hence the time complexity of Algorithm 3 is $O(|E| + |V|)$.

The complete steps of terrain guarding using GFM- algorithm is illustrated in Figure 3.7.

3.5 Removal of Redundant Guards

Once a terrain is fully guarded using the 'Greedy Ranking Algorithm', the next step is to remove the redundant guards. We follow the same approach as explain in the section 3.3. Figure 3.8 shows the guard placement result using GFM- algorithm. We can see that the guard placed at vertex v_{13} and v_{21} is redundant as all the side covered by the guard at v_{13} is covered by the guard at v_{17}. Similarly all the side covered by the guard at v_{12} is covered by the guard at v_{24}.

Figure 3.9 illustrate the guard placement after redundant guard removal.

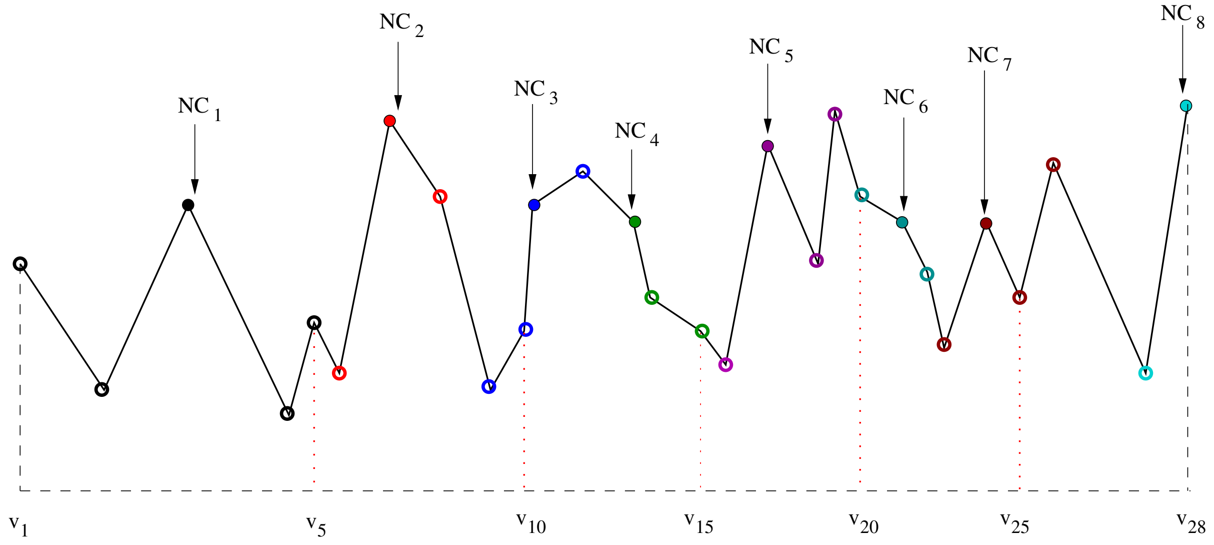


Figure 3.7: Illustration of Greedy Forward Marching algorithm

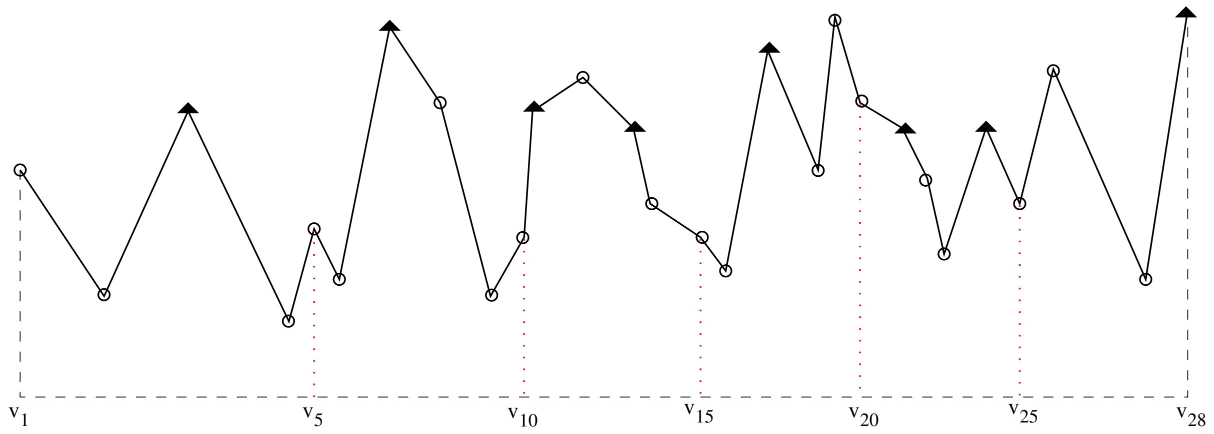


Figure 3.8: Illustration of guard placement with redundant guards

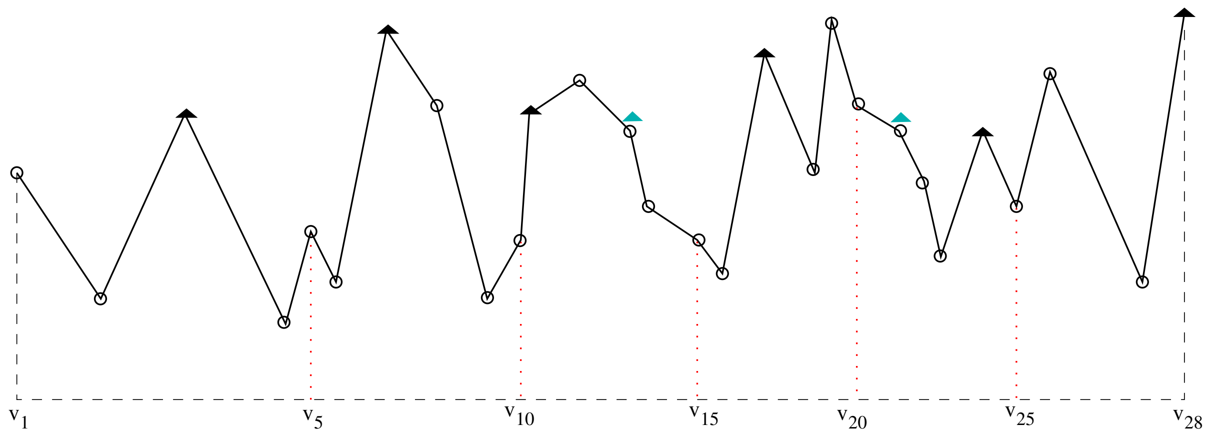


Figure 3.9: Illustration of guard placement after redundant guards removal

Chapter 4

Experimental Investigation

In this chapter we present the experimental investigation of several algorithms proposed in Chapter 3. The implemented algorithms include (i) Generation of terrain models, (ii) Placement of point guards based on Greedy Ranking Algorithm, (iii) Placement of point guards using Greedy Forward Marching Algorithm, and (iv) Removal of redundant guards,

4.1 Program prototype

We developed the program prototype by using Python programming language (Version 2.7). For the ease of program execution, the prototype supports a graphical user interface (GUI) populated by user-friendly GUI-components that include (i) Drawing panel, (ii) Check boxes, (iii) Scroll-able table, and (iv) File input/output and related drop-down menu. We used the GridBagLayout to place various elements on the window. GridBagLayout supports a dynamic grid where each UI component can be placed

The structure of the top-level of the front-end interface (Main Frame) consists of eight components as depicted in Figure 4.1.

The main frame consists of eight components which are (i) Tool-bar Panel, (ii) Status-bar Panel, (iii) Left Top Panel, (iv) Left Bottom Panel, (v) Central Panel, (vi) Right Top Panel, (vii) Right Middle Panel, and (viii) Right Bottom Panel.

4.1.1 File Menu Items

The file menu bar is part of the main frame which contains a drop-down menu as describe below. The file menu consists of four options: Open, Save, Save As, and Quit. These four options facilitate

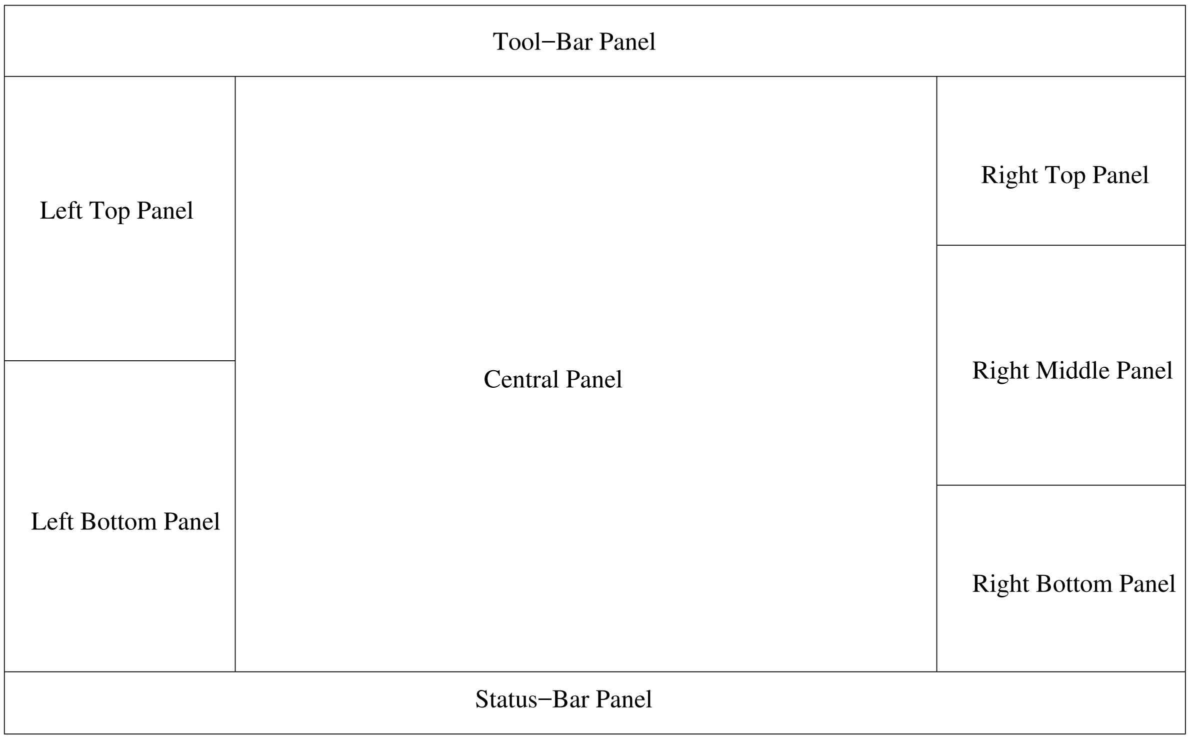


Figure 4.1: Illustration of GUI layout

the user to Open the previously saved data file, save the current data in a new file etc.

S.N.	**File Menu Item**	**Function**
1	Open	Open a previously saved terrain vertices set
2	Save	Save the current terrain vertices to currently open file
3	Save As	Save current terrain vertices to the new file
4	Quit	Quit the application

Table 4.1: Detail of File Menu Items

4.1.2 List of Panels

Seven panels are used as containers for the various GUI components. The purpose and contents of each of these panels are:

Tool-Bar Panel : This is designed for placing GUI buttons that execute (i) Print function, (ii)

Capture Screen-shot, and (iii) Zoom-in/ Zoom-out.

Status-Bar Panel : For displaying copyright information (Text Box).

Left Top Panel : Contains the general control buttons for terrain drawing (draw node, edit node, delete node, move drawing etc.)

Left bottom Panel : Used for placing control buttons and text input fields required for random terrain vertices generation tasks (point count, upper cut, lower cut, set guiding chain, draw guiding chain etc.)

Central Panel : Used for displaying the input terrain and guarding result.

Right Top Panel : Contains algorithm selection check boxes and scrollable table to display guards coordinates generated by the algorithm.

Right Middle Panel : Used for displaying coordinates of terrain vertices in editable and scrollable text-box table.

Right Bottom Panel : Contains text field for vertex count and data set count and click button for data generation.

4.1.3 List of Buttons

The functionalities of various GUI buttons are as describe in the following table.

S.N.	**Buttons**	**Function**
1	CLEAR WINDOW	Erase drawing area (Central Panel).
2	DRAW TERRAIN	Draw terrain implied by the ordered input vertices.
3	VISIBILITY POLYGON	Displaying visibility polygon corresponding to highest ranked vertex.
4	DISPLAY GUARD	For displaying the placement of generated guards.
5	REMOVE REDUNDANT	Displaying placement of guards after removal of redundant guards.
6	RANDOM GENERATE	Randomly generate terrain vertices (nodes).
7	SAVE	Save edited coordinate of Nodes.
8	DISCARD	Discard and restore the old coordinate of the edited Nodes.
9	GENERATE	Triggers generation of data sets as specified in vertex and data set count in text input box of Right Bottom Panel.

Table 4.2: Functionality of Buttons

4.1.4 List of Check-Boxes

S.N.	**Check-Boxes**	**Function**
1	DRAW NODE	Enable vertex drawing with mouse click.
2	EDIT NODE	Enable editing of node coordinates.
3	DELETE NODE	Enable editing of node coordinates.
4	MOVE DRAWING	Enable block movement of all drawn object in Central Panel.
5	SHOW GUIDING CHAIN	Display currently used guiding chain.
6	SET GUIDING CHAIN	Set guiding chain for input vertices.
7	GREEDY RANKING ALGORITHM	Select Greedy Ranking Algorithm for guard placement
8	GREEDY FORWARD MARCHING	Select Greedy Forward Marching Algorithm for guard placement

Table 4.3: Description of check box functionality

4.1.5 List of Text-Areas

S.N.	**Text-Area**	**Function**
1	Guard Coordinate	Display the coordinates of guards in text area.
2	Nodes Coordinates	Display the coordinates of a terrain vertices in text area.

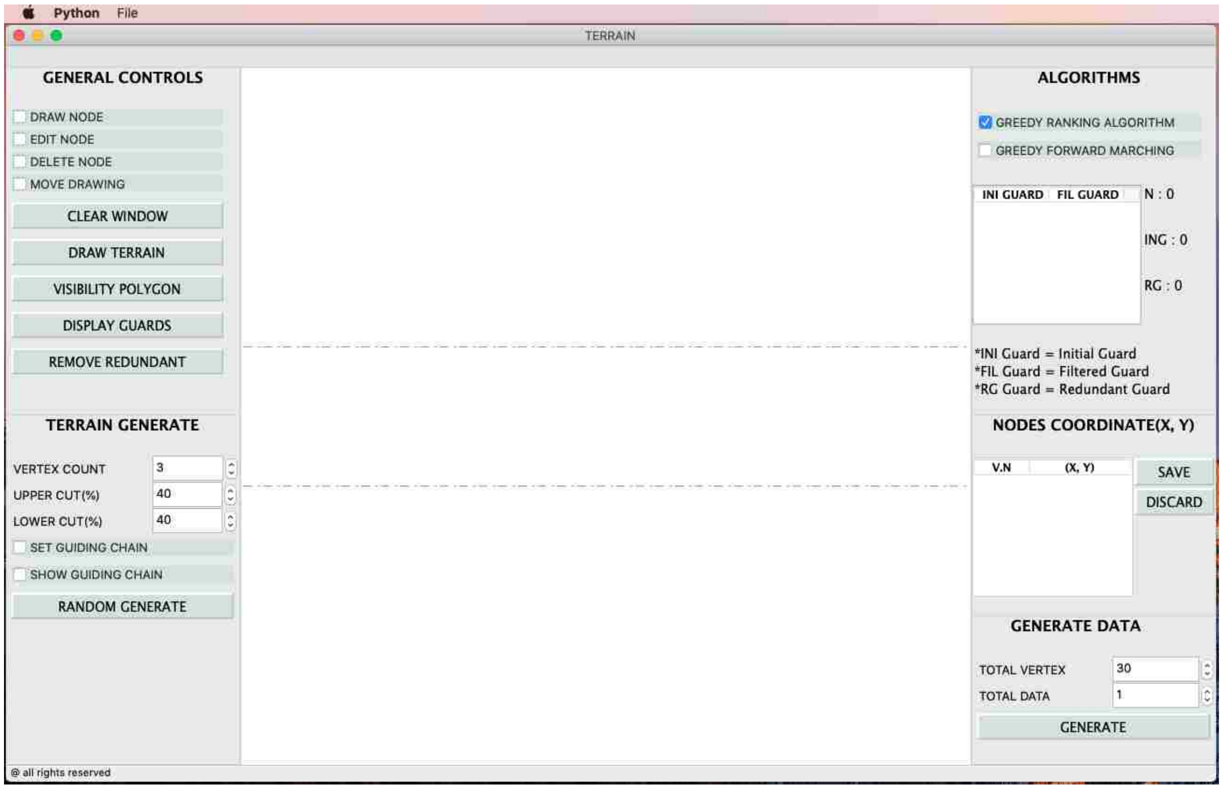
Table 4.4: Description of Text-Areas Functionality

4.1.6 List of Text-Inputs

S.N.	**Text-Inputs**	**Function**
1	VERTEX COUNT	Setting number of random vertices.
2	UPPER CUT (%)	Used for upper limit of guiding strips.
3	LOWER CUT (%)	Used for lower limit of guiding strips.
4	TOTAL VERTEX	Setting number of vertices count.
5	TOTAL DATA	Total number of data set.

Table 4.5: Description of Text-Inputs Functionality

A snapshot of the GUI application is shown in Figure 4.2 below:



[✓ GREEDY RANKING ALGORITHM]

N : 0

ING : 0

RG : 0

*INI Guard = Initial Guard
*FIL Guard = Filtered Guard
*RG Guard = Redundant Guard

VERTEX COUNT 3
UPPER CUT(%) 40
LOWER CUT(%) 40

TOTAL VERTEX 30
TOTAL DATA 1

Figure 4.2: Snapshot of GUI application

4.2 Data Generation

We generate the random terrain vertex for 1.5D terrain using a uniform random number generation algorithm available in Python 2.7. We fix the upper and lower bounds of terrain vertices using terrain generation guiding strip. The terrain generation guiding strip is bounded by two zig-zag chains as illustrated in the Figure 4.3 below.

4.3 Computing Visible Vertex Table (VVT)

For each vertex v_i the set of the vertices visible from it are maintained in a table. For terrain of Figure 4.4 the corresponding table is shown in Table 4.6.

To construct the visibility vertex table, we use the ray tracking methodologies with O(n^2) time

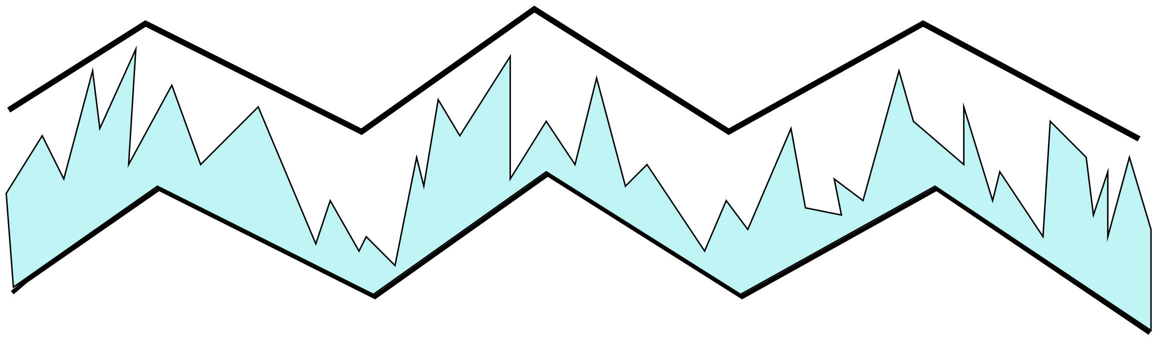


Figure 4.3: Illustration of guiding strip

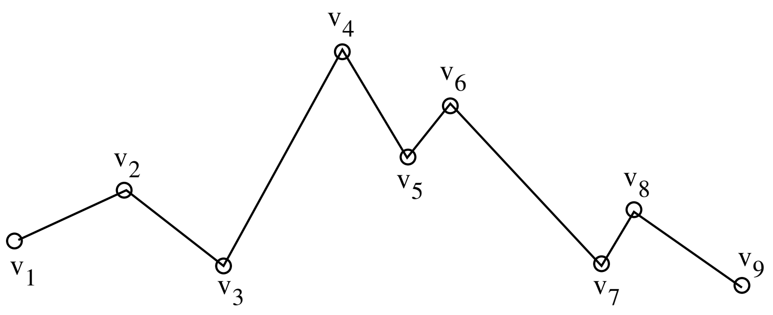


Figure 4.4: Illustration of concept of visibility table

S.N.	**Vertex**	**Visible Vertex List**
1	v_1	[v_2,v_4]
1	v_2	[v_1,v_3,v_4]
3	v_3	[v_2, v_4]
4	v_4	[v_1,v_2, v_3, v_5, v_6]
5	v_5	[v_4, v_6]
6	v_6	[v_4, v_5, v_7, v_8]
7	v_7	[v_6, v_8]
8	v_8	[v_6, v_7,v_9]
9	v_9	[v_8]

Table 4.6: Description visible vertex table

complexity. In this method, we scan terrain from left to right. If we can draw a ray between vertex v_i to v_j without any intersection with the terrain chain lying above it, then we can say that vertex v_i is visible from vertex v_j and vice-versa. For three vertex v_i, v_j and v_k of increasing x-coordinates, their visibility relation can be determined by examining implied left-turn/right-turn. Specifically, if v_i and v_j are visible then v_k is visible from both v_i and v_j if v_i, v_j ,v_k is a left-turn. If it is a right-turn then v_k is not visible from v_i of v_j.

For implementation we identify the *Running Farthest Visible Vertex (RFVV)* for each candidate vertex when the terrain vertices are scanned from left to right. In Figure 4.5, v_2 and v_4 are the first and second RFVV for v_1.

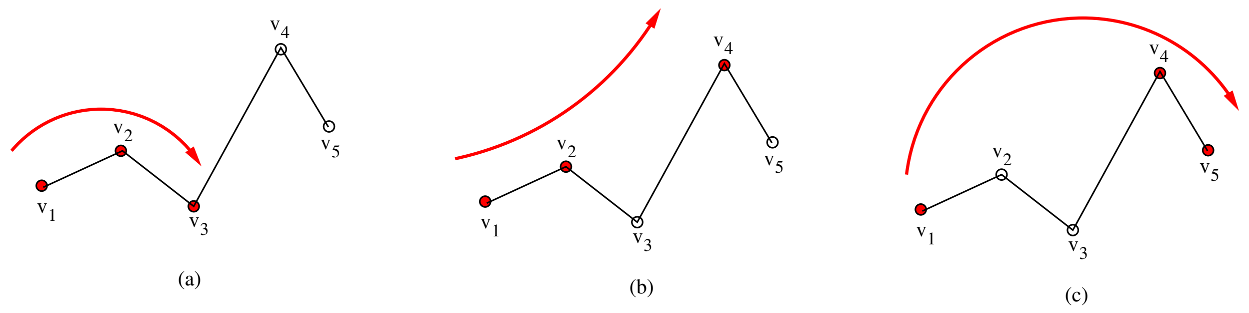


Figure 4.5: Illustration of left turn and right turn

A Formal sketch of the algorithm of constructing the visible vertex table is listed as Algorithm 4.

Algorithm 4 Algorithm to find the vertex visibility table

```
1: **Input**: Terrain vertices $T_i$ sorted according to x-coordinate
2: **Output**: Visibility Table $T_v$
3: $T_v = \varnothing$
4: N = Total terrain vertices
5: for i = 0 to N-1 do
6:     $v_1 = T_i$[i]
7:     RFVV = $T_i$[i+1]
8:     for j = i to N do
9:         $v_2 = T_i$[j]
10:        if isVisible($v_1$, RFVV, $v_2$) then
11:            RFVV = $v_2$
12:            insert $v_2$ into visible vertex set of $v_1$ and update $T_v$
13:            insert $v_1$ into visible vertex set of $v_2$ and update $T_v$
14: return $T_v$
```

Note: isVisible() return true if v_1, LFVV, v_2 has Right-Turn

4.4 Finding Visibility Edge Table (VET)

For vertex v_i, an edge $e_j(v_j,v_{j+1})$ is visible if both end-point vertices v_j and v_j+1 are visible from v_i. In Figure 4.7, for vertex v_1, edge e_1 is visible, e_2 not visible and e_3 is partially visible. So we can not consider e_3 in the visible edge list for v_1. Table 4.7 lists the set of edges visible from each vertex for the terrain shown in Figure 4.6.

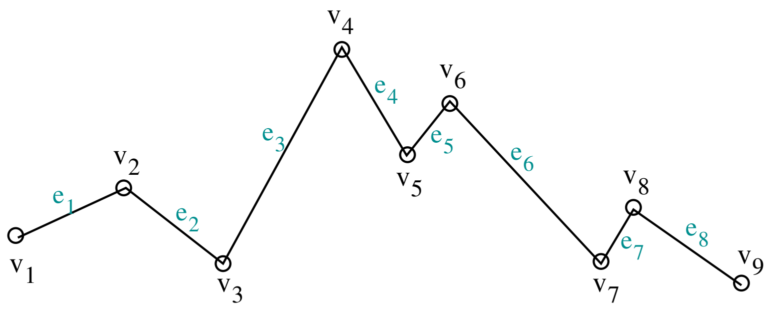


Figure 4.6: Illustration of concept of visibility edge table

S.N.	**Vertex**	**Visible Edge List**
1	v_1	[e_1 (we can not put e_3 as it is not completely visible by v_1)]
1	v_2	[e_1,e_2,e_3]
3	v_3	[e_2,e_3]
4	v_4	[e_1,e_2,e_3,e_4]
5	v_5	[e_4, e_5]
6	v_6	[e_4,e_5,e_6,e_7]
7	v_7	[e_6,e_7]
8	v_8	[e_6, e_7,e_8]
9	v_9	[e_8]

Table 4.7: Example visibility edge table

4.5 Finding the Visibility Polygon

The visibility polygon or visibility region for any vertex v on the terrain is the unbounded polygonal region of all the points of the plane visible from v. Once we compute the visibility vertex table and visibility edge table, we can easily compute the visibility polygon for any vertex of a terrain. For

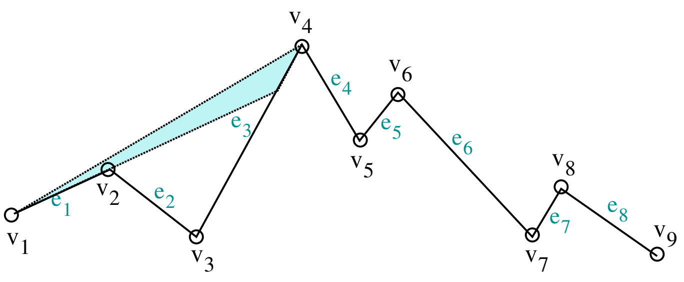


Figure 4.7: Illustration of partially visible edge by vertex

Algorithm 5 Algorithm to find the edge visibility table

```
1: **Input1**: Terrain vertices $T_i$ sorted according to x-coordinate
2: **Input2**: Vertex Visibility Table $T_v$
3: **Output**: Visibility Edge Table $T_e$
4: for each $v_i$ in $T_i$ do
5:     $T_{v_i}$ : Visible vertex set for vertex $v_i$
6:     if length($T_{v_i}$) $= 2$ then
7:         Add $T_{v_i}$ to $T_e$
8:     if length($T_{v_i}$) $> 2$ then
9:         for each consecutive vertices pair ($v_i$,$v_{i+1}$) in $T_{v_i}$ do
10:            Add ($v_i$,$v_{i+1}$) to $T_e$
11: Return $T_e$
```

partially visible edges we use the ray shooting method to identify the partially visible portion of that edge as shown Figure 4.8.

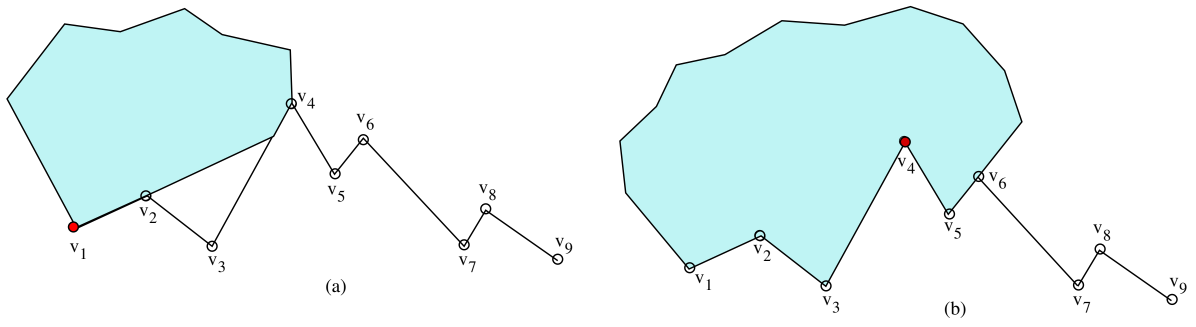


Figure 4.8: Illustration of visibility polygon

4.6 Implementation of Greedy Ranking Algorithm

To implement the Greedy Ranking Algorithm, we first compute the visible vertex table (VVT) and visible edge table (VET). By inspecting these tables, the visibility polygon for each vertex can be computed in a straightforward manner. By examining vertices in each visibility polygon, we determine the ranks. The first guard is placed at the vertex with the highest rank. After this placement, ranking is updated by counting only uncovered edges. The process is repeated until all edges are not covered. A flow chart diagram of this approach is shown in Figure 4.9.

In our GUI , we generate the vertices for terrain in random manner. We create the terrain using these random vertices and use greedy ranking algorithm to generate the guard placement result. A snapshot of our GUI implementation is shown in Figure 4.10.

4.7 Implementation of Greedy Forward Marching Algorithm

Once we compute the visible vertex table (VVT) and visible edge table (VET), we use the algorithm explained in Algorithm 3 to place the point guards on the terrain surface. In each iteration, we find the Next Candidate Node and place the guard at that node. We run our algorithm until any point on the terrain is visible from at-least one guard. The flow chart in Figure 4.11 below illustrates the process used to implement this algorithm.

A snapshot of our GUI implementation is shown in Figure 4.12.

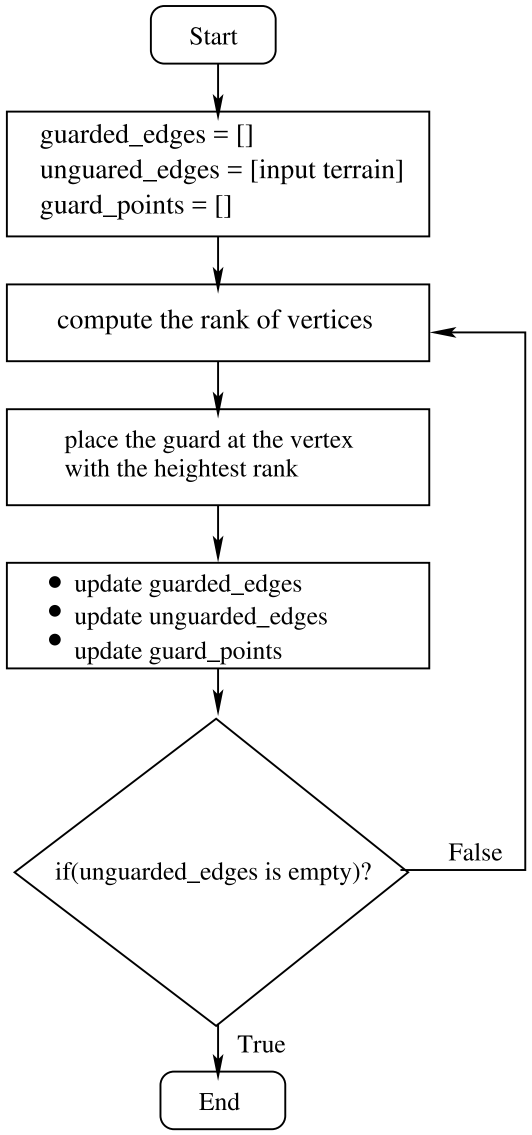
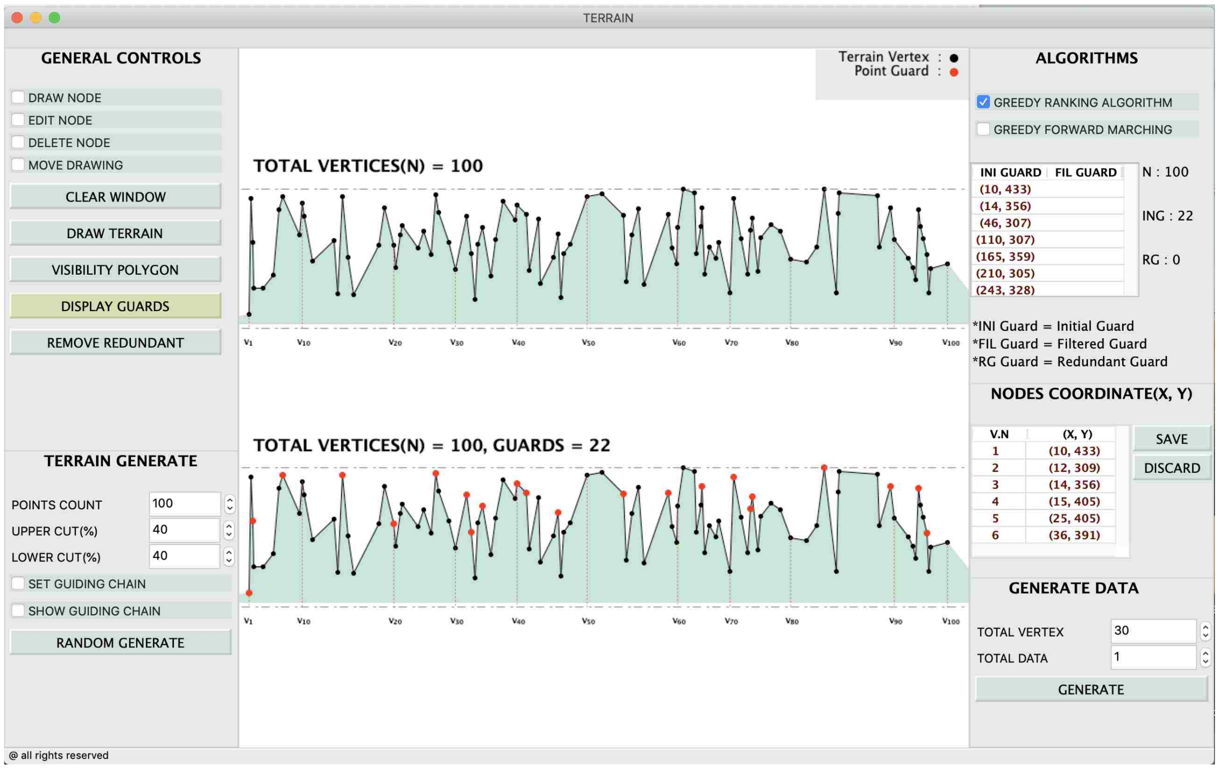


Figure 4.9: Flow chart for implementation of greedy ranking algorithm



POINTS COUNT	100
UPPER CUT(%)	40
LOWER CUT(%)	40

[✓] GREEDY RANKING ALGORITHM
[ ] GREEDY FORWARD MARCHING

INI GUARD	FIL GUARD
(10, 433)	
(14, 356)	
(46, 307)	
(110, 307)	
(165, 359)	
(210, 305)	
(243, 328)	

N : 100

ING : 22

RG : 0

*INI Guard = Initial Guard
*FIL Guard = Filtered Guard
*RG Guard = Redundant Guard

V.N	(X, Y)
1	(10, 433)
2	(12, 309)
3	(14, 356)
4	(15, 405)
5	(25, 405)
6	(36, 391)

TOTAL VERTEX	30
TOTAL DATA	1

Figure 4.10: Snapshot of Greedy Ranking Algorithm implementation in GUI application

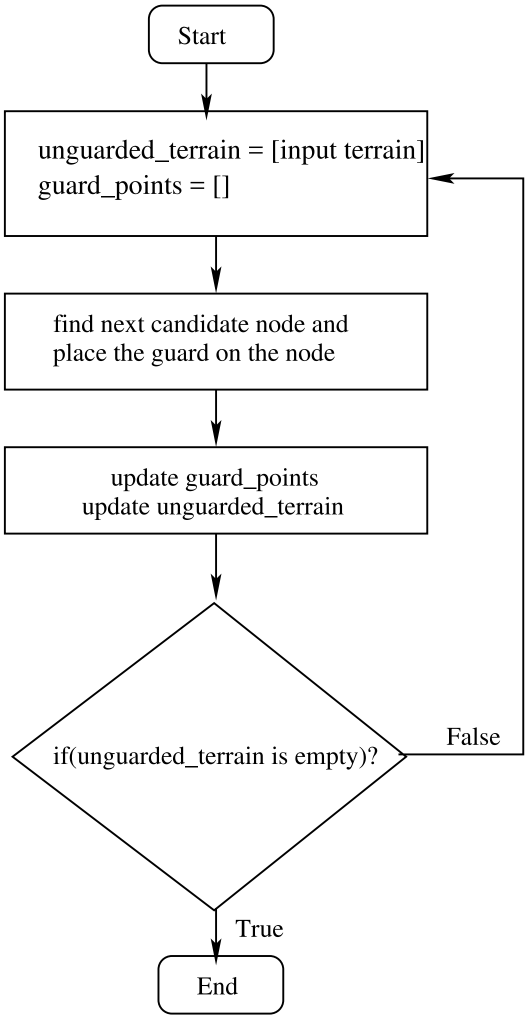
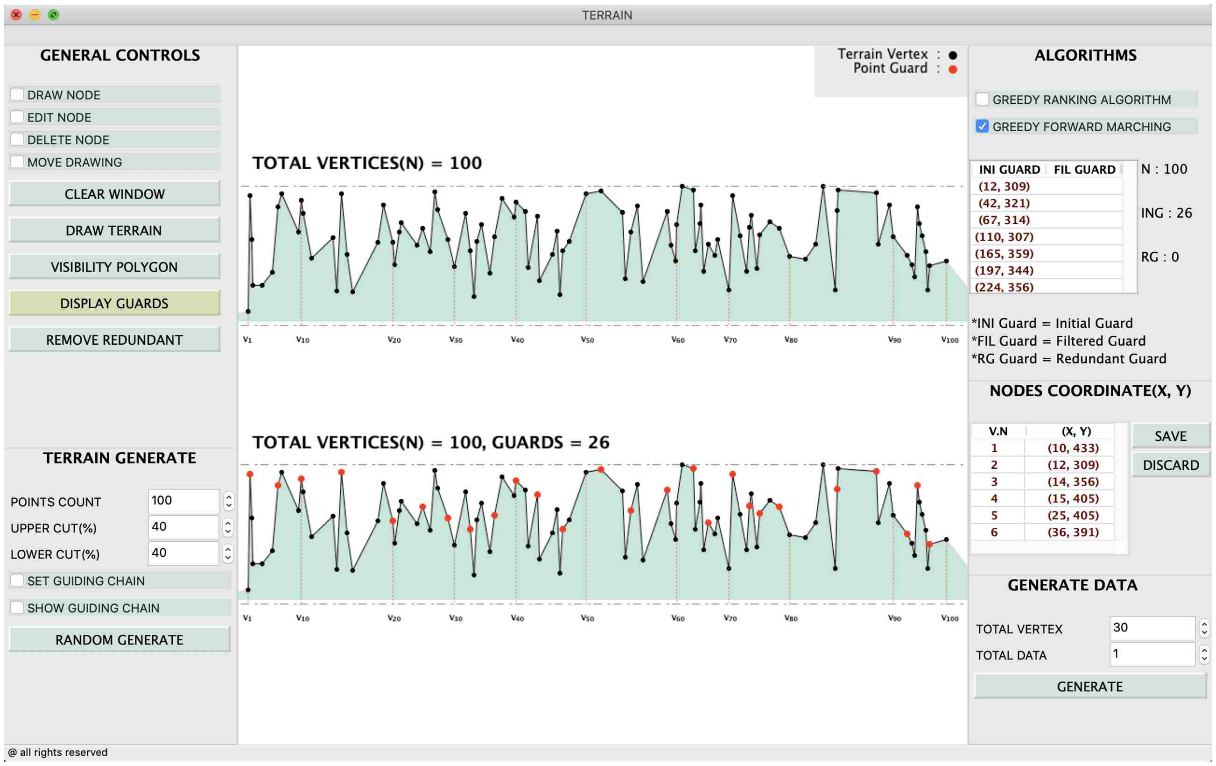

Start
unguarded_terrain = [input terrain]
guard_points = []
find next candidate node and
place the guard on the node
update guard_points
update unguarded_terrain
if(unguarded_terrain is empty)?
False
True
End

Figure 4.11: Flow chart of Greedy Forward Marching algorithm



POINTS COUNT	100
UPPER CUT(%)	40
LOWER CUT(%)	40

[✓ GREEDY FORWARD MARCHING]

INI GUARD	FIL GUARD
(12, 309)	
(42, 321)	
(67, 314)	
(110, 307)	
(165, 359)	
(197, 344)	
(224, 356)	

N : 100

ING : 26

RG : 0

*INI Guard = Initial Guard
*FIL Guard = Filtered Guard
*RG Guard = Redundant Guard

NODES COORDINATE(X, Y)

V.N	(X, Y)
1	(10, 433)
2	(12, 309)
3	(14, 356)
4	(15, 405)
5	(25, 405)
6	(36, 391)

GENERATE DATA

TOTAL VERTEX	30
TOTAL DATA	1

Figure 4.12: Snapshot of Greedy Forward Marching algorithm in GUI application

4.8 Result Comparison

We executed a series of experiments to test the performance of the proposed algorithms. We ran the algorithm with terrains having different numbers of nodes. We computed the number of guards using two algorithms: Greedy Ranking, and Greedy Forward. We then removed the redundant guards and prepared the final guard placement result. The result obtained from the experiment are listed in Table 4.8 - Table 4.10.

Nodes (N)	**Number of Guard by Greedy Ranking (GR)**	**Guard after Redundancy Removal (GR)**	**Number of Guard by Greedy Forward Marching (GFM)**	**Guard after Redundancy Removal (GFM)**
10	3	3	3	3
25	8	8	8	8
50	10	10	11	11
75	14	14	16	16
100	23	22	24	23
125	29	28	29	27
150	30	29	33	33
175	40	38	42	41
200	42	41	46	45
250	56	54	58	57
300	63	62	67	67
350	72	71	81	78
400	83	80	94	89
450	93	91	102	97
500	103	101	115	113
550	110	109	121	119
600	116	114	133	128
650	135	133	148	143
700	150	147	161	158
750	151	150	165	160
800	170	169	190	187
850	182	179	191	184
900	165	164	203	200
950	211	208	221	220
1000	205	204	230	224
1050	217	214	234	231
1100	227	216	250	241
1150	248	242	255	252
1200	237	231	271	263

Table 4.8: Table for number of nodes count (10-1200)

Nodes (N)	**Number of Guard by Greedy Ranking (GR)**	**Guard after Redundancy Removal (GR)**	**Number of Guard by Greedy Forward Marching (GFM)**	**Guard after Redundancy Removal (GFM)**
1250	255	253	284	279
1300	280	275	294	283
1350	291	287	305	287
1400	284	279	316	305
1450	302	298	325	315
1500	315	307	339	330
1550	311	306	350	339
1600	310	304	365	358
1650	342	334	371	354
1700	347	344	384	371
1750	369	366	394	390
1800	379	370	406	400
1850	389	382	419	412
1900	397	393	439	422
1950	410	403	448	436
2000	417	415	455	438
2050	424	420	468	456
2100	437	429	470	453
2150	452	444	493	480
2200	463	456	496	486
2250	481	475	513	500
2300	485	481	525	510
2350	489	487	546	530
2400	504	499	558	541
2450	520	512	563	550
2500	523	517	564	549
2600	528	521	578	565
2700	566	560	625	607
2800	592	584	638	627
2900	607	600	668	647
3000	614	608	679	660
3100	652	643	696	680
3200	664	659	734	711
3300	693	685	770	739
3400	700	691	780	761
3500	717	705	796	775
3600	760	749	816	796
3700	793	780	857	840

Table 4.9: Table for number of nodes count (1250-3700)

Nodes (N)	**Number of Guard by Greedy Ranking (GR)**	**Guard after Redundancy Removal (GR)**	**Number of Guard by Greedy Forward Marching (GFM)**	**Guard after Redundancy Removal (GFM)**
3800	796	784	865	843
3900	819	805	903	875
4000	833	821	923	892
4100	839	829	938	922
4200	864	850	950	928
4300	886	873	992	964
4400	894	884	1030	998
4500	934	920	998	971
4600	962	948	1040	1019
4700	975	963	1063	1040
4800	998	982	1088	1064
4900	1028	1009	1120	1095
5000	1048	2028	1130	1105
5100	1091	1072	1174	1146
5200	1099	1082	1199	1166
5300	1106	1089	1216	1175
5400	1132	1109	1232	1206
5500	1152	1136	1255	1222
5600	1162	1146	1288	1246
5700	1192	1180	1306	1269
5800	1219	1195	1323	1282
5900	1239	1225	1338	1305
6000	1248	1222	1349	1318
6100	1296	1274	1406	1359
6200	1302	1286	1411	1375
6300	1313	1290	1445	1399
6400	1322	1306	1468	1435
6500	1331	1311	1459	1419
6600	1375	1351	1505	1450
6700	1385	1360	1527	1487
6800	1423	1402	1558	1511
6900	1444	1423	1579	1529
7000	1472	1443	1601	1556

Table 4.10: Table for number of nodes count (3800-7000)

The tabulated data was plotted in a graph with the number of guards as y-axis and the number of nodes as x-axis, for both Greedy Ranking Algorithm and Greedy Forward Marching Algorithm as shown in Figure 4.13.

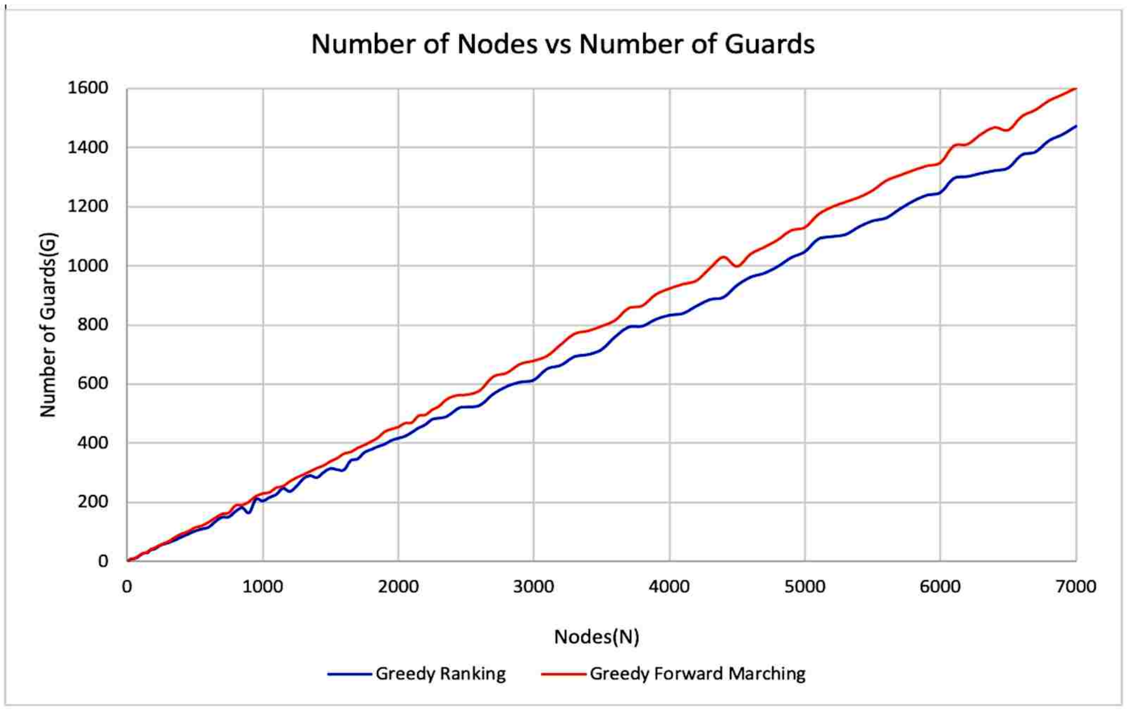


Figure 4.13: Graph between Number of Nodes vs Number of Guards

To find the effectiveness of redundancy removal, Figure 4.14 shows the number of identified redundant guards for both algorithms. The plot shows that the number of redundant guards is consistently larger for the placement obtained by the Greedy Forward Marching Algorithm

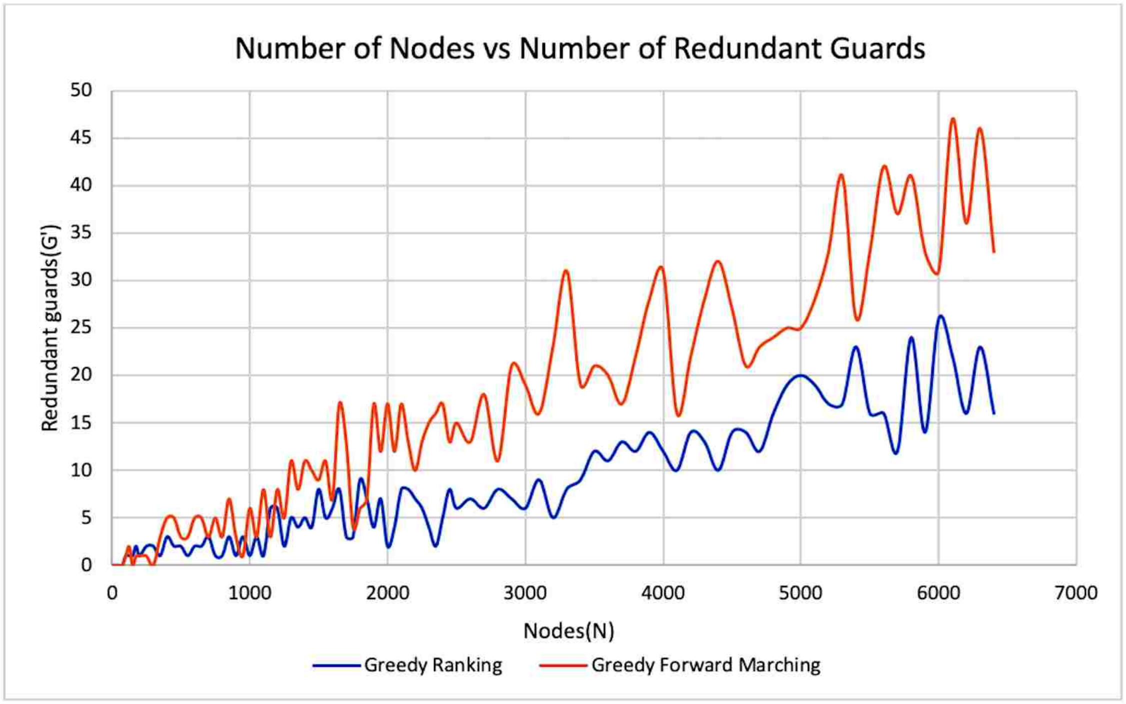


Figure 4.14: Graph between Number of Nodes vs Number of Redundant Guards

Chapter 5

Conclusion

We presented a brief review of existing algorithms for placing point guards in 1.5D terrains and simple polygons. We presented two heuristic algorithms for covering 1.5D terrain by point guards. We obtained several experimental results on the performance of the presented algorithms (Greedy Ranking Algorithm and Greedy Forward Marching Algorithm). To enhance the performance of both algorithms we considered identifying redundant point guards (guards that are not necessary). Our experimental result shows that the performance of the Greedy Ranking Algorithm is better than the performance of the Greedy Forward Marching Algorithm. We observed this result on several terrain input sizes 10, 25, 50, 75 ,......7000. For all these input sizes, the data shows the performance of Greedy Ranking is consistently better. This can be observed in Figure 4.3. One of the additional contributions of this thesis is the generation of 1.5D terrain data of various sizes. The generation is done randomly by using guiding a strip. At present, an implementation by the guiding strip is taken as a shape with zig-zag structure. To make it more realistic it would be interesting to have strip of other structures. This can be an interesting future work. The performance of Greedy Forward Marching is not very good. There is room for improvement by enhancing this approach. One approach for improvement would be to look forward beyond the Next Candidate Node while placing the next guard. This is expected to improve the performance of this algorithm at the expense of time complexity. Recently, some authors have proposed the notion of one-sided guard placement [LH18]. It would be interesting to convert our proposed heuristic to a one-sided version of visibility.

There is ample scope to developing better algorithms for identifying redundant guards. In the method proposed to identify redundant guards at v_i we only look for the pair of guards (one to the left and one to the right of v_i). A generalization of this technique is to look for coverage by more than two guards (say three). This should improve the spotting of redundant guards at the expense of time complexity.

A better approach for generating realistic 1.5D terrain would be to sample points on the horizon of a real terrain and connect them. This approach is certainly feasible and would be a good avenue for further research. We have taken an unlimited visibility model for defining visible vertices: two vertices are visible as long as the line segment connecting them does not intersect with the terrain, no matter how far apart they are located. A more realistic model is to incorporate the notion of *limited visibility.* Under this model, two vertices v_i and v_j are visible if (i) the line segment connecting them does not intersect with the terrain, and (ii) they are not farther apart than a certain distance d. It would be an interesting research exercise to develop guard placement algorithms under limited visibility.

Bibliography

[BMKM05] Boaz Ben-Moshe, Matthew Katz, and Joseph 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. Society for Industrial and Applied Mathematics, 2005.

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

[GD18] L. Gewali and B. Dahal. Covering 1.5d terrain by common height towers. In *2018 26th International Conference on Systems Engineering (ICSEng)*, pages 1–6, Dec 2018.

[Gew95] Laxmi P. Gewali. Recognizing s-star polygons. *Pattern Recognition*, 28(7):1019–1032, 1995.

[Gho10] Subir Kumar Ghosh. Approximation algorithms for art gallery problems in polygons.(report). *Discrete Applied Mathematics*, 158(6), 2010.

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

[GM87] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 11–19, Oct 1987.

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

[LH18] Wei-Yu Lai and Tien-Ruey Hsiang. Continuous terrain guarding with two-sided guards. 2018.

[LL86] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, March 1986.

[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.

[VP12] Tzvetalin Vassilev and Stefan Pape. Visibility: Finding the staircase kernel in orthogonal polygons. volume 2, 05 2012.

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Jiwan Khatiwada

nawijj@gmail.com

Degrees:

Master of Science in Computer Science 2019

University of Nevada Las Vegas

Thesis Title: Approximation Algorithm For Illuminating 1.5D Terrain

Thesis Examination Committee:

Chairperson, Dr. Laxmi Gewali, Ph.D.

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

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

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