WEIGHTED MULTI-VISIBILITY ANALYSIS ON DIRECTIONAL PATHS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞIL ŞEKER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
GEODETIC AND GEOGRAPHIC INFORMATION TECHNOLOGIES

DECEMBER 2010

Approval of the thesis:

WEIGHTED MULTI-VISIBILITY ANALYSIS ON DIRECTIONAL PATHS

submitted by **ÇAĞIL ŞEKER** in partial fulfillment of the requirements for the degree of **Master of Science in Geodetic and Geographic Information Technologies Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Assoc. Prof. Dr. Mahmut Onur Karslıoğlu _____
Head of Department, **Geodetic and Geographic Information Technologies**

Prof. Dr. Vedat Toprak _____
Supervisor, **Geological Engineering Department, METU**

Examining Committee Members:

Assoc. Prof. Dr. Zuhal Akyürek _____
Civil Engineering Department, METU

Prof. Dr. Vedat Toprak _____
Geological Engineering Department, METU

Assoc. Prof. Dr. Mahmut O. Karslıoğlu _____
Civil Engineering Department, METU

Assoc. Prof. Dr. Şebnem Düzgün _____
Mining Engineering Department, METU

Dr. Cevat Şener _____
Computer Engineering Department, METU

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÇAĞIL ŞEKER

Signature :

ABSTRACT

WEIGHTED MULTI-VISIBILITY ANALYSIS ON DIRECTIONAL PATHS

Şeker, Çağıl
M.S., Department of Geodetic and Geographic Information Technologies
Supervisor : Prof. Dr. Vedat Toprak

December 2010, 155 pages

Visibility analysis is an important GIS tool that is used in a diverse array of disciplines ranging from earth sciences to telecommunications.

Multi-visibility, as a cumulative type of visibility, combines many point-to-point results into a multi-value array. Points, lines, or areas can be used as sources or targets; and the combined values can be calculated in both ways. Through multi-visibility, a special 2.5D visibility value surface can be constructed over a digital elevation model. The effectiveness of multi-visibility can be increased with weighted target zones. Other types of weighting criteria can be defined, such as distance and angle.

Open source GIS tools offer a limited amount of support for that type of multi-visibility analysis. In this study, a weighted multi-visibility methodology has been developed which accepts a path as the source. The path can have a specific direction to account for moving subjects that have a specific view angle based on their direction. A software tool has been developed to apply the methodology

in a practical and automated way. The tool was written in Python programming language and can be run as a plugin to the open source Quantum GIS software.

The proposed weighted multi-analysis methodology and its software tool can be used to assess the quality of visibility through the generation of value surfaces and calculation of a combined quantitative visibility value for the full path.

Keywords: Visibility analysis, Multi-visibility, GIS, Digital elevation model

ÖZ

YÖNLÜ HATLARDA AĞIRLIKLI ÇOKLU-GÖRÜNÜRLÜK ANALİZİ

Şeker, Çağıl
Yüksek Lisans, Jeodezi ve Coğrafi Bilgi Teknolojileri
Tez Yöneticisi : Prof. Dr. Vedat Toprak

Aralık 2010, 155 sayfa

Görünürlük analizi, jeolojiden telekomünikasyona, geniş bir disiplinler dizisi tarafından kullanılan önemli bir GIS aracıdır.

Çoklu-görünürlük, yığışımlı bir görünürlük türü olarak, birden çok noktadan-noktaya sonucu, bir çoklu-değer dizisi içerisine birleştirmektedir. Nokta, hat, veya alanlar, kaynak ve hedefler olarak kullanılabilmekte ve birleşik değerler her iki yönde de hesaplanabilmektedir. Çoklu-görünürlük aracılığıyla, bir sayısal yükseklik modeli üzerine özel bir 2.5B görünürlük değer yüzeyi oluşturulabilir. Çoklu-görünürlüğün etkinliği ağırlıklı hedef bölgeleri ile yükseltilebilir. Uzaklık ve açı gibi diğer ağırlıklandırma kriterleri de tanımlanabilir.

Açık kaynak GIS araçları bu tür çoklu-görünürlük analizi için sınırlı bir destek sunmaktadırlar. Bu çalışmada, kaynak olarak bir hattı kabul eden, ağırlıklı bir çoklu-görünürlük metodolojisi geliştirilmiştir. Yöne dayanan belirli görüş açıları bulunan, hareket halindeki özneleri de dikkate almak için hatlar belirli yönlere sahip olabilmektedir. Metodolojiyi pratik ve otomasyona dayalı bir şekilde

uygulayabilmek için bir yazılım aracı geliştirilmiştir. Araç Python programlama diliyle geliştirilmiştir ve açık kaynak Quantum GIS yazılımına eklenti olarak çalıştırılabilmektedir.

Önerilen ağırlıklı çoklu-görünürlük metodolojisi ve ilgili yazılım aracının, değer yüzeylerinin yaratılması ve tüm hat için bütünleşik nicel görünürlük değerinin hesaplanması aracılığıyla, görünürlük kalitesinin değerlendirilmesinde kullanılması mümkündür.

Anahtar Kelimeler: Görünürlük analizi, Çoklu-görünürlük, CBS, Sayısal yükseklik modeli

To
my family
and
my friend Oya

ACKNOWLEDGMENTS

I would like to express my thanks to my advisor Prof. Dr. Vedat Toprak, for his continuous guidance, valuable support and great patience. Without his support and encouragement, this thesis would not even have been possible.

I would like to express my gratitude and love to my mother Ayşe and my brother Ilgın, for their amazing moral support and for always being there for me.

I would also like to thank to my friends Hale, Serdar, Ayberk, Gökçe, and Güney, for their understanding and friendship throughout this journey.

I would like to specially thank to my dear friend Oya Sertel, for her awesome encouragement, valuable contributions and continuous moral support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

One of the most often used topographic relationships between two spatial locations is their inter-visibility. Visibility in this sense, can either mean direct optical visibility or in a general sense, detectability by any kind of electromagnetic signal traveling through the medium.

Building on that basic premise, several types of visibility assessments can be carried out. The usual method is to define a set of source or observer locations, and a set of target locations. Either of these sets can be defined as distinct points, line-based paths, or entire regions. Standard visibility analysis usually deals with point sources and regional targets. The target area is usually given as a radius of a circular area having the source location as its center point.

The resulting surface of visible area for a point source is usually called as a "*viewshed*". When the source set consists of more than one point, the analysis becomes a "*multi-visibility*" or a "*cumulative viewshed*" analysis.

As a popular GIS technique, visibility analysis is more or less supported by many professional tools in the field. However, open source, non-commercial GIS software offer less solutions in this aspect compared to their commercial counterparts. Many popular open source tools currently have no support even for a standard single point visibility analysis.

1.1 Assessment of Visibility on Paths

One of the most common uses for a visibility analysis is when the source is a series of lines; *i.e.,* when it is a path. When the source set is a line-based path, since it involves multiple (potentially infinite) points, it is a multi-visibility situation.

An important parameter for any kind of visibility analysis is the direction of sight. It is usually expressed as an angular interval which restricts the target sets to the set of points lying inside this interval, forming a field of view (FoV). Visibility along a path usually indicates a sense of movement along that path. The direction of sight along the path would usually change in accordance with the direction of path.

Another useful parameter for the assessment would be the 'value' or the 'weight' of visibility for the target locations. Not all of the target points may be of equal importance to the viewer. Usually, it is desirable to define some kind of preference for specific target locations through the use of a weighting system. This weighting can be used to construct a weighted multi-visibility result.

Current popular GIS tools usually do not have direct solutions to these types of non-binary directional visibility problems, making them difficult to use for many practical applications.

1.2 Purpose and Scope

The aim of this study is to develop a unified visibility analysis methodology to assess the 'visibility value' of directional paths. The methodology uses cumulative viewsheds with weighted target sets to calculate quantitative visibility values as an enhancement to the standard multi-visibility analysis. Moreover, the methodology can be used to produce weighted visibility surfaces for the target areas and visibility value profiles for the source paths. These visibility surfaces and visibility value profiles are constructed from weighted cumulative counts of the target points.

For the automation of the process, an open source software tool has been developed with the ability to calculate and display weighted multi-visibility results for both the directional source paths and the target areas.

This study has been limited to vector based one dimensional paths and direct ground level optical visibility on raster type topographic surfaces. Specific details such as the Earth's curvature, atmospheric conditions, error assessment, edge effects are not included in the scope.

For the software tool, the raster paths were defined as vector poly-lines and the target zones were given as vector polygons. The tool operates directly on the discrete pixels of the raster data (corresponding to the given vectors) and does not perform any interpolation or non-discrete analysis.

1.3 Summary of the Chapters

In this chapter, an overview of the problem and the proposed solution is given along with the limits of this study. In Chapter 2, some of the core concepts of visibility analysis is discussed like the digital elevation models and data formats. Furthermore, a detailed analysis of visibility is provided along with the summaries of some previous works on the subject.

The general methodology for the proposed solution is defined in Chapter 3. The distinct approaches combined in the methodology is discussed in this chapter, and then the visualization aspect of the results is examined.

Chapter 4 deals with the software development process and provides information on the development platform, the algorithms used and the features implemented.

A test case for the methodology is provided in Chapter 5 where three different paths around a lake are compared in both directions.

The discussion of the results and the implications are given in Chapter 6. Finally, a commentary and final remarks for the study is provided in Chapter 7. That chapter also discusses further research possibilities on the subject.

CHAPTER 2

BACKGROUND

Visibility information for a given area has always been an interest to researchers in a number of diverse fields such as archeologists, city planners, tourism agencies, forest managers, geological engineers, astronomers, and military analysts.

Due to the computational nature of the problem, visibility related information is usually produced with the help of computers and specialized software. Therefore, many of the approaches rely on some kind of digital elevation data to be ready.

Studies on computational visibility date far back to sixties and seventies since it is one of the first interest areas of the GIS field. Consequently, there are many studies from various disciplines incorporating visibility analysis into their researches.

2.1 Digital Elevation Models

2.1.1 Definition

Visibility analysis requires the availability of accurate relief (elevation) data of the target area as the primary input. Elevation, being a typical geographic field, can be defined and measured at an infinite selection of points for a bounded finite area.

When a geographic field is needed to be represented in a finite and discrete domain, such as the memory of a computer, some kind of approximation is needed. In geographical information systems, the preferred approximation is to store a finite

but logically chosen set of points along with an interpolation function to derive the value for the remaining points. For elevation, this approximation is usually called as a *Digital Terrain Model (DTM)* or a *Digital Elevation Model (DEM)* (DEM will be preferred from here on).

Lee (1989) stated that "any digital representation of the continuous variation of relief of the earth's surface may be referred to as a *Digital Elevation Model (DEM)*." A more formal, but more restrictive definition is given by Floriani and Magillo (1993):

> *A natural terrain can be described as a continuous function* $z = f(x, y)$*, defined over a connected subset* D *of the* $x - y$ *plane. Thus, a* Mathematical Terrain Model (MTM) *can be defined as a pair* $M \equiv (D, f)$*.*
>
> *A* Digital Terrain Model (DTM) *is defined as a planar subdivision* Σ *of the domain* D *into a collection of planar regions* $R = \{R_1, R_2, \ldots R_m\}$ *and by a family* F *of continuous functions* $z = f_i(x, y), i = 1, 2, \ldots m$*, each defined on* R_i *and such that* $f_i(x, y) = f_j(x, y)$*, for every* $(x, y) \epsilon R_i^* \bigcap R_j^*$ *(where* R_i^* *denotes the closure of region* R_i*). Thus, a DTM can be expressed as a pair* $M \equiv (\epsilon, F)$*. (Floriani and Magillo, 1993)*

Although DEMs store third dimensional elevation data, they are not fully three dimensional representations. Longley et al. (2005) define DEMs as *2.5D*, since the third dimension is treated as a single-valued function of two horizontal variables.

Elevation values in a DEM are usually defined as exact values. However, there are usually known error limits for the data. Anile et al. (2003) defined a *fuzzy DEM (FDEM)* that stored the elevation data as fuzzy upper and lower bounds to capture the known error limits in the data itself.

Mark (1978) identified mathematical and image methods as the two main classes for a DEM classification framework. Floriani and Magillo (1993) classified the DEMs into two categories: *Regular Square Grids (RSGs)* and *Polyhedral Terrain Models (PTMs)*. A different type of categorization has been defined by de By and Kainz (2001) with three groups: *tessellations*, *vector representations*, and their *hybrids*.

Tessellation is a partition of the field into areas or cells in a systematic way. An important property of such cells is that they do not themselves record their specific geographic references; it must be calculated from a base coordinate (de By and Kainz, 2001). Usually, every cell in a tessellation would have a single value representing the field state in that area.

Tessellations can be regular or irregular. A simple square-wise, triangular, or hexagonal division of the area would constitute a regular tessellation. Regular Square Grids (RSGs) are then, square-wise regular tessellations (Figure 2.2). An example for the irregular tessellation would be the region quadtree (de By and Kainz, 2001). Figure 2.1 shows a quadtree and Figures 2.2 and 2.3 show how the regularly spaced points form a grid on a surface.

There are several ways a tessellation can be interpreted in terms of the cell values. For example, the field value of a cell can either be a representation of the full cell area, or just a single point inside that cell. The first option is usually not preferred as it results in a discrete, non-continuous field. The second option requires the specification of the point (center or one of the corners) and one or more good interpolation functions for the intermediate points.

Vector representations (for example, contour line based methods) are not as widely used as tessellations in its purest form. However, a hybrid of tessellations and vector representations, known as the *Triangular Irregular Network (TIN)*, is relatively

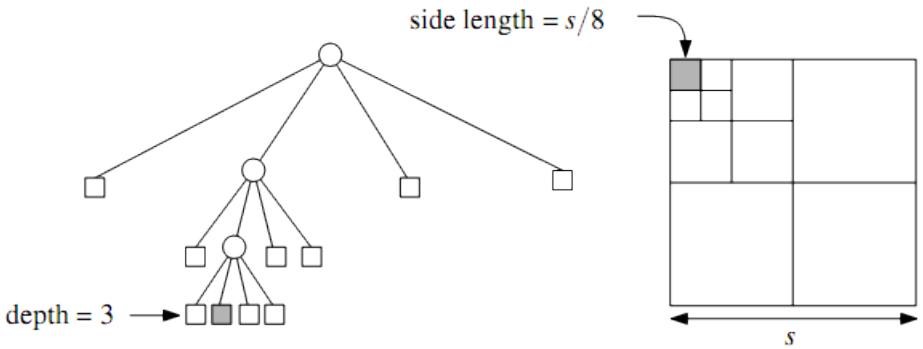


Figure 2.1: A quadtree example. In the diagram, a third level node in the tree and its corresponding area inside the whole field are marked.(de Berg et al., 2008)

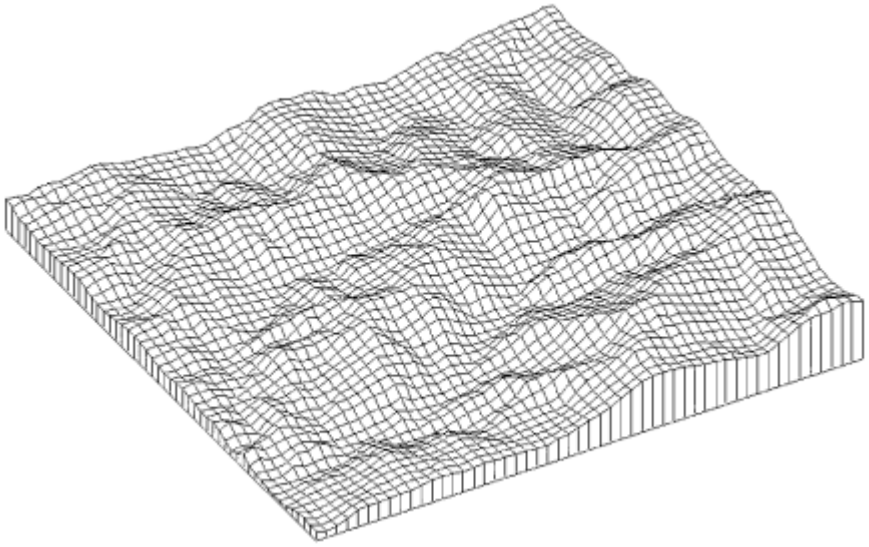


Figure 2.2: Perspective 3D view of a terrain with regular square grid superimposed. (Konecny, 2003)

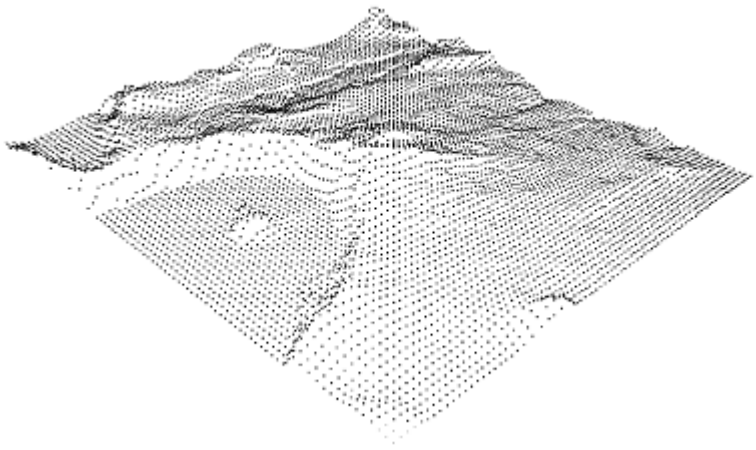


Figure 2.3: Perspective 3D view of a terrain defined by regular points. (Kaučič and Žalik, 2002)

common in the industry. TIN was initially designed by Peucker et al. (1978) in 1978. TINs are a subgroup of Polyhedral Terrain Models according to Floriani and Magillo (1993), since it defines a three dimensional polyhedral surface. Figure 2.4 shows the polyhedral surface of TIN with respect to the two dimensional surface.

In a TIN, a set of specially selected points with known elevation values are used to divide the area into triangular cells. Although there are several ways to perform the tessellation, an optimal approach is the *delaunay triangulation* (Figure 2.5). A mathematical explanation for delaunay triangulation is given by de Berg et al. (2008).

The reference points form the corners of triangles in a TIN and the value for a cell has to be calculated by an interpolation method. One way of doing this interpolation would be to define the cell as a triangular plane in a three dimensional space and calculate the elevations on that plane surface accordingly. The points in a single TIN plane do not necessarily have the same elevation values (Maloy and Dean, 2001).

TIN model allows more efficient context-based distribution of measurement points. Areas with more irregular surfaces (such as ridges, peaks etc.) can be defined with more data points as opposed to relatively smooth areas which can be defined with only a couple of data points.

Longley et al. (2005) listed six different approximate representations of a geographic field as represented in Figure 2.7.

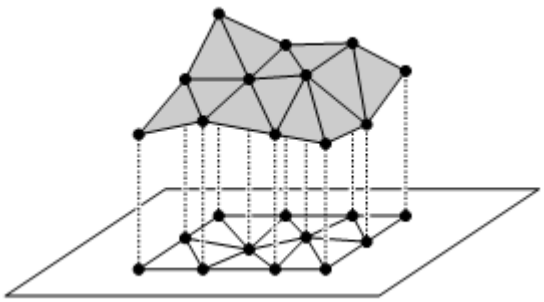


Figure 2.4: Illustration of a polyhedral surface constructed from sample points of a TIN. (de Berg et al., 2008)

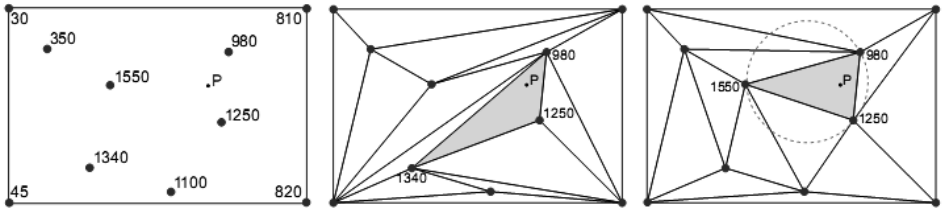


Figure 2.5: Construction of TIN surfaces from sample points. The last one shows optimal Delaunay Triangulation. (de By, 2001)

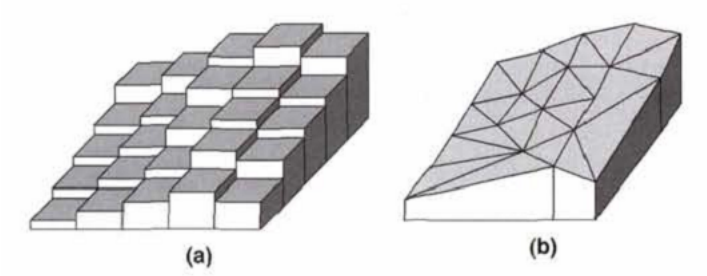


Figure 2.6: Comparison of (a)raster and (b)TIN data models of the same terrain. (Maloy and Dean, 2001)

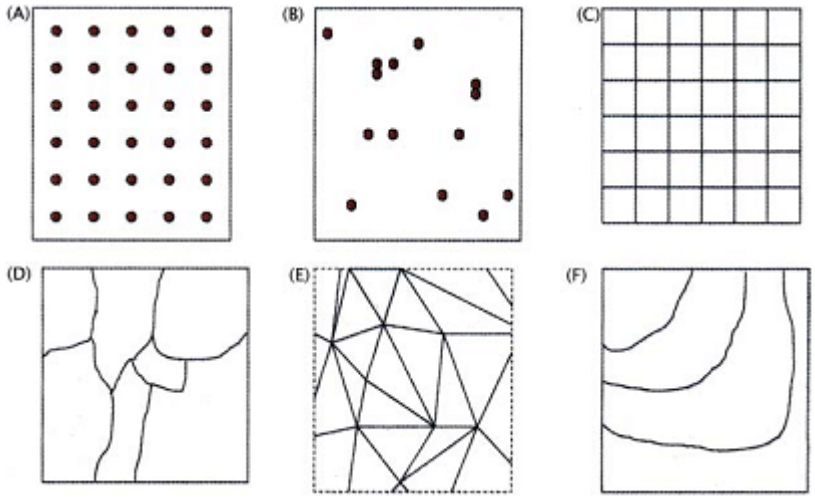


Figure 2.7: Six Approximate Representations of a Geographic Field. (A) Regularly spaced sample points. (B) Irregularly spaced sample points. (C) Rectangular cells. (D) Irregularly shaped polygons. (E) Irregular network of triangles (TIN). (F) Polylines representing contours. (Longley et al., 2005)

Out of all the options, Regular Square Grid (RSG) is, by a good margin, the most popular method; mostly due to the simplicity of the geo-referencing and the relative easiness of its digital representation (Figure 2.8). RSGs are sometimes also called as *raster* formats because of the similarity of its grids to the pixels in a computer display. In this sense, cells in a raster grid are often called as pixels.

For visibility analysis, although TINs allow faster and more efficient algorithms to be constructed, RSGs are again the preferred DEM method. This can be attributed to the relative simplicity of the constructed algorithms and the wider availability of RSG map data. Maloy and Dean (2001) found out that raster based visibility analysis was generally equal or better than the TIN based visibility analysis in terms of agreement with the field survey results. However the TINs in that study were produced using the same rasters, resulting in a regular spaced point structure for the TIN.

2.1.2 Available Digital Elevation Data

Almost all available digital elevation data is stored and distributed in a raster-based (RSG) format. One of the main distributers for digital elevation maps is *U.S. Geological Survey (USGS)* with its seamless data distribution system.

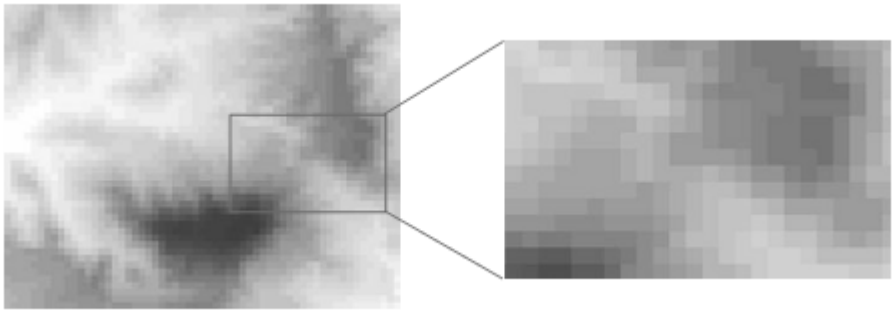


Figure 2.8: A sample raster representation with regular grids that use gray intensity to represent relative elevations. (de By, 2001)

USGS elevation data consists of *National Elevation Data (NED)* (Gesch et al., 2002) and *Shuttle Radar Topography Mission (SRTM)* (van Zyl, 2001). NED is available in 1, 1/3, and 1/9 arc second resolutions (USGS, 2010a). SRTM is available in 1 arc second resolution for the US data and 3 arc second resolution for the global data (USGS, 2010b).

NED data is a combination of many data sources and is available in *ArcGRID*, *GeoTIFF*, *BIL* and *GridFloat* formats with accompanying textual metadata. Measured *Root Mean Square Error (RMSE)* value for the overall absolute vertical accuracy was published as 2.44 meters.

SRTM dataset were collected by a radar system using the interferometric synthetic aperture radar technique covering 80% of the Earth's surface in 2000. The main difference of SRTM from NED is that the elevation from SRTM is canopy based (*i.e.,* DTM) whereas the elevation from NED is ground based (*i.e.,* DEM). The vertical relative height error of SRTM is specified to be less than 10 meters at 90% confidence level.

There is also another relatively new global elevation data available from *Advanced Space-borne Thermal Emission and Reflection Radiometer (ASTER)* project called *ASTER-GDEM* (NASA JPL, 2010b). It is comprised of 22,600 1°-by-1° tiles. ASTER GDEM is distributed in *GeoTIFF* file format with grids of 30 meters. RMSE estimation is 20 meters for vertical data and 30 meters for horizontal data at 95% confidence (NASA JPL, 2010a).

USGS also developed a special transfer file format for elevation data called *Spatial Data Transfer Standard (SDTS)* that includes spatial data, attribute, geo-referencing, data quality report, data dictionary, and other supporting metadata (USGS, 2010c).

2.1.3 GeoTIFF Data Format

GeoTIFF (OSGeo, 2010) has become one of the most popular data formats for storing raster image files with accompanying geographic information. GeoTIFF is an open, public domain, and non-proprietary format and it was developed at

NASA-JPL (Jet Propulsion Laboratory) with input from industry, starting from 1994. GeoTIFF is currently maintained by the Open Source Geospatial Foundation (`http://osgeo.org`).

GeoTIFF is based on the popular image format TIFF (version 6.0), which is a tag-based file format for storing and transferring raster type images. It is one of the few formats in the public domain that supports tiling, compression, and extensions for geographic metadata. GeoTIFF implements the geographic meta-data using the existing TIFF tagging mechanisms.

GeoTIFF supports two pixel value conventions. The pixel values are either the value of single point *postings* for the upper left corner of a pixel, or they are the average value for the whole of a pixel area. In this study, single valued GeoTIFF DEMs that have the upper left corner as their reference point are used. The reference corners for the pixels of such a DEM are illustrated in Figure 2.9

2.2 Visibility Analysis

Visibility analysis deals with the inter-visibility between points on a DEM (Figure 2.10). This is usually accomplished by a *line-of-sight (LOS)* between two points and is available within many GIS applications. Elevation offsets for both the target and the source can also be defined in a visibility analysis.

A natural extension to LOS is the ability to calculate visibility from a single source point to a set of target points. This creates a binary surface over the terrain showing visible locations; *i.e.,* a viewshed of the source point. The viewshed can also be interpreted as an answer to the question, "*where can the source be seen from?*". There may be situations or interpretations of visibility where the source can see the target, but it can not be seen from the target, as illustrated in Figure 2.11. An example viewshed can be seen in Figure 2.12.

An angular range can also be defined for the source, or observer point. Then, only the target cells falling into the defined field of view should be considered for the analysis. This property can be used to specify a direction of view for the observer.

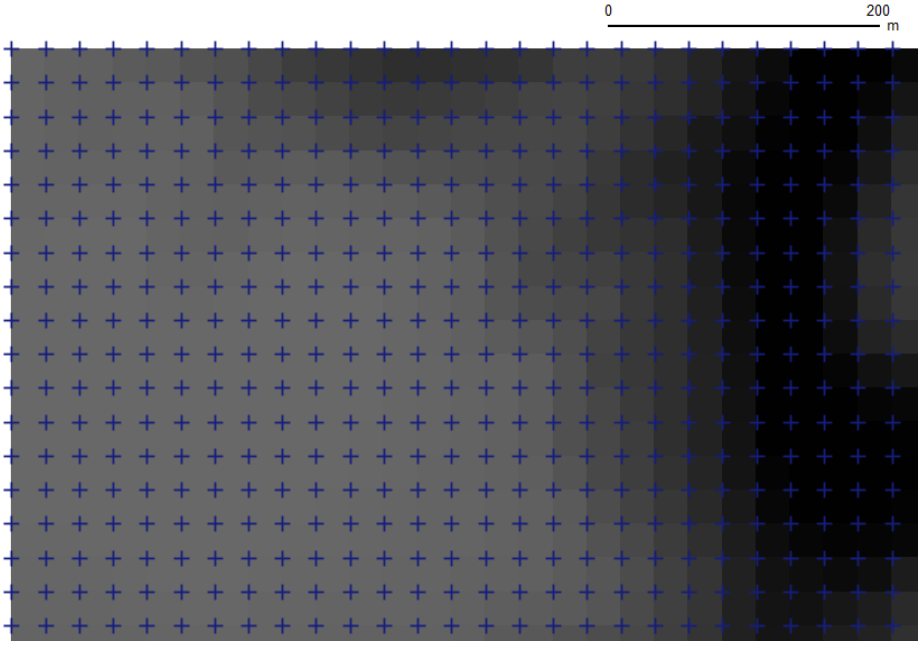


Figure 2.9: A sample GeoTIFF DEM map that has single valued pixels with a reference corner of upper left.

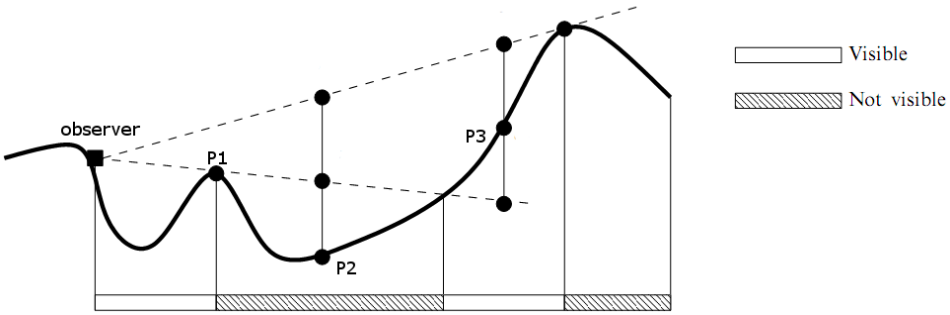


Figure 2.10: Viewshed cross section on a terrain, modified from De Floriani and Magillo (2003). Three points on the terrain are shown. Point 1 is on a local horizon and defines the start of the non-visible terrain section. Point 2 is at the non-visible section, while point 3 is at the visible section. The diagram also shows how the line of sights from the observer to the local horizon points define the limits of the visible and non-visible sections on the terrain.

Another common parameter is the restriction of the set of target cells. For example, a set of towers would form a set of individual and possibly unrelated cells. On the other hand, an areal target, like a lake, can be thought as the set of cells composing the lake. Similarly, a path-like target is a set of cells lying on the path. Target data can be provided as a vector layer of points, lines, or polygons. Another possibility is to use the elevation values of the raster data for a selection criteria, or to provide another raster map for that purpose.

One of first visibility calculation examples can be seen in the work by Travis et al. (1975). In that study, a computer program called VIEWIT has been developed that were able to calculate visibility, slope, and aspect data for an elevation map. Other early pioneering works were by Felleman (1979), Tomlin (1983), and Yoeli (1985).

2.3 Previous Works

Franklin and Ray (1994) investigated three algorithmic approaches to calculate visibility on raster based terrains which were called R3, R2, and Xdraw. R3, which has a cubic algorithmic time, was defined as the most accurate method for visibility analysis. To calculate the visibility of a point, R3 only uses the information previously calculated for the immediate preceding points. Several enhancements to R3 were described in order to decrease the complexity to squared times, namely R2 and Xdraw algorithms. These more efficient algorithms rely on the order of the points for analysis and introduce some approximations that reduce the accuracy of the results. R2 is a relatively complex algorithm and calculates the line-of-sights

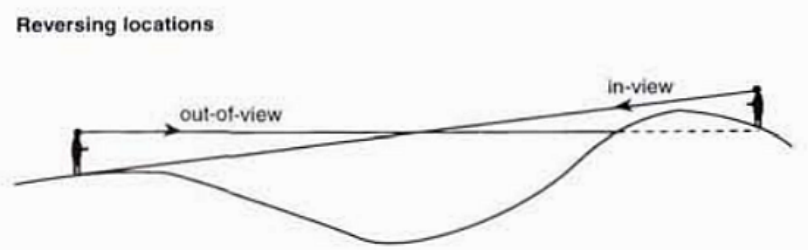


Figure 2.11: An example of one-way visibility situation (Fisher, 1996a).

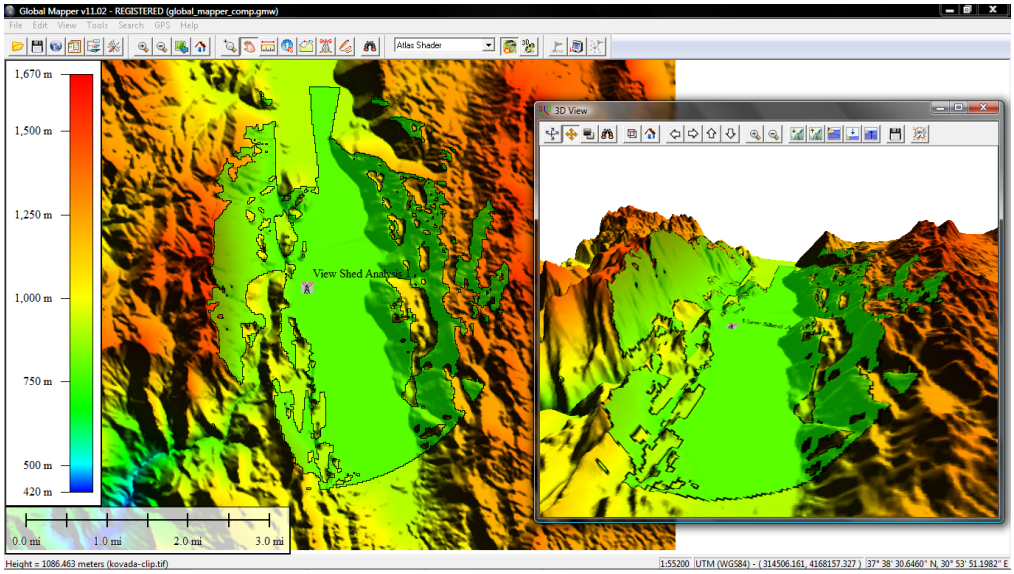


Figure 2.12: A viewshed example produced with the Global Mapper GIS software.

for only the perimeter points. The inner points use the closest line-of-sight passed through them for their calculation. Xdraw, proceeds by successive rectangular rings going outwards. Each ring only uses the information in the inner ring. Xdraw is the fastest of the three proposed algorithms; however it is also the one with the least accuracy. The main problem with Xdraw is that the visibility of a point may be affected by a remote unrelated point through the carrying of successive interpolations. While no comparison data was presented in the paper for Xdraw, the agreement between R3-Interpolated and R2 was given to be 98.8% while the computation time was reduced from n^3 to n^2.

For the software tool developed for this thesis, the visibility computations between a source and a target location are done by the use of the R3 algorithm mentioned in Franklin and Ray (1994). Although this algorithm is slower than the other proposed methods, it is preferred because of the high level of accuracy it provides. The calculation speed has turned out to be at acceptable levels for the purposes of this study as can be seen in Chapter 6.

Wheatley (1995) was one of the first researches that worked on the "cumulative viewshed analysis" problem which is one of the cornerstones of this thesis. In

that study, the relationships of inter-visibility between related archeological sites within an area have been investigated. The source set was consisting of the selected sites, while the target set was the entire region. In that sense, it was a multiple point source multi-visibility analysis. The visibility surfaces were gathered for two different site sets using the map algebra functions of IDRISI GIS package. Also, the inter-visibility information between the sites was produced. In order to test the statistical significance of the results, a one-sample Kolmogorov-Smirnov test was undertaken for each area. The test showed that one of the archeological sites tested (the Stonehenge barrows) had statistically significant inter-visibility among the barrows.

Fisher (1996a) proposed alternative viewshed functions in order to deal with specific queries such as forest-fire observation, visual impact of a new structure, and route determination. The alternative methods involve the categorization of targets as local and global horizons. Local horizon for an observer is the farthest point before the first invisible point. Global horizon is where the land surface meets the skyline. The offset between a line-of-sight (LOS) and these horizons can be used to answer these problematic queries. Also, an error modeling was performed based on the spatially autocorrelated simulations using DEM errors. A local offset viewshed can be transformed to a binary viewshed with a given vertical offset. This offset can be the rise of smoke for forest fire observation problem. Local and global offsets can also be helpful in determining the maximum allowed height of a proposed building for some given locations. Concealment of a target which can be defined in terms of its visibility against the skyline can also be analyzed using the global offset viewsheds. Another advantage of these new functions, are that they can answer the height limits for visibility situations without the need for a recalculation.

Nackaerts et al. (1999) studied the effects of errors in the DEM map to the viewshed results. They compared a 10m contour interval topographic map of ancient city of Sagalassos in Turkey with a DEM interpolated from the 50 m interval digitized maps to investigate the error characteristics. The results showed that error was distributed with a mean error of 0 m and a standard deviation of 10 m. For viewpoints ancient watch tower locations were used with a 10 m height and 5 km

view radius. The authors applied Monte Carlo simulation method based on the error distribution data to produce different sets of simulated DEMs, and then calculated ordinary Boolean viewsheds on these maps. Every cell value in the viewshed got multiple binary values from different simulations and the combination of these values gave a probabilistic visibility value for the cell. As probabilistic data is more difficult to use than a Boolean viewshed, the authors also presented a method to produce a binary viewshed from a probabilistic viewshed. That method allowed them to map different probabilistic visibility threshold values for binary viewsheds with corresponding user's and producer's accuracy values.

Bishop (2003) reviewed GIS-based (2.5D) and 3D visual assessment studies. The first tools for visibility have been developed during the seventies. It is noted that some of the features of those early attempts were seldom found in contemporary GIS. Several studies involving visual quality, impact analysis, moving objects, and atmospheric haze were discussed. Moreover, studies dealing with the extensions to binary viewsheds and the movement on paths were summarized. Finally, 3D-based visibility analysis studies were presented and they were proposed as an alternative to GIS-based approaches in visual impact and quality analysis.

Ervin and Steinitz (2003) investigated the distinction between what is *visible* and what is *seen*. The secondary measurements of the viewshed were discussed; such as its total area, longest reach, roughness measure of its perimeter, the presence and number of *islands*, and the aspect ratio of its major and minor axes. Also, more human related aspects of visibility, like psychology and cultural perception were discussed. The authors emphasized the value of path-based visibility for landscape planners and designers and stated the need to develop specific methods for it. This methodology was partly inspired by that specific recommendation.

Caldwell et al. (2003) used the the Complete Inter-visibility Database (CID) which stores pre-computed viewsheds for every point on a raster map to aid visibility related decision-making process. Two types of products were described: descriptive metrics and tactical decision aids. Descriptive metrics included cumulative visibility, slope of cumulative visibility, fragmentation, core area visibility, and ratio of cumulative visibility to core area visibility. Tactical decision aids included

percentage of a target that was visible and the least/most visible routes. The CID generation for a 466×336 pixels elevation map took approximately 44 hours on a multi-computing architecture having 13 processors.

Kim et al. (2004) studied the optimal site selection problem in terms of total viewshed ratio which is usually absent in current GIS software. The aim of the optimal site selection is to find the best observer locations that when combined, give the largest viewshed area possible. They studied a version of that problem where the number of observer points is fixed (v). Then, optimal site selection is a combinatorial problem with a computational complexity of $O(nv)$, which has a very high computation time restricting its applications in any practical situation. To overcome this problem, the authors proposed two methods. First is using a small number of candidate sites rather than searching for all the possible locations. The second is to use a number of heuristic algorithms with relatively small computation times. For heuristic algorithms, they applied three different approaches that were originally used for location-allocation type problems: swap algorithm, genetic algorithm, and simulated annealing algorithm. On a 40x40 pixel sample DEM map, these two methods cut down the computation time two orders of magnitude but at the cost of a 10% loss in total viewshed area. While the simulated annealing produced the best results, the gains over swap approach was only marginal and the relative computational cost was too high.

Wu et al. (2006) analyzed the perception of visual quality through the use of several impact factors in GIS. Their method extensively used the visibility information with distance factors to weight the visual quality perception at a given source location. The eight features affecting the quality were determined as: pasture, other vegetation, shore, creek, road, building, slope, and sea. Also, the height of mapped surfaces were superimposed over the ground DEM layer to include the buildings, canopy, etc. Distance factor was taken as 1/d. A table of factor indices were constructed which was the basis for the methodology. Visual quality coefficients for the factors were generated with the help of regression modeling techniques and opinions of human observers about the photos of various landscape locations in the

area. Finally, all of the study area has been assigned a visual quality score based on the spatial interpolation of the sample locations by the use of kriging method.

Although the method used by Wu et al. (2006) employed many different factors, the results were for single point locations and it focused on the perception of scenic beauty only. The method in this thesis aims for a more general and simpler methodology that is more suitable for movement on directional paths and also that can be used for many other application areas aside from the perception of beauty.

Riggs and Dean (2007) studied the error sources contributing to viewshed analysis. They compared a field surveyed viewshed with GIS software generated predicted viewsheds by the use of a survey-generated TIN surface which was converted to raster data and a USGS 10m DEM data. They categorized the errors as DEM errors, spatial resolution related errors, and algorithm based errors. The effects of errors in the DEM data were analyzed by the addition of pseudo-random spatially auto-correlated noise surfaces. This showed that substantial errors in viewshed results can be caused by relatively small errors in the source DEMs. The accuracy of predicted viewsheds as compared to the field survey viewshed was shown to be at best 85%. The agreement between GIS software was between 89% and 93% on visible regions. The lack of documentation describing the assumptions and the processes for major GIS viewshed tools was stated as a problem. The authors also remarked that floating-point arithmetic in computers is inherently imperfect leading to some errors in the computational results. A surprising finding was the failure of TIN-based analysis approaches in producing more accurate results which was attributed to the weakness of algorithms for TIN surfaces.

Lu et al. (2008) studied the problem of finding the least visible path (*LVP*) on a terrain which is a type of least cost path planning with visibility as the cost surface. They proposed a new method that considered the visibility correlation of adjacent points. Path based visibility were calculated using accumulative total of the visibility indices of the path points. The new method provided a path with fewer points in the terrain that can see the path than the traditional method.

CHAPTER 3

METHODOLOGY

3.1 Methodology Overview

The methodology developed in this study incorporates three different approaches into standard visibility analysis. These approaches can be named as *multi-visibility for paths*, *weighted targets*, and *directional visibility*. Although these approaches have been individually researched in some degree before, this study aimed to use them in a more collaborative way in order to visually and quantitatively assess the quality of visibility along a path, or the 'visibility value' of a path.

A supportive idea to the methodology is to automate the process as much as possible using GIS based software tools. Since the available free and open source tools provided limited solutions in this aspect, a new, more unified tool has been developed for this study. Software development process is described in detail in Chapter 4.

Although the steps should be applicable to any type of DEM data, this study focuses on the raster based (RSG) terrain models. Furthermore, the source and targets for the analysis are restricted to the set of grid points, or the cells on the raster map. All inter-pixel locations use the value of the nearest pixel locations. Otherwise, the computation needs for such inter-pixel positions and related interpolations would considerably slow down the speed of calculation.

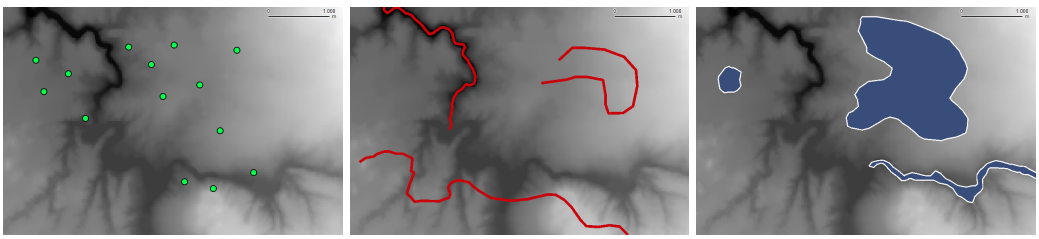
Simple binary visibility forms the basis for the analysis. All further enhancements sit on that foundation. For the software, an accurate algorithm that finds whether

two points are inter-visible or not is created in the first place. The software is developed as a plugin to the open source and free Quantum GIS package. Although it can readily calculate line of sight profiles by the use of other plugins, Quantum GIS currently does not have the ability to calculate full binary viewsheds.

3.2 Multi-Visibility for Paths

Multi-visibility, or cumulative visibility, simply combines many simple binary visibility results from a set of points. It can be called as the multiple-source version of a viewshed. The source set can be multiple distinct points, lines, or areas as seen in Figure 3.1. When the source set is a line or an area, a suitable set of points on the line or inside the area can be chosen as the source points. The suitable set of points can be any set that reasonably defines the line or area and depends on the specific goals of the study.

For raster data, one reasonable way of choosing source points would be to select all points corresponding to the underlying raster pixel locations. This kind of operation is usually called as the '*rasterization*' of a vector data. Rasterization uses specific algorithms to find the correct raster positions matching with the continuous vector lines or areas. It is used extensively in the computer graphics field. One of the most popular rasterization algorithms is known as the Bresenham's algorithm (Bresenham, 1965). An example rasterization process on a DEM map can be seen in Figure 3.2. The cells in this DEM have point-type values located on their top-



(a) (b) (c)

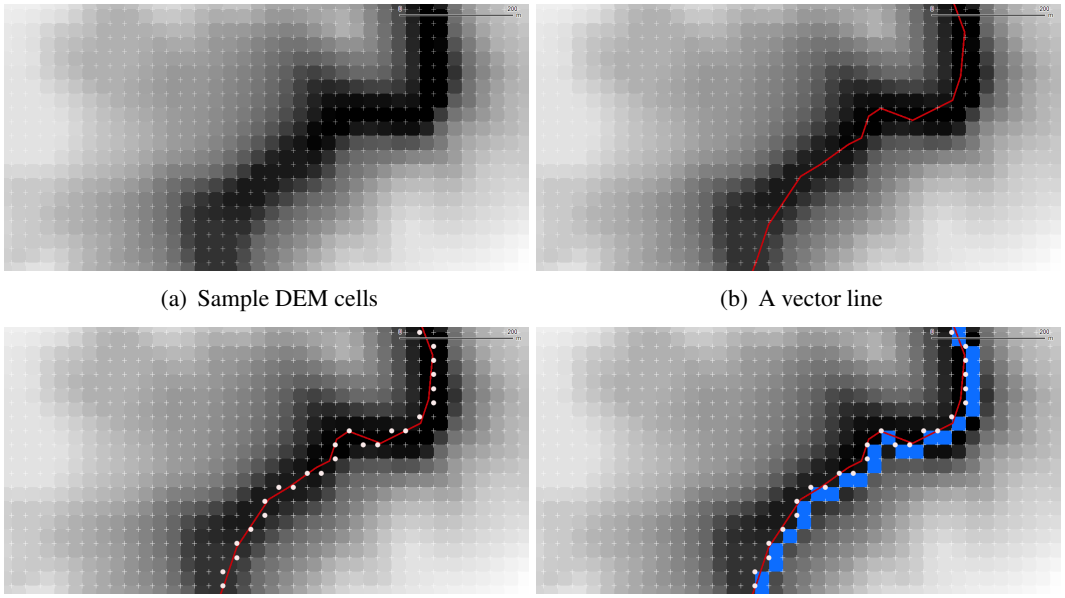
Figure 3.1: A DEM map showing sample multi-visibility sources for points (a), lines (b), and polygons (c).

left corners. Therefore, the rasterization algorithm selects the cells based on their top-left corner locations.

The source set for this study consists of one dimensional vector lines representing a real path on the terrain. The vector can be constructed on the middle line of a path having a specific width. The vector path was treated as a multi-point set of source points selected by the rasterization algorithm. Then, the binary viewsheds calculated for the selected cells on the line can be combined in a number of ways.

Some of the combination alternatives can be illustrated with an example source set consisting of 8 discrete points (Figure 3.3). One simple combination alternative would be the set union of the visible points. In the set union, all the visible points of all the source viewsheds can be seen in a unified way as seen in Figure 3.4(a). Another alternative is using the set intersection on the sets and display only the visible points that exist in both viewsheds (Figure 3.4(b)). If we are interested in the locations that can be seen from only one of the observers but not both, we can use the symmetric difference (*i.e.,* set union minus the set intersection) of two sets (Figure 3.4(c)).

Although these set operations can be useful for certain types of selection queries, they are usually ineffective in answering aggregate questions. For example, if we are interested in how many cells are affected by any kind of overlap for these 8 sample points, we can use the set intersection on every pair and then perform a set union on these intersections to get the overlapped cells. But this would be an unnecessarily complicated operation. Instead, let's assume we have a visibility count (C_i) for every cell indicating how many times that particular cell was seen. Then, a simple selection operation with $C_i > 1$ would give us the correct result. There are many situations that can take advantage of visibility counts. It can also be used to calculate a quantitative measure of overlap for a given configuration by a simple addition of the visibility counts of the cells. However, one of the biggest advantage aggregation provides is the opportunity to display the counts as a pseudo-surface overlayed on the actual map. These types of aggregate surfaces can be called as *visibility surfaces*.



(a) Sample DEM cells

(b) A vector line

(c) DEM cell corners selected by the rasterization process

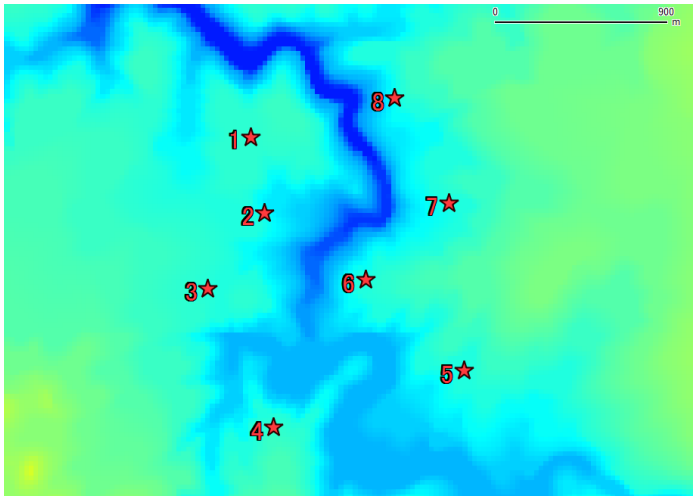
(d) DEM cells describing the selected corners

Figure 3.2: An example rasterization process using an optimized Bresenham's algorithm on a DEM having top-left-corner based point-type cells.

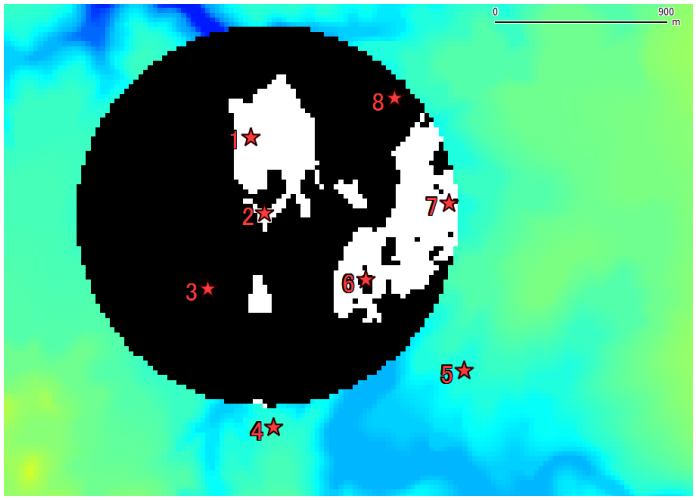
In this study, visibility counts and visibility surfaces were used for the multi-visibility operations. A visibility surface for the given 8 observers can be seen in Figure 3.5. Since there were only 8 observers (and only 5 viewsheds that overlap in maximum), the surface has not much detail for this example.

When every cell position in a DEM map is taken both as an observer and as a target, a *total viewshed* is produced (Figure 3.6). Total viewshed can be useful in analyzing the general visibility characteristics of an area. It also usually reveals very rich details about visibility that were otherwise hidden. However, boundary effects should be taken into account when assessing total viewsheds.

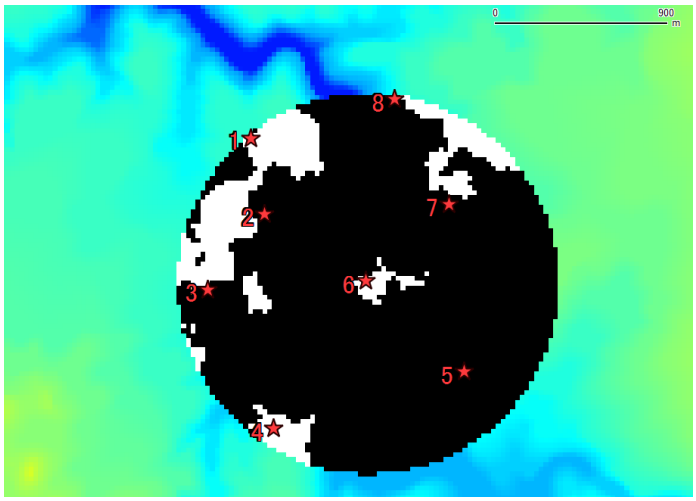
When the source set is a line, the values accumulated for each target cell can be interpreted in different ways, besides "*how many times it has been seen*". If the line is considered as an object, the question would transform into "*how much of the target can see a part of me?*". Another possibility would be to see the line as a designated path for a point type observer. This time, the question would be "*how*



(a) Sample points

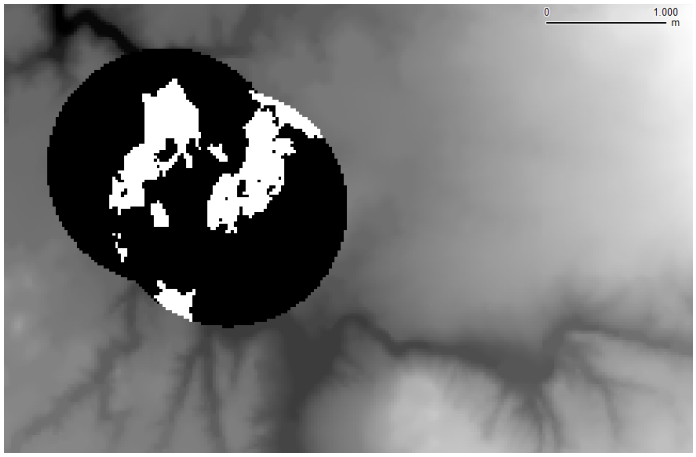


(b) Point 2 viewshed

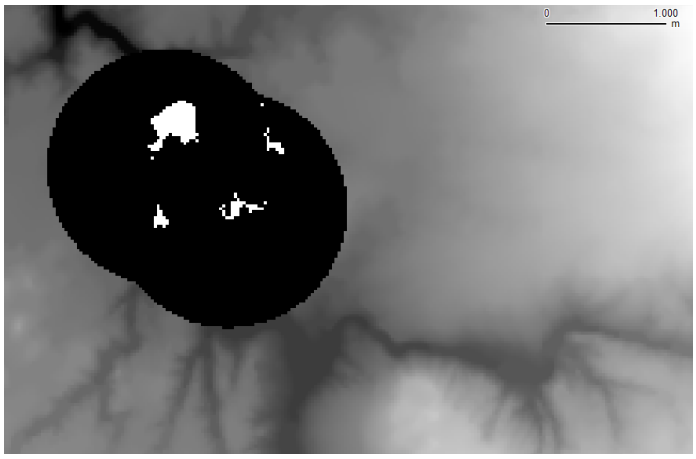


(c) Point 6 viewshed

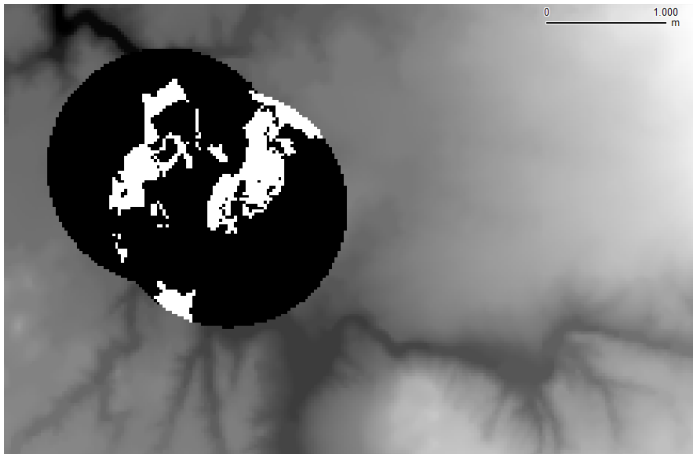
Figure 3.3: Eight discrete observer points (a) and the viewsheds for the point 2 (b) and point 6 (c) within 1 km radius of the observers. White cells indicate the visible areas.



(a) Union of viewsheds for points 2 and 6

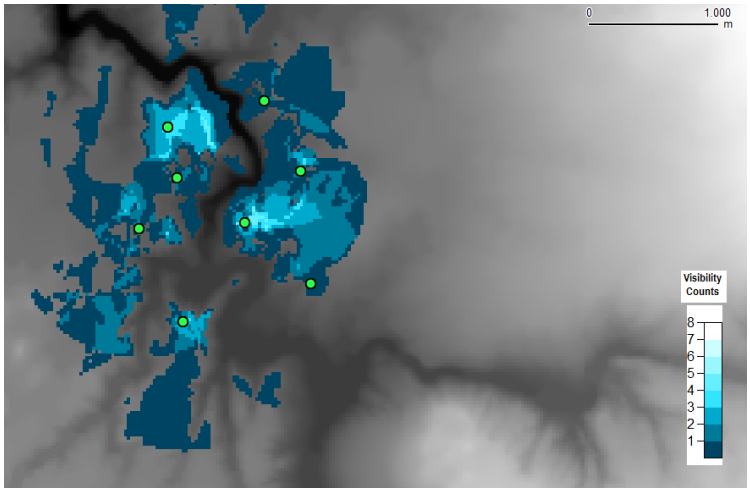


(b) Intersection of viewsheds for points 2 and 6

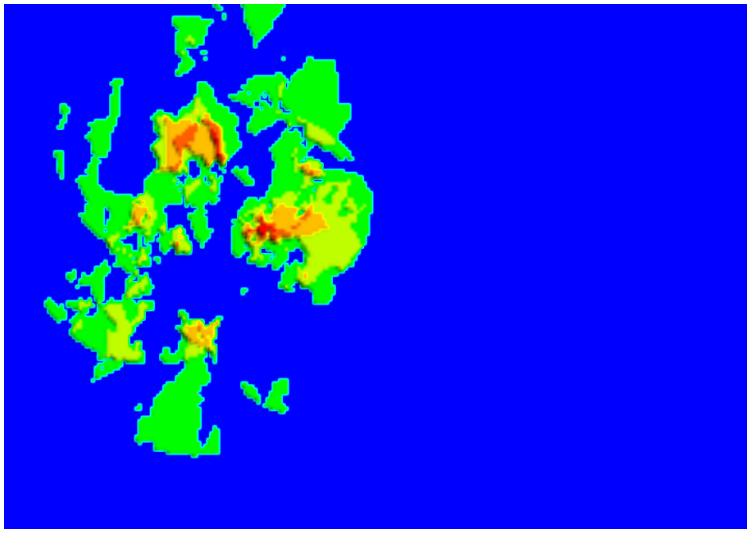


(c) Symmetric difference of viewsheds for points 2 and 6

Figure 3.4: Some combination alternatives for a multi-visibility situation: union (a), intersection (b), and symmetric difference (c).

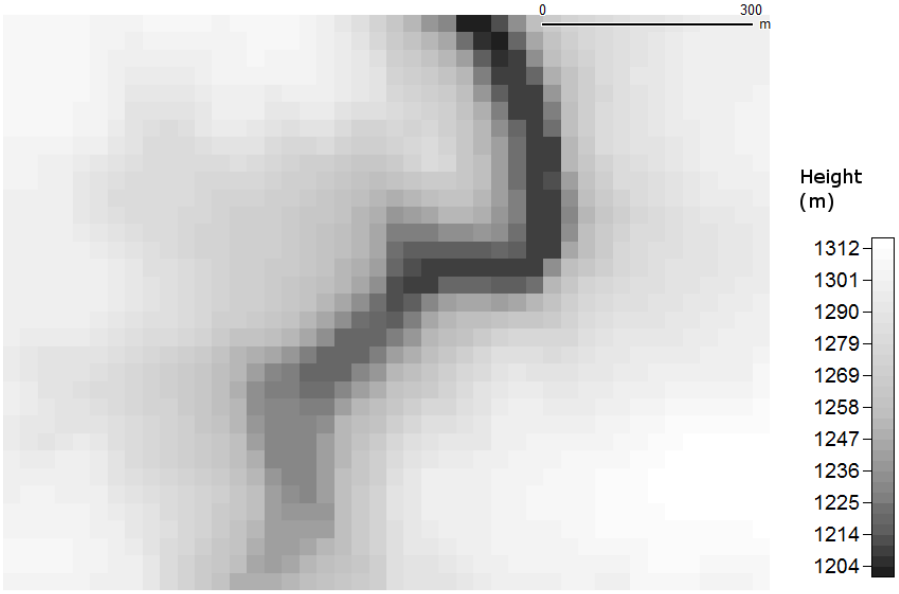


(a) Color shaded

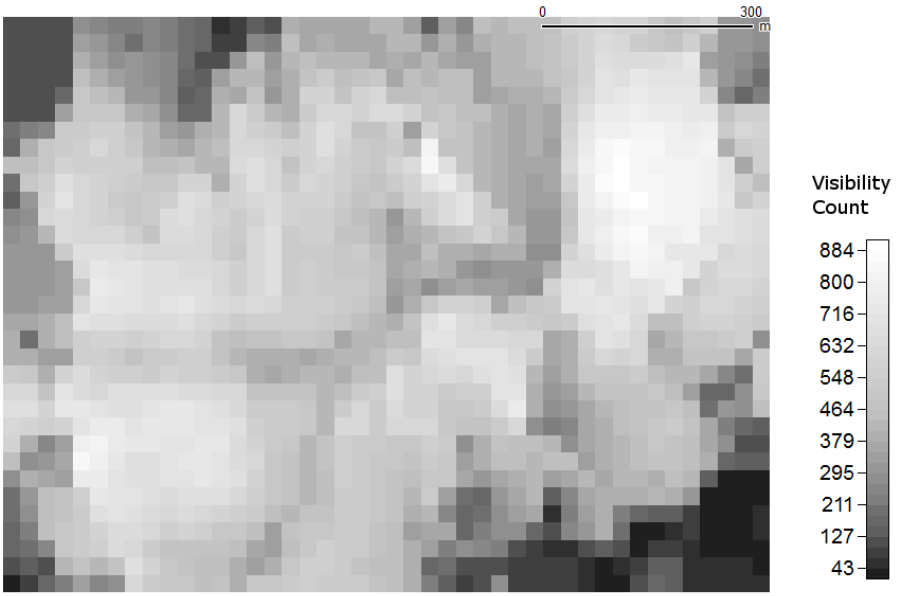


(b) Relief shaded

Figure 3.5: Visibility surface for the multi-visibility of 8 observers.



(a) Real elevation of the area



(b) Total viewshed result

Figure 3.6: Total viewshed example for a small area that has 44x33 cells.

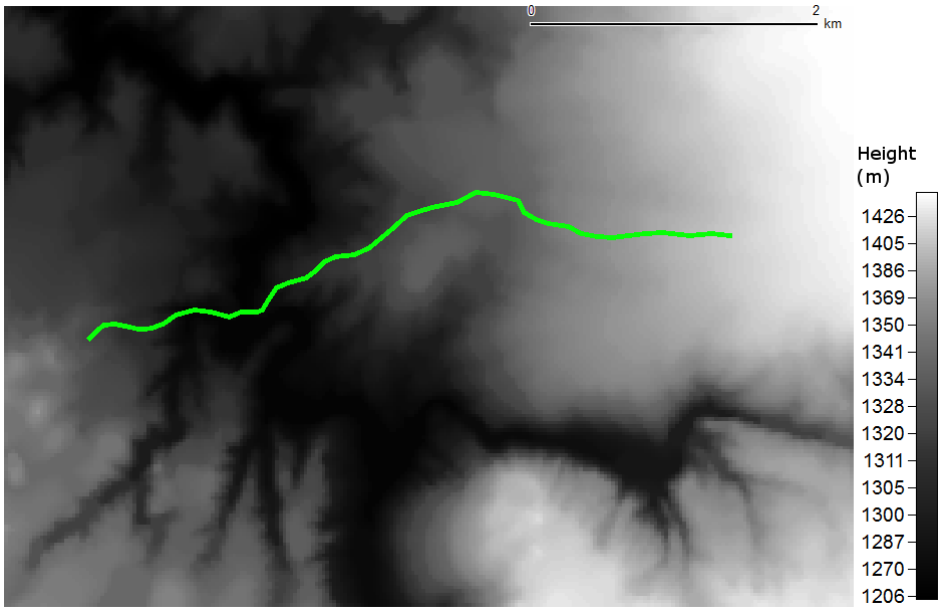
often can the observer see me?". In this study, that last scenario has been taken into account.

The path visibility surface is illustrated in Figure 3.7. The sample path is approximately 4.9 km in length and the observer is assumed to be 2 m in height above the ground level. The path covers 183 cells of the DEM map as given by the Bresenham's algorithm. In total, 19,705 unique (and cumulatively, 913,242) target locations were analyzed for this example. The minimum value was 0 and the maximum value was 69 with a mean of 14.06 and a standard deviation of 14.80. The visibility surface can be seen to form virtual sharp ridges and peaks around the real valley slopes and dips around the bottom of real valleys. Also, it usually forms large virtual plains around locations having high elevations.

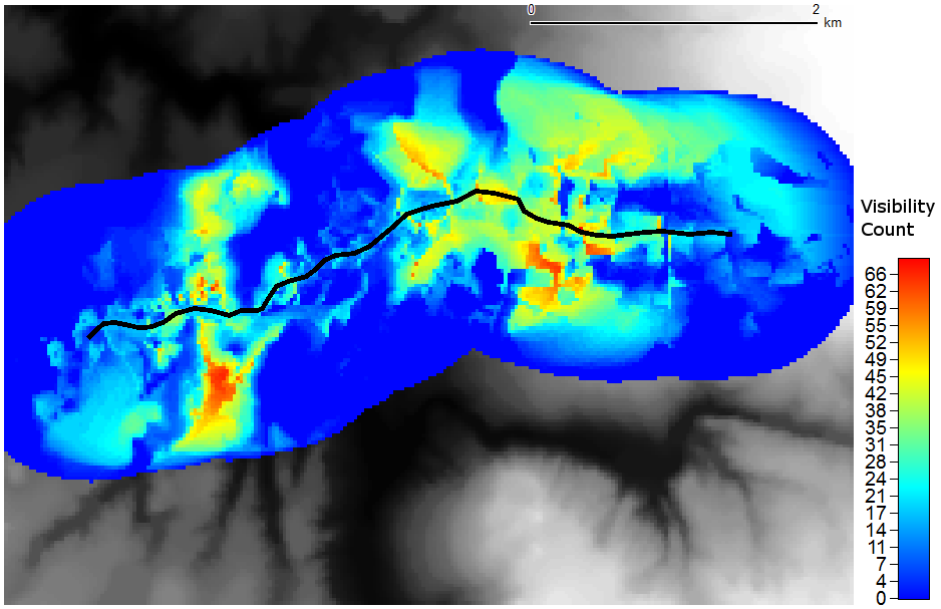
The aggregations up to this point have all been about the target cells. However, we can also do the inverse, *i.e.,* use a count for the source cells. These cumulative values would indicate the number of target cells visible from each source cell. With path-type sources, these inverse visibility counts indicate the amount of visible area for every point on the path. An inverse visibility count example for a path was given in Figure 3.8. The minimum value for the cumulative path counts was 673 and the maximum value was 2,555 with a mean of 1,502.71 and a standard deviation of 499.32.

We can define a *total path value*, as the sum of all the inverse counts for the observer. In the given example, total path value was 277,071. Since there are 183 cells on that path, we can find an *average path value* for it; which would be 1,514.05 for this particular example.

These two methods ensure that every target and every source cell can individually be assigned a cumulative visibility value. In this study, both of these methods are used together to analyze the visibility value of a path.



(a) Sample path



(b) Visibility surface of the path (color shaded)

Figure 3.7: A sample 4.9 km length path on a terrain (a) and its color shaded visibility surface within a 1 km radius of the observer (b).

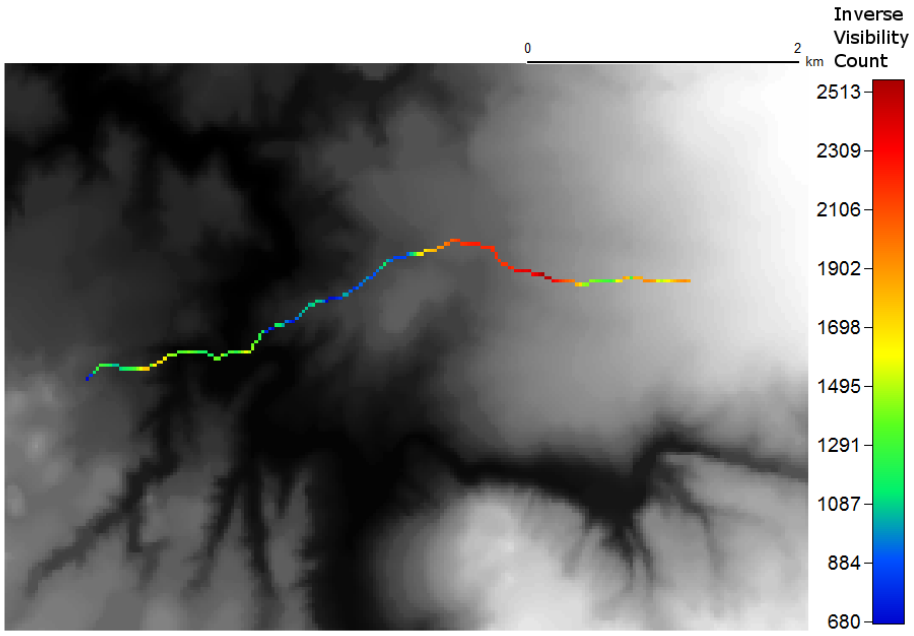


Figure 3.8: Inverse visibility count for the sample path displayed by corresponding raster cells.

3.3 Weighted Targets

In the standard multi-visibility, cumulative values are calculated by a simple addition of visibility counts. This is acceptable for the situations where we are only interested in the counts or when the target cells all have equal importance. However, when the visibility state of some locations are considered more important than the others, this simple approach may not be powerful enough.

Through the definition of importance factors (or weights) for specific target zones, multi-visibility can be greatly enhanced. Then, weights (W_i) for those zones may be used to indicate the desirability of visibility for them. If we assign the weight of 1 to the non-weighted cells, we can define more preferable or important cells by giving them a weight such as $W_i > 1$. For less preferable or less important cells, the weight would be defined such that $0 < W_i < 1$. An ineffective cell can have a weight with $W_i = 0$. Furthermore, we can define negative weights; the points that

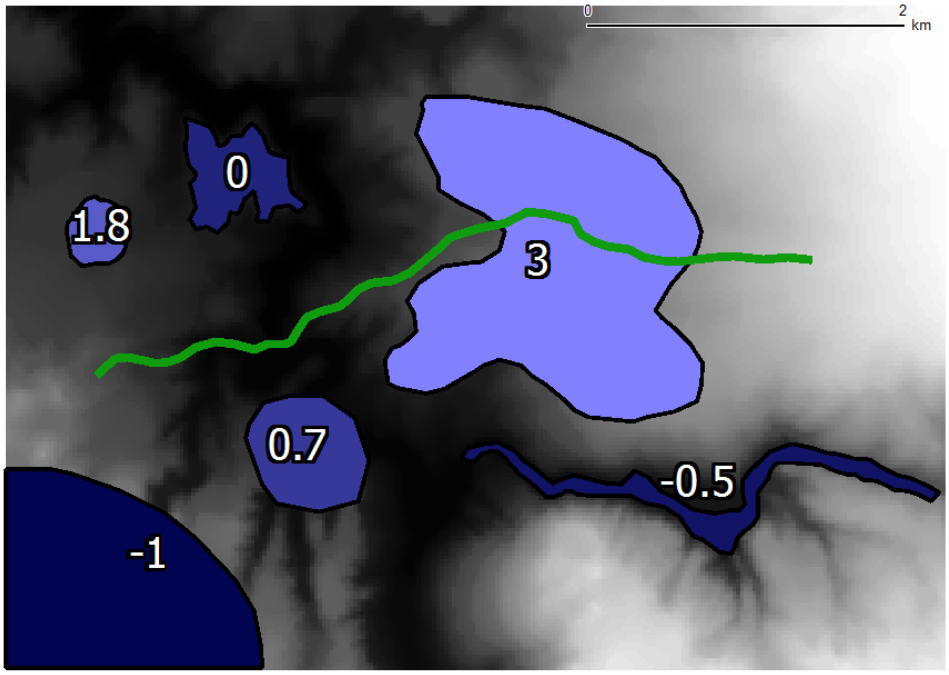
are undesirable with $W_i < 0$. Such weighted visibility surfaces, in a more primitive form, have been proposed even in the earliest works such as Travis et al. (1975).

The weights can also be defined for the source cells, but this scenario is not in the scope of this study which focuses on the weights defined for the target cells. Weights can be taken into account for just the target surface, for just the source surface (which is a path in our case), or for both of them. For this study, weights have been taken into account for both of them.

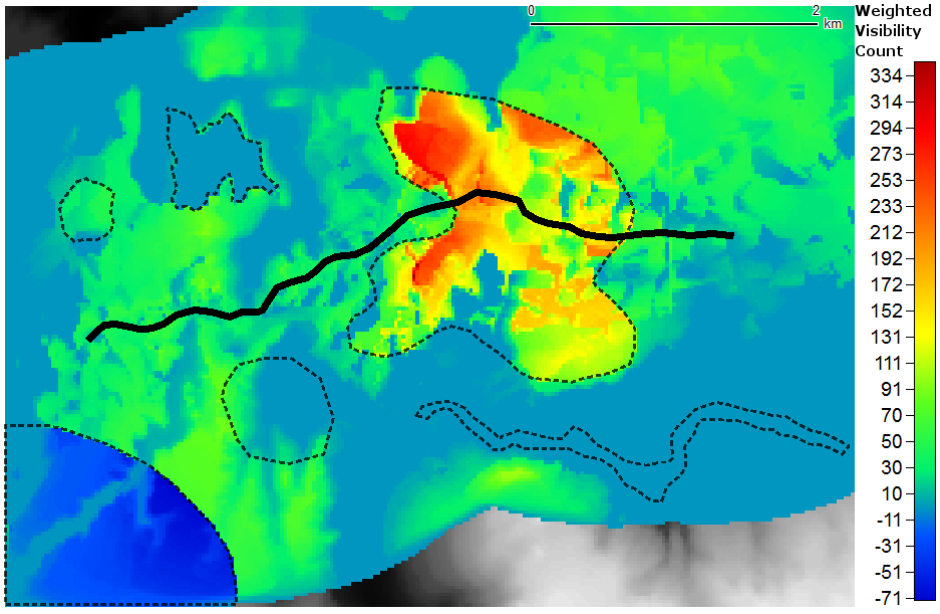
The resulting weighted visibility surface for the targets can be used to see the weighted effects of seeing a particular target point from a particular path. Similarly, the weighted inverse visibility surface (or count) for the path can be used to see the important observer locations having more preferred views. Then, we can calculate the *weighted total path value* and the *weighted average path value* quantitatively for any path and compare different paths with each other.

A simple way to define weighted zones in a GIS environment would be to describe them by vector polygons. In Figure 3.9(a), various weight zones are illustrated with different weights. The visibility surface for our previous path was calculated with the same parameters and a 2 km radius. But this time, the weight zones for the targets were also taken into account. The results can be seen in Figure 3.9(b). The resulting surface was clearly affected by the weight zones. Relief shaded version that are shown in Figure 3.10 provide strong visual information about the weighted (or preferred) visibility structure of the sample area. Minimum value for the surface was -73, the maximum value was 345, and the total target cell number was 36,274 for this example.

In the weighted inverse visibility count, the weighted total path value was 1,068,425.6 and the weighted average path value was 5,838.39 for 183 observer locations. The weighted inverse visibility surface is given in Figure 3.11.



(a) Weight zones



(b) Weighted visibility surface

Figure 3.9: Sample weight zones defined as vector polygons (a) and the weighted visibility surface for the path with a radius of 2 km (b).

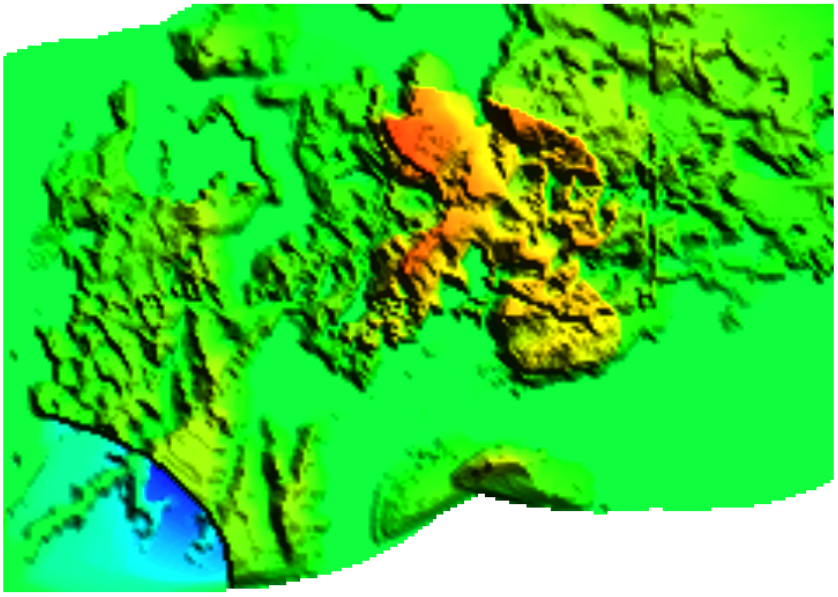


Figure 3.10: Relief shaded view of the weighted visibility surface given in Figure 3.9.

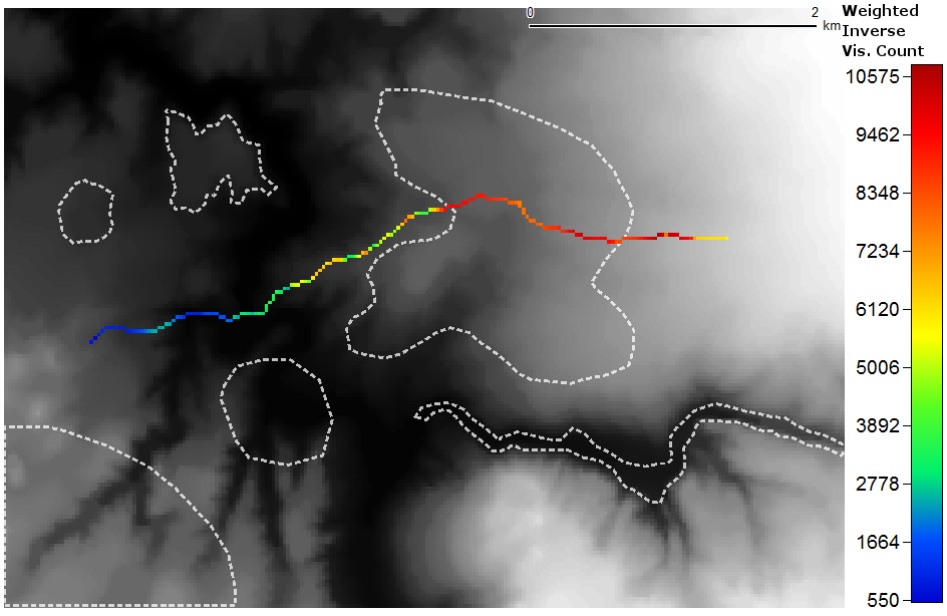


Figure 3.11: Weighted inverse visibility count for the sample path displayed by corresponding raster cells.

3.4 Directional Visibility

Another parameter that can be included for every observer point is angular range of the view, or field of view (FoV). Angular range can be defined with a direction of sight and an angle of view (Figure 3.12). The direction of sight is the bisector of the view angle.

When there are multiple observer locations as in a multi-visibility situation, the direction and view angle information should be defined for all of the observers. If the source set is a line, it would be reasonable to give a single direction and angle for the set of points lying on that line. In our scenario, we assume a moving observer on a given path. If we also assume that our observer has a fixed angle of view, we can define an automated way of finding the field of view for every point on the path. The directions for the points can be derived from the tangent line to the path at that specific location. This requires a direction to be defined also for the path itself by designating one of its end points as the *origin* point. The directions and the field of views have been illustrated for a couple of sample points on a path going from left to right in Figure 3.13. With directional visibility taken into account, the multi-visibility surfaces of two opposite directions would differ from each other whenever the view angle is smaller than 360°.

In Figure 3.14, the effects of direction on the visibility surface is shown for our path with a view angle of 60°. The height of the observer has been increased to 20 meters in order to have more visible targets and the distance limit was defined to be 1 km. The difference can be seen more easily from the shaded versions of the surfaces as illustrated in Figure 3.15 and Figure 3.16.

A more thorough comparison between the two directions is given in Table 3.1. The table clearly shows there may be great differences in the multi-visibility analysis for opposite directions of the same path. It can be said that the left to right direction is about 18.70% more preferable (according to the average path values) than the opposite one, assuming that more visibility value is being aimed and if edge effects are not taken into account.

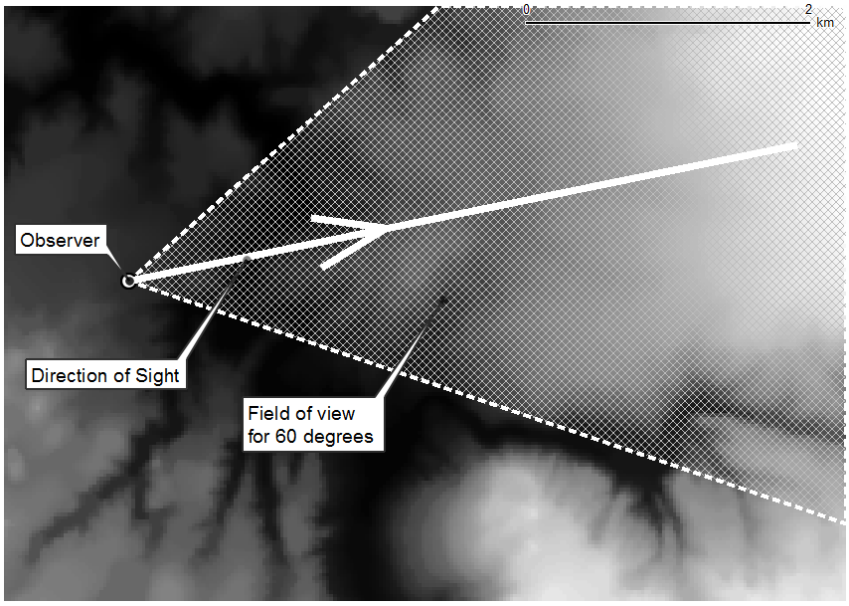


Figure 3.12: Directional field of view for 60° of view angle.

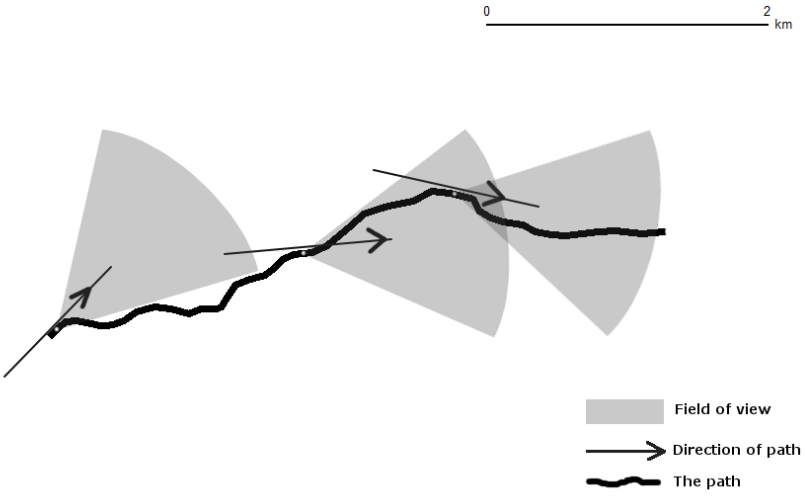
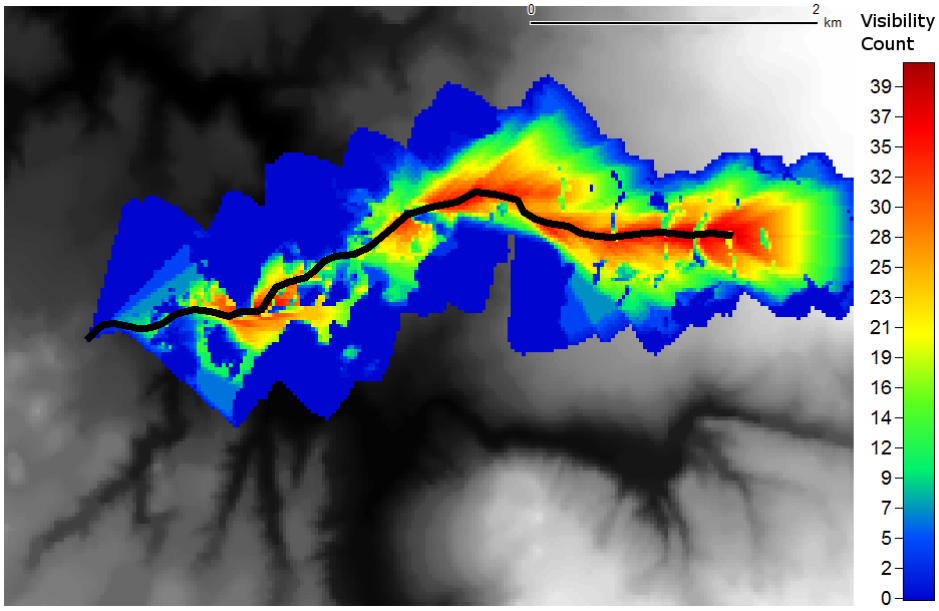
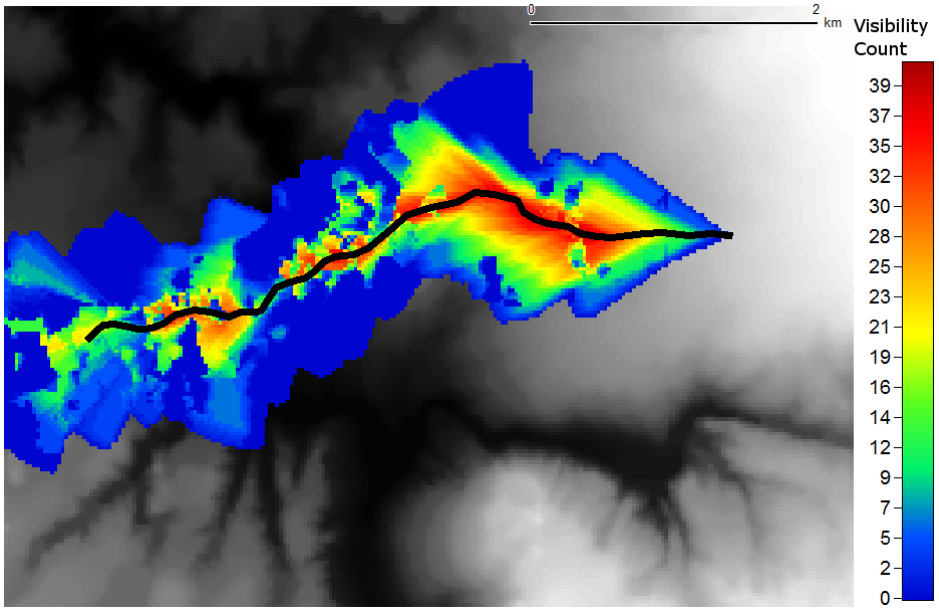


Figure 3.13: Directional field of view on a path with a constant view angle and dynamic directions.



(a) Directional visibility surface from left to right



(b) Directional visibility surface from right to left

Figure 3.14: Directional visibility surfaces for a path shown in both directions (left to right in (a) and right to left in (b)). The constant view angle for this particular example was 60°, the height of the observer was 20 meters, and the distance limit was defined to be 1 km.

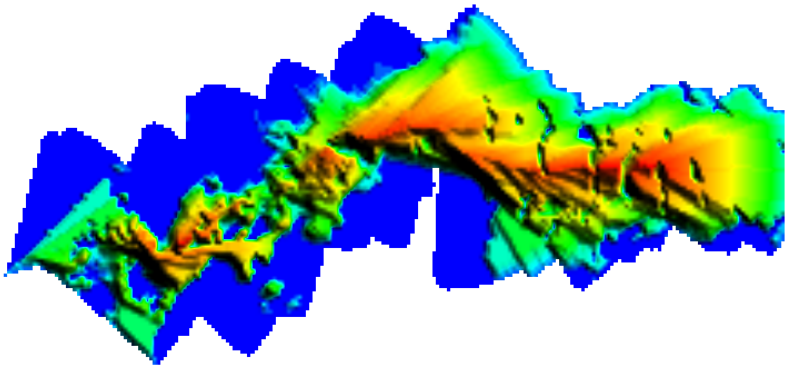


Figure 3.15: Relief shaded version of Figure 3.14(a).

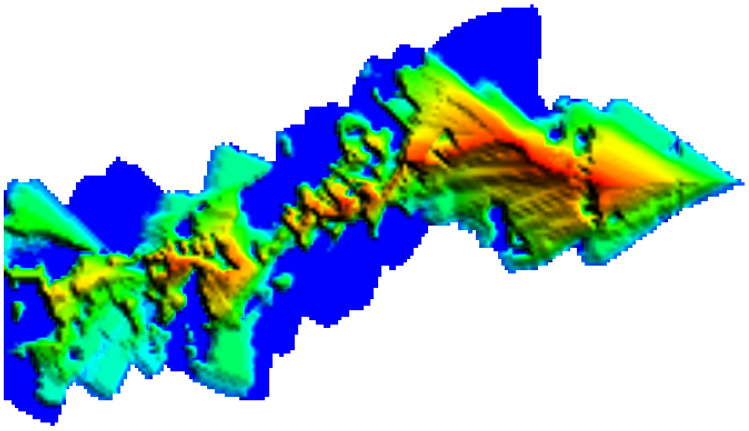


Figure 3.16: Relief shaded version of Figure 3.14(b).

Table 3.1: Comparison of multi-visibility analysis for different directions on the same path.

	Left to Right		**Right to Left**	
	Visibility Surface	**Observer Path**	**Visibility Surface**	**Observer Path**
Number of Cells	11,489	183	10,599	183
Minimum Value	0.0	67.0	0.0	24.0
Maximum Value	41.0	828.0	41.0	822.0
Mean Value	9.04	562.98	8.25	476.00
Standard Deviation	10.45	217.27	9.89	227.44
Total Path Value	-	103,845.00	-	87,486.00
Average Path Value	-	567.46	-	478.06

3.5 Combined Approach and Visualization

The approaches mentioned in this chapter have been combined in this study in a supportive way to define a general methodology for path-based multi-visibility analysis. The combined approach can be used to assess and compare the '*visibility value*' of different paths for specific situations.

This methodology can be used to model observers with limited view angles and ranges that are moving on directional paths on a terrain with weighted zones. The weights can be modified to model different scenarios and preferences.

The resulting weighted visibility surfaces and weighted path value profiles can be visualized at the same time. These can provide a visual perspective in the assessment of the visibility properties for a given scenario.

Besides the visual outputs, quantitative results, such as the *average path value*, can be used to formally analyze different models and to compare them with each other.

3.6 Flowchart

The general steps of the methodology are illustrated in Figure 3.17. The flowchart includes two main loops: one over the source path, another over the target cells. A

third inner loop is also present for the calculation of the inter-visibility between two locations as part of the specific algorithm for that.

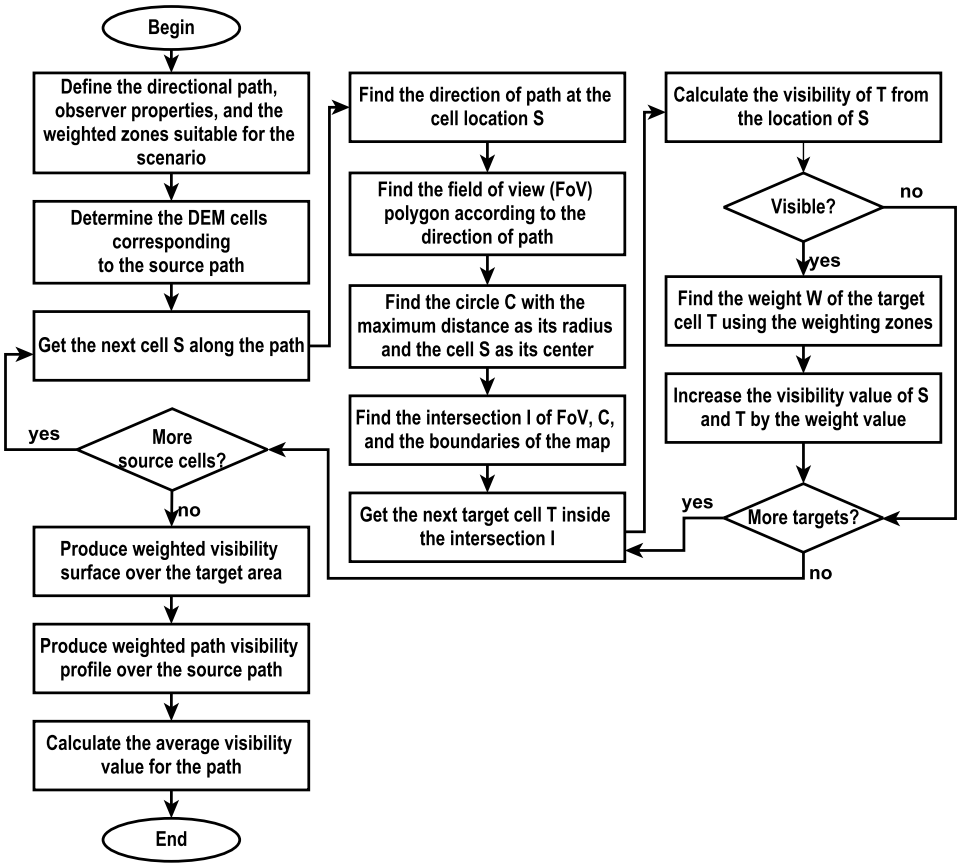


Figure 3.17: General flowchart for the methodology.

CHAPTER 4

SOFTWARE DEVELOPMENT

4.1 Development Platform

As geographical information systems become more and more capable in their features while finding wider application areas, available software development options have been multiplied. Software developers are now able to choose from many GIS libraries and host environments.

In a general sense, there are two main alternatives for desktop based GIS application development. In the first one, a developer builds everything from scratch using other third party libraries as necessary. There are many useful commercial and non-commercial GIS libraries dealing with different aspects of the domain. Many GIS applications also provide their software capabilities as a series of code libraries.

The other alternative is to use a host application that has a plugin mechanism, and develop a software plugin for that host environment. The plugin, in this scenario, can access to the user interface and features of its host application. This approach simplifies development and makes the software easily available for the already existing user base of the host platform.

In this study, the second approach was followed on the Windows™ version of the popular Quantum GIS software (`http://www.qgis.org`). Quantum GIS project was founded in 2002 by Gary Sherman as a free and open-source software. Over the years, it has become one of the most popular GIS tools with its many new features and its ability to run on multiple platforms (Sherman, 2008). It provides a plugin

mechanism that supports extensions written in C++ and Python programming languages. Python (http://www.python.org) was preferred for the development of the plugin as its interpreted nature and advanced dynamic data manipulation features proved to be highly useful in the development process.

Plugin development for Quantum GIS requires several libraries (Quantum GIS Development Team, 2010). To match the development environment with the dependencies of the host platform, version 2.5 of Python was installed with the Win32 extensions as opposed to the latest version (2.7). A special Python module named NumPy was used to handle array operations necessary on the raster data. Since Quantum GIS uses Qt windowing toolkit from Nokia (http://qt.nokia.com), special Python bindings of Qt was needed for the plugin. This binding has been provided by the PyQt project (http://www.riverbankcomputing.co.uk). For the geometry related operations such as polygon intersections, Quantum GIS provides the necessary interfaces by the use of the Shapely (http://pypi.python.org/pypi/Shapely) package for Python. As editor, free Komodo Edit software was used as it could easily manage special whitespace requirements of Python. Another important library used was GDAL/OGR Python bindings (http://www.gdal.org), which handles many vector and raster data types including GeoTIFF. A list of software used for the development can be seen in Table 4.1.

Table 4.1: Software setup for the development environment of the plugin.

Software	**Version**	**Installation File**
Quantum GIS	1.5.0 (Tethys)	QGIS-OSGeo4W-1.5.0-14109-Setup.exe
Python	2.5.2	python-2.5.2.msi
PyWin32	214	pywin32-214.win32-py2.5.exe
PyQt	4.7.4-1	PyQt-Py2.5-gpl-4.7.4-1.exe
Komodo Edit	5.2.4	Komodo-Edit-5.2.4-4343.msi
NumPy	1.4.1	numpy-1.4.1-win32-superpack-python2.5.exe
GDAL/OGR	1.6.1	GDAL-1.6.1.tar.gz & gdalwin32exe160.zip

4.2 Viewshed Comparison

The algorithm used to calculate the inter-visibility between two points is based on the R3 method of Franklin and Ray (1994) as mentioned in Section 2.3. The accuracy of the whole methodology mostly depends on the accuracy of the implementation of the R3 algorithm. To investigate this, a sample binary single-point viewshed was calculated and then compared with the results obtained from commercial Global Mapper 11 (`http://www.globalmapper.com/`) software.

An observer height of 100 meters and a distance limit of 4 kilometers were used for the comparison viewshed for the same location (Easting: 312319.8057, Northing: 4168645.815, Zone: 36N in UTM). As the output of Global Mapper is in vector format, the raster output for the implementation was also converted to vector representation for comparison. The viewshed area obtained from Global Mapper (Figure 4.1) and the implementation (Figure 4.2) show similar characteristics. Figure 4.3 shows the overlapped area and the non matching parts. The viewshed areas were 22,578,750 m^2 for Global Mapper and 21,741,875 m^2 for the implementation. The agreement between the two is 90.97% based on the ratio of the intersection area to the union. This value is within the margin that Riggs and Dean (2007) found (89%-93%) in their comparison of visibility calculations from different GIS software. The small difference among the results can be attributed to the specific assumptions made by different software like the value point of the raster cells, whether discrete or continuous analysis has been done, the effect of the floating-point calculations, etc.

4.3 Software Features

The main dialog of the plugin can be seen in Figure 4.4. The plugin works on the loaded layers of Quantum GIS. Suitable DEM files from the current project is listed to be chosen as the *elevation layer*. Also, a descriptive text for the analysis can be given by the user. The other main inputs for the plugin are the source and target selection areas.

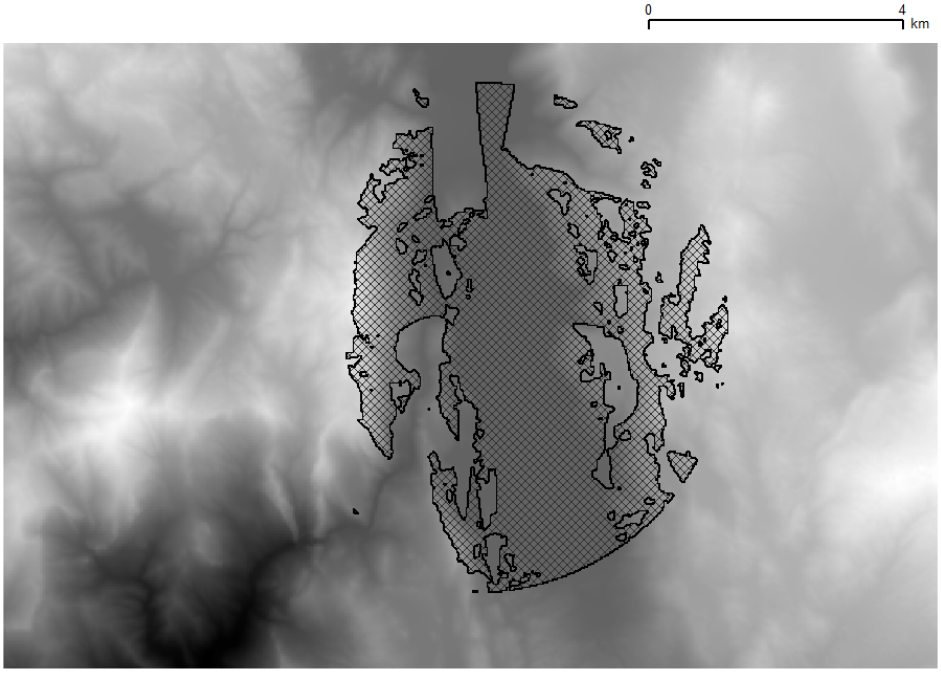


Figure 4.1: A sample single-point binary viewshed result calculated by Global Mapper 11 software.

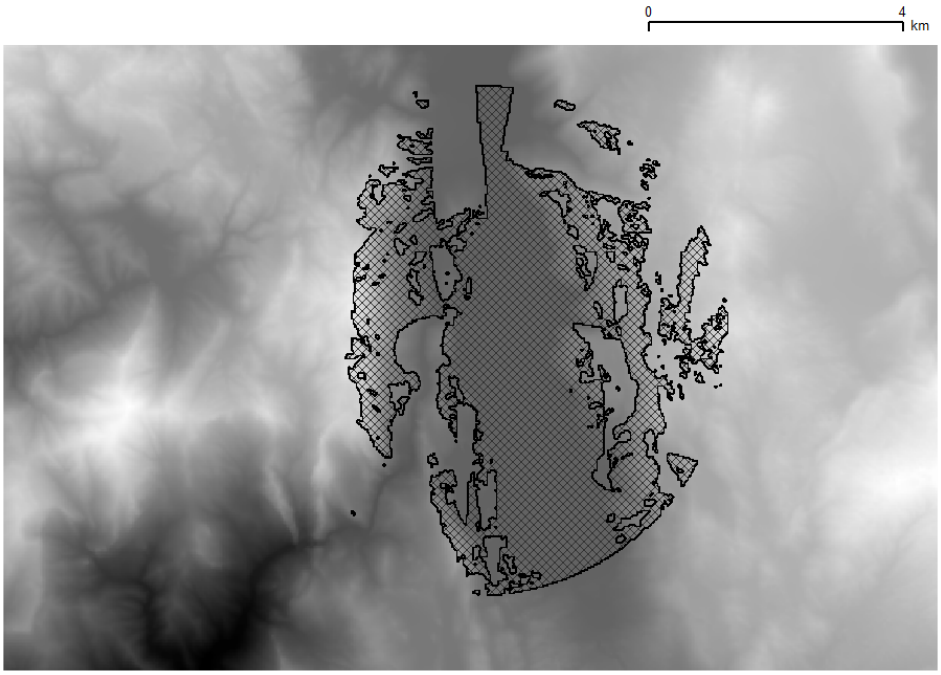


Figure 4.2: The same sample single-point binary viewshed from Figure 4.1 as calculated by the algorithm implemented for this study.

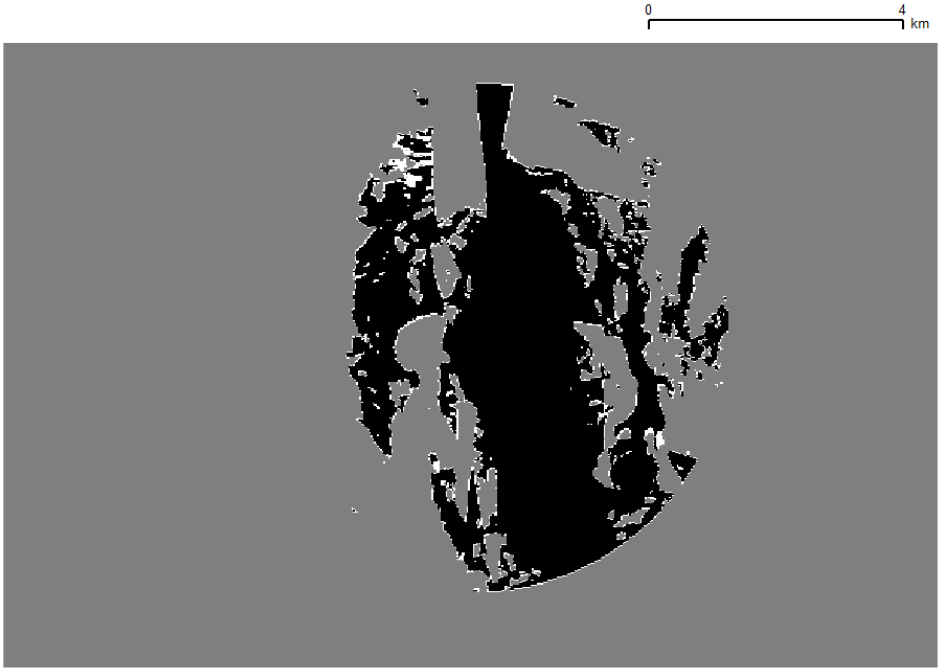


Figure 4.3: The overlap map showing the intersection area of the viewshed regions from Global Mapper and the implementation. The overlapped area is 90.97% of the union of results. The black regions are the overlapping parts and the white regions are the differences. The DEM map has been replaced with a gray background for better contrast.

In the source selection area, a path layer can be chosen. Only the line type vector layers are listed here. The user should select a path in the layer by the selection tool of the host application. The plugin can only operate on one path at the same time. Observer height value can be also given here in the positive map units. The ends of the selected path are marked with the letters 'A' and 'B' by clicking the "*Display A-B on map*" button. The user then can select one of the end points as the starting location of the path. A view angle can also be given in this area as degrees or by selecting "*No restriction*" for 360° views.

Target selection area defines the target zone and the weighting criteria. A distance limit for the observer can be given here in map units. A suitable polygon layer for the weight zones can also be chosen. The decimal type attribute fields of the chosen polygon layer are listed for the selection of the weight value field. A checkbox allows the user to limit the targets only to the areas inside the polygons. Finally, a

distance factor can be entered here as an additional weighting effect. The distance factor is used to diminish the weight by a value proportional to the distance of the target from the observer. The formula used in the plugin was "$w = w_o/2^{(d/f)}$", where w is the new weight, w_o is the original weight, f is the distance factor, and d is the distance in map units. Effectively, the weight is halved whenever the distance reaches a new multiple of the distance factor.

Once all the information is entered, analysis can be started by clicking the "Analyze" button. Through the analysis, a convenient blue circle is displayed along the path indicating the current observer location as seen in Figure 4.4. When the visibility analysis is complete, two new raster maps are created with the same settings as the original DEM map. The calculated visibility surface and the path profile are stored in these raster maps. The new layers are automatically loaded into the current project. Also the calculated total and average path values are written in a results file. A sample project result can be seen in Figure 4.5.

4.4 Implementation Details

The core of the visibility calculation in the plugin is the *visibility* function. The *visibility* function simply calculates the visibility of a target from an observer location. It accepts the locations that were given in the raster coordinate system (*i.e.,* pixel counts from the top-left corner of the raster data). The *visibility* function is defined as:

```
1  def visibility(self, rObserver, rTarget):
2      (rObserverX, rObserverY) = rObserver
3      (rTargetX, rTargetY) = rTarget
4
5      lineIter = bresenham(0, rObserverX, rObserverY, rTargetX, rTargetY)
6      try:
7          lineIter.next()
8      except StopIteration:
9          return True
10
```

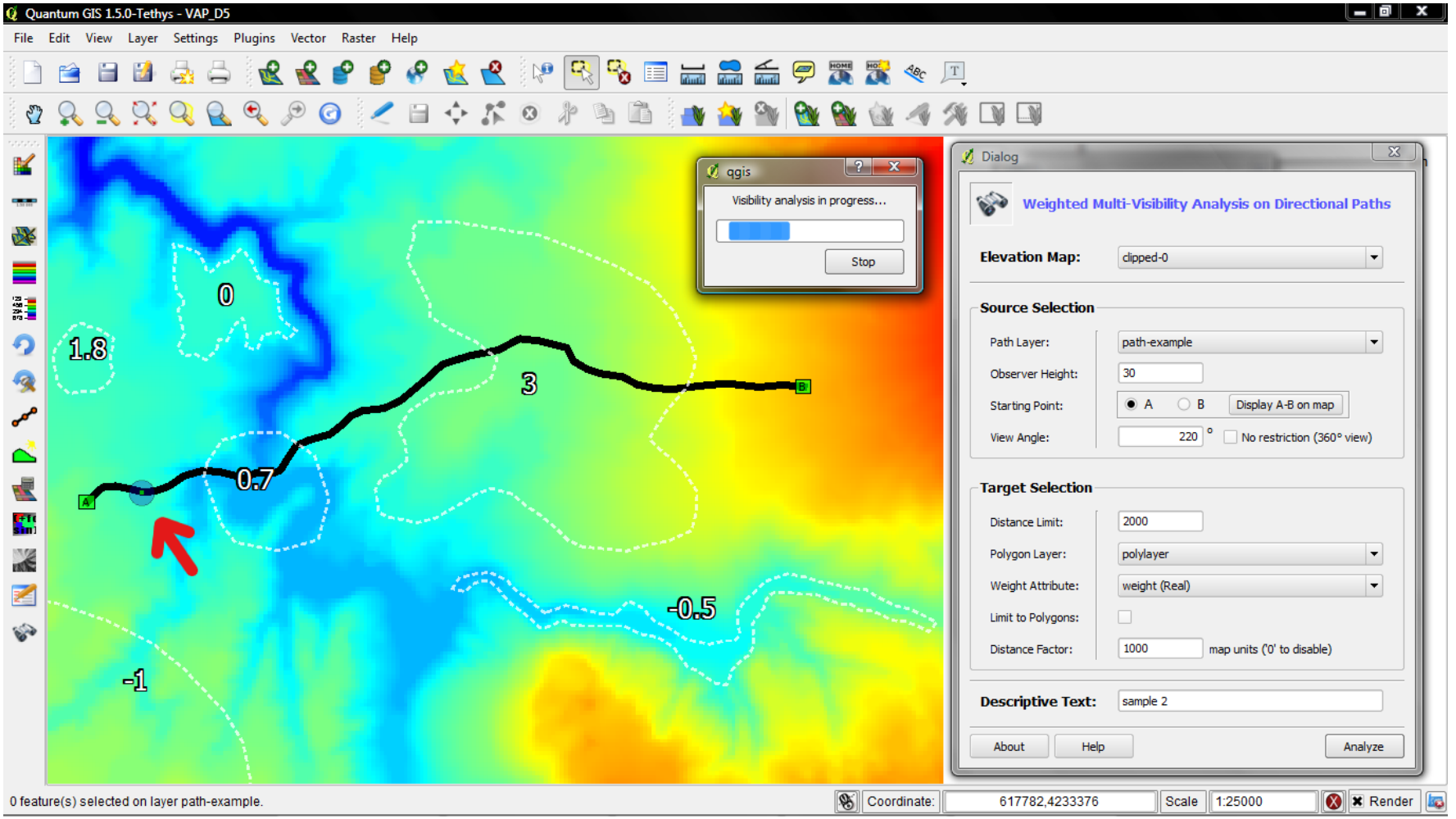


Figure 4.4: The main interface and the marker position during an analysis process. The red arrow shows the marker for the current observer location.

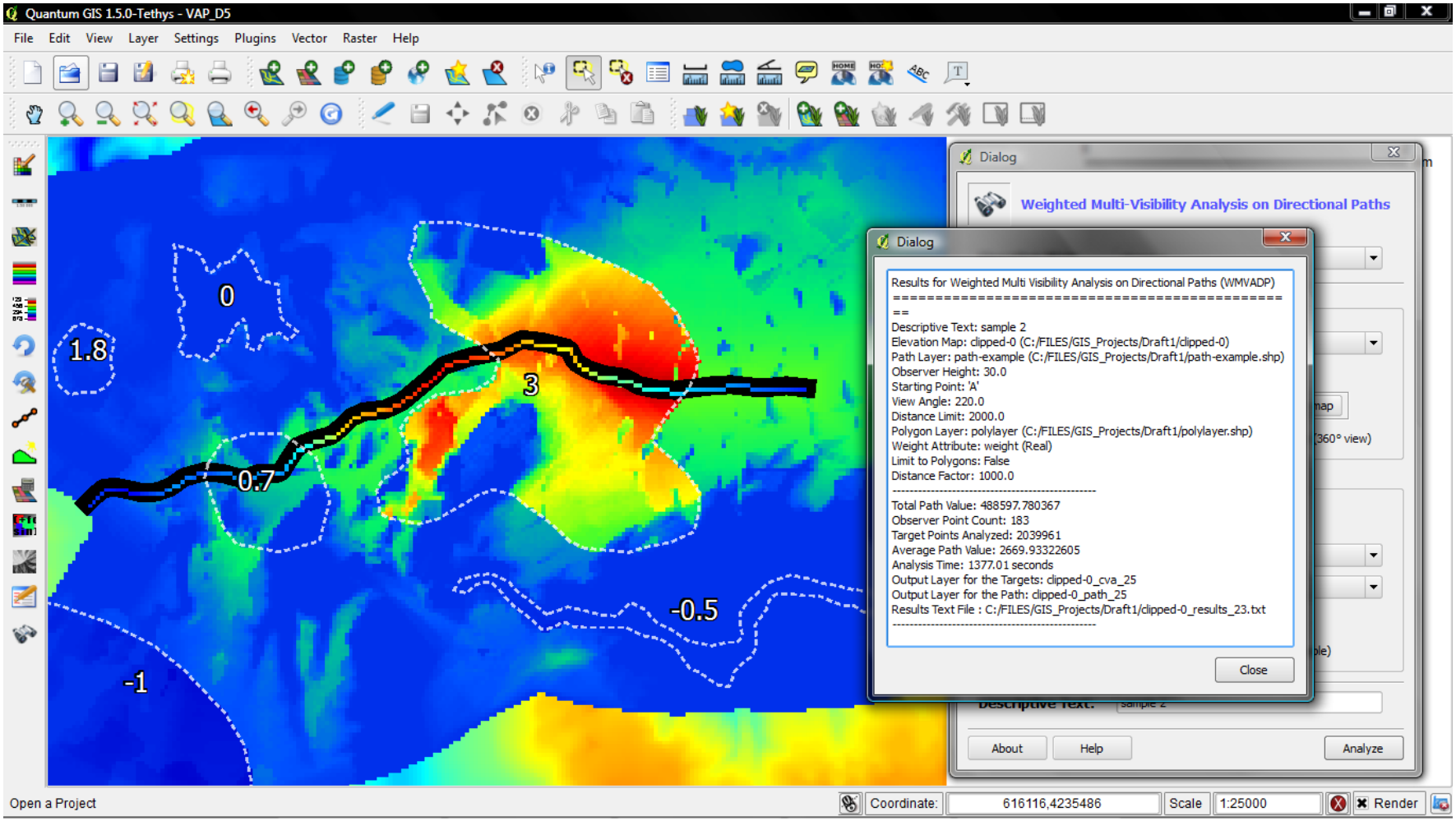


Figure 4.5: A sample project view after an analysis.

```
    delta = max(abs(rTargetY - rObserverY), abs(rTargetX - rObserverX))

    observerGroundElev = self.rasterIO.getValue(rObserverX, rObserverY)
    observerElev = observerGroundElev + self.currentObsHeight
    targetElev = self.rasterIO.getValue(rTargetX, rTargetY)

    losElevStep = self.calculateStep(observerElev, targetElev, delta)
    losElev = observerElev
    for (x, y) in lineIter:
        losElev += losElevStep
        rValue = self.rasterIO.getValue(x, y)
        if rValue > losElev:
            return False

    return True
```

This function starts with the calculation of the 2D line of sight (on the x-y plane) using Bresenham's algorithm to generate a Python iterator function. Bresenham's algorithm is defined by:

```
def bresenham(includeTarget, x0, y0, x1, y1):
    steep = abs(y1 - y0) > abs(x1 - x0)
    if steep:
        x0, y0 = y0, x0
        x1, y1 = y1, x1
    reverse = x0 > x1
    deltaX = abs(x1 - x0)
    deltaY = abs(y1 - y0)
    error = int(deltaX / 2)
    y = y0
    if y0 < y1:
        yStep = 1
    else:
        yStep = -1
    if reverse:
        r = xrange(x0, x1-includeTarget, -1)
    else:
        r = xrange(x0, x1+includeTarget)
    for x in r:
```

```
20          if steep:
21              yield (y, x)
22          else:
23              yield (x, y)
24          error -= deltaY
25          if error < 0:
26              y += yStep
27              error += deltaX
```

This function is optimized to use strictly integer-based calculations in order to increase the processing speed. It uses an integer *error* value to determine the discrete coordinate changes for the raster axes. This function also takes advantage of the *generator* feature of the Python language which allows the lazy generation of the raster pixels.

The *visibility* function then calculates the elevation change per pixel on the line of sight. The elevation of the intermediate locations on the path is compared with the elevation of line of sight on that location. If the elevation of the location exceeds the elevation of line of sight, the point is blocking the sight and the function returns *false*. Otherwise, if no intermediate pixel blocks the sight, the target is determined as visible.

The methodology translates into two nested loops in the main algorithm for the plugin. The outer loop iterate over the observers, while the inner loop iterate over the targets for the current observer. The observers and targets are defined in the discrete raster coordinate system. In the plugin code, the outer loop has been implemented as two nested loops. The outer loop finds the straight line segments of the path and the inner loop finds the observers in the current line segment. The *viewFieldFromSegment* function uses geometry capabilities of Quantum GIS to find the polygon that defines the view field based on the observer location and the direction of the line segment. The main algorithm for the plugin can be listed as:

```
1  for rSegmentCoordsF in rPathSegmentsIter:
2      rSegmentCoords = map(int, map(round, rSegmentCoordsF))
```

```
    rSegmentPointsIter = bresenham(0, *rSegmentCoords)
    for rObserver in rSegmentPointsIter:
        totalObservers += 1
        vObserver = mtp.toMapCoordinates(*rObserver)
        rasterIO.addPathValue(rObserver, 0)
        if not fullView:
            viewFieldPolyGeo = viewFieldFromSegment(
                                              rObserver, *rSegmentCoordsF)
        rTargetPointsIter = rasterTargetGenerator(vObserver)
        for rTarget in rTargetPointsIter:
            visible = visibility(rObserver, rTarget)
            weight, wasPolygon = findWeight(rTarget)
            df = float(vapInput.distFactor)
            dist = float(resolution *
                      math.sqrt(QgsPoint(*rObserver).sqrDist(*rTarget)))
            if (not vapInput.limitPoly) or (wasPolygon):
                if visible:
                    if df <> 0.0:
                        weight /= 2**(dist/df)
                    rasterIO.addValue(rTarget, weight)
                    rasterIO.addPathValue(rObserver, weight)
                    self.totalPathValue += weight
                else:
                    rasterIO.addValue(rTarget, 0)

averagePathValue = totalPathValue / float(totalObservers)
```

4.5 Algorithmic Complexity

The name for the R3 algorithm that was defined by Franklin and Ray (1994) comes from the time requirements of the algorithm based on the radius of visibility limit of the observer. In this sense, R3 has a cubic time complexity of $O(n^3)$ when the input size is controlled by the radius of visibility. If the radius is taken as r in pixel counts, the target area (full view field) would have $pi \cdot r^2$ cells.

For every target cell, the intermediate cells from the observer to the target should be analyzed for obstruction. Whenever an obstructing cell is found, the target

is marked as not visible. Although the precise cut point would be dependent on the specific characteristics of the terrain, it can be assumed that half of the intermediate cells would be checked on average for any target. The distance to the target obviously would be different for every target. However, in any case, it can be said that the intermediate cell checks will increase with the visibility radius of the observer.

In total, the time complexity for a single observer visibility computation with R3 becomes $O(n^3)$ for a radius based input definition. This is valid only for a single binary viewshed computation with R3 algorithm. For the multi-visibility situation, the only change would be the number of observers. Since the number of observers is an independent variable from the number of targets, the algorithmic complexity can only be defined if the observer locations can also be defined in terms of the input size (radius on this case). For example, a path length that is equal to the visibility limit radius would result in a complexity value of $O(n^4)$.

CHAPTER 5

APPLICATION OF THE METHODOLOGY

5.1 Sample Region and Paths

Kovada Lake in Turkey has been selected as the test case for this study (Figure 5.1). Kovada Lake is located 30 km south of Egirdir, a district of Isparta province. The lake with the surrounding area of 6,534 hectares has been one of the national parks of Turkey since 1970. The circumference of the lake is 20.6 km and there are walking courses around the lake for hiking.

Figure 5.1: A satellite view of the Kovada Lake.

For the analysis, the DEM data of a sample region of 14.65×9.8 km around the Kovada Lake has been used used. The map data is acquired from the General Command of Mapping of Turkey (`http://hgk.mil.tr`) as GeoTiff files. The map consists of 586×392 (229,712) pixels with a spatial resolution of 25 meters in both vertical and horizontal dimensions. DEM data in two different color shadings can be seen in Figure 5.2. The minimum elevation for the area is 420 meters and the maximum elevation is 1,670 m with a mean of 1,160 meters. The topographical characteristics of the region may be better understood through the use of a relief-shaded map of the same area as given in Figure 5.3

The walking courses around the lake are good candidates for the sample paths. Three such paths have been constructed based on the actual walking courses (Figures 5.4 and 5.5). Although those three paths start from different areas, the paths 1 and 2 meet somewhere along the road, and all the three paths finish on the same general location where the walking courses meet. Total (2D) line lengths for the path 1 (4,651 m.) and the path 2 (4,495 m.) are close to each other, while the path 3 is covering a larger distance (7603 m.).

5.2 Weighted Zones

For the purposes of this study, a number of imaginary target zones have been defined over the sample area as seen in Figure 5.6. These zones represent the interesting targets that have been defined for the sample case. The target locations outside of the zones would have a weight that is equal to 1 by default, while a value of 0 indicates non-visibility. First of all, we have our main interest: Kovada lake. Obviously, a view of the lake would greatly affect the visibility value of a vantage point in a positive way. Another important target may be one the hills on the west side of the area which also has a positive impact on the visibility value. We can also define some historical ruins in the area that have some value for the observers. Moreover, a peculiarly beautiful forest scenery and fields at the northern side of the lake can have their own weights.

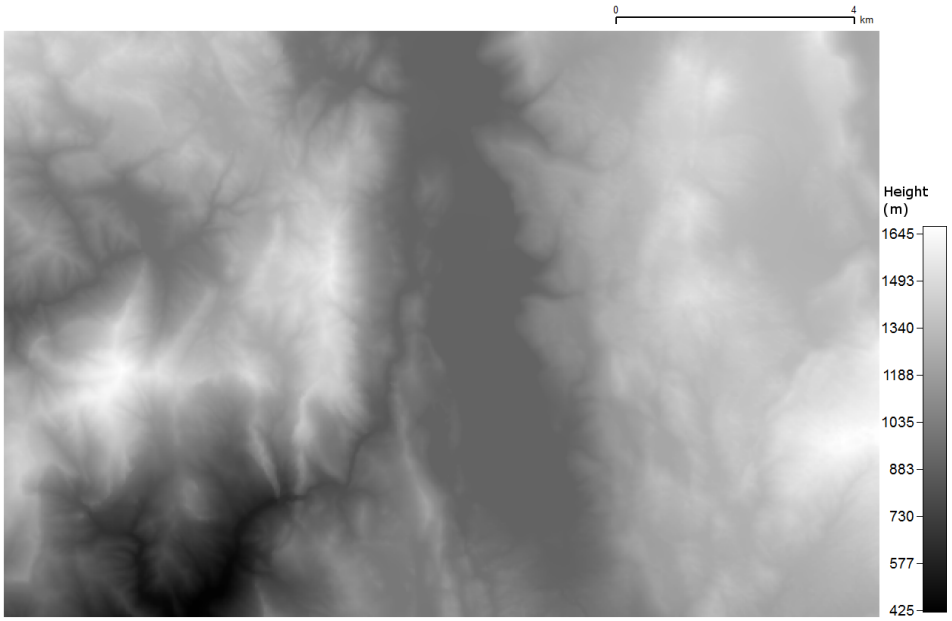


Figure 5.2: DEM data for the sample region in grey-shade.

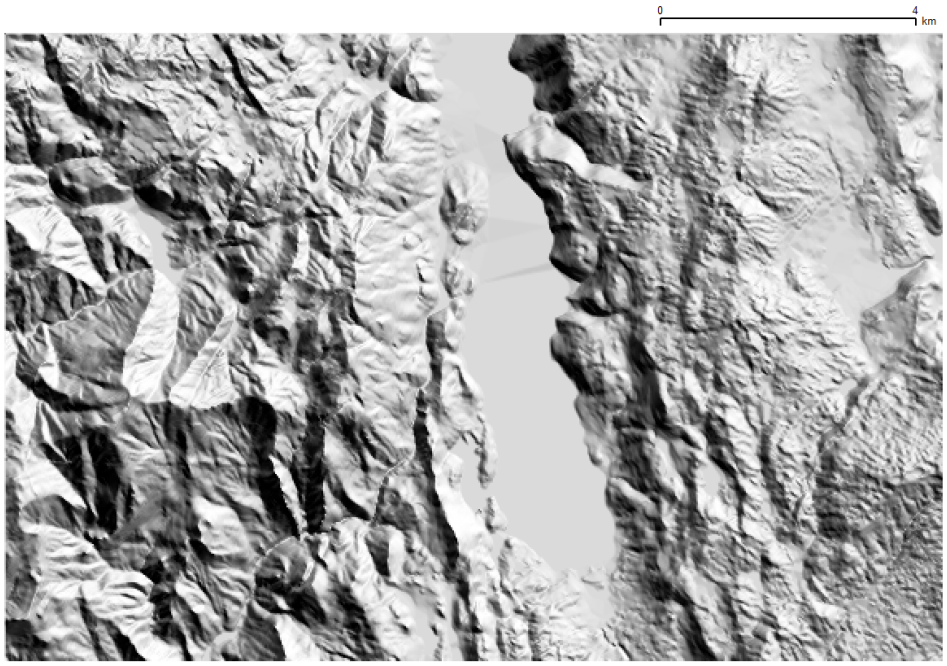


Figure 5.3: Relief-shaded version of the sample region.

Figure 5.4: Three sample paths based on the walking courses as shown over the satellite view of the lake.

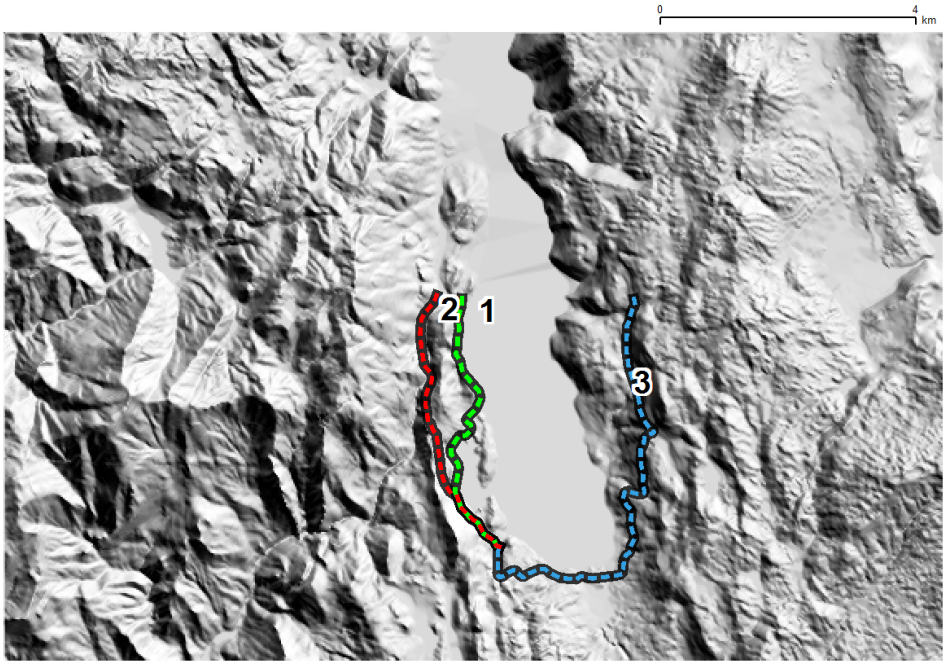


Figure 5.5: The sample paths around the Kovada lake overlayed on top of the relief-shaded map of the area.

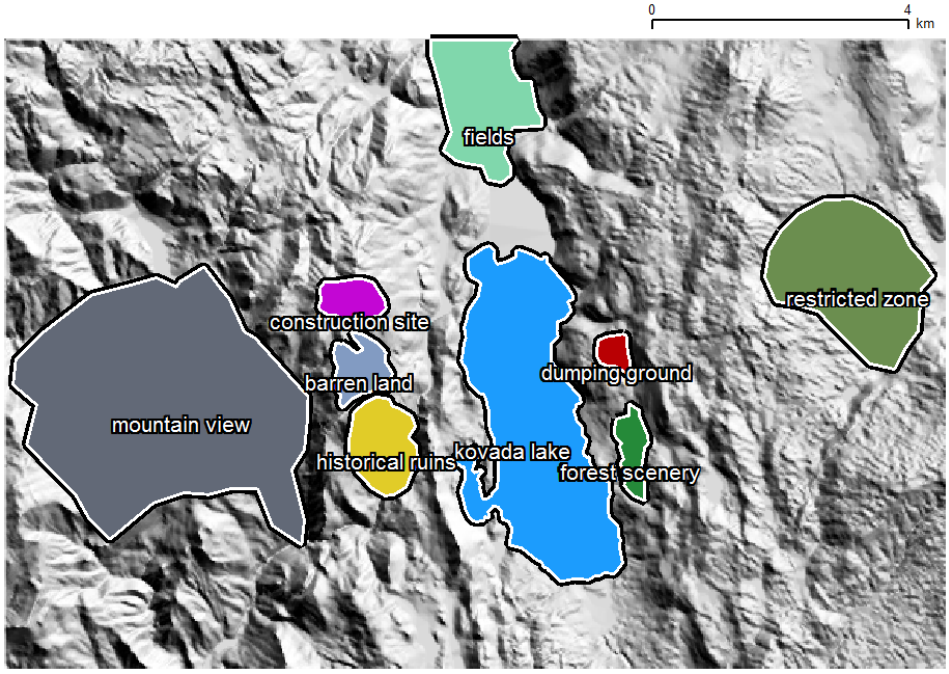


Figure 5.6: Sample zones that were defined for the study.

Targets with weights smaller than the default value of 1 will have a negative effect on the visibility value of the observer. There are four such zones in the example: a barren land, a construction site, a dumping ground, and a restricted zone. A weight of zero would equate the visibility of the target with its non-visibility state. A negatively weighted target would decrease the visibility value whenever it is visible. The assigned weights for the sample zones are shown in Figure 5.7.

5.3 Calculation

The software developed for the methodology is used with the paths and target zones defined in the previous section. For the observer height, 1.6 m is used representing a walking person. The average fixed viewing angle for the observer in this study is defined as 140° considering the comfortable field of view for a person. Visibility calculation is limited to a radius of 4,000 m from the observing point for the purposes of this study.

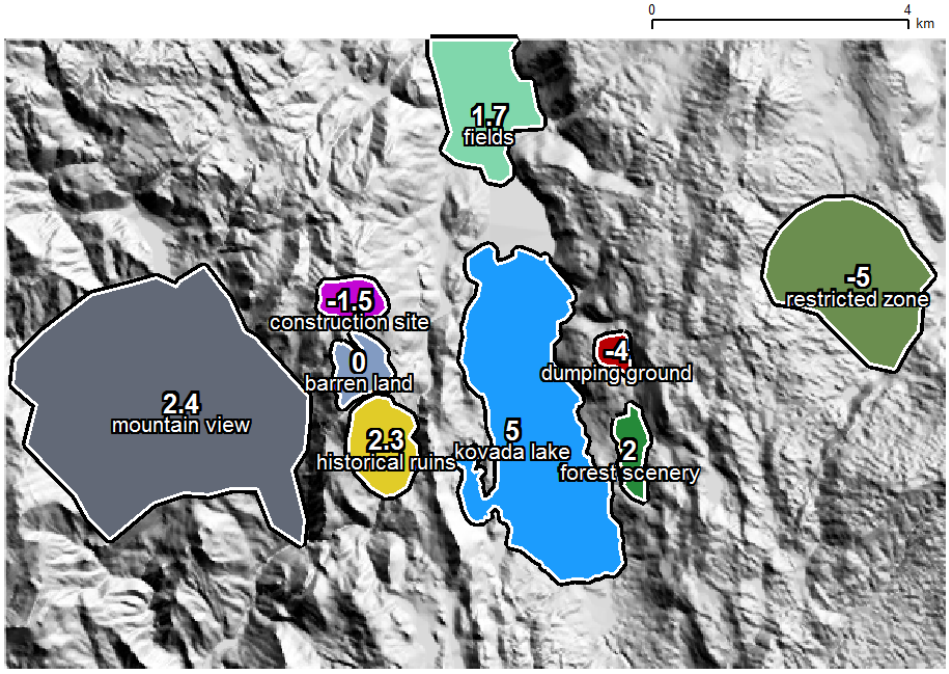


Figure 5.7: Weight values for the sample zones.

All the target points that fall inside the cross section of the field of view and circle of distance limit are tested by the software for the visibility. Also, a distance factor of 10,000 m is defined. This distance factor applies a degradation on the weight of the target points such that the weights are halved every ten kilometers. These additional parameters are summarized in Table 5.1.

The three sample paths are analyzed with the given parameters in both directions. The original direction which runs from the northern side of the map to the meeting point on the southern side is labeled as the direction 'A' while the opposite direction becomes 'B'. In total, six different analysis results are obtained for the test case.

5.4 Results

The visibility surfaces that have been constructed for the three sample paths in the previous section are presented in Figures 5.8, 5.9, 5.10, 5.11, 5.12, and 5.13. The dominant color in these color-shaded maps usually indicates the zero valued

Table 5.1: Software parameters for the test case.

Observer Height	1.6 m.
Directional View Angle	140°
Distance Limit Radius	4,000 m.
Limited to Polygons	No
Distance Factor	10,000 m.

locations (*i.e.,* non-visible in most cases) and can be seen to define the borders of the visibility surface.

It can be observed that some of the target zones did not have any effect on the result. This happens when no points inside that zone are visible from any of the source points. On the other hand, the effects of direction can clearly be seen. Higher visibility values over the lake skews towards the end points of the paths. Since the lake has a relatively higher weight, target locations inside that area usually dominate the visibility surface. Also, it can be observed that the surface closely reflects the reduction effect of the negatively weighted zones.

A 3D view of the terrain can provide an alternative view regarding the relation of the visibility surface to the actual terrain. The visibility surfaces are overlayed on the shaded 3D views of the actual terrain in Figures 5.14, 5.15, 5.16, 5.17, 5.18, and 5.19. This overlay map is constructed by the use of several modules from SAGA GIS software (`http://www.saga-gis.org`).

It can be seen that, there are many segments on the paths that the visibility range of the observer is severely diminishing. Another observation is that the view of the lake is relatively uniform only for the paths 1-A and 3-B. The other path options emphasize a smaller part of the lake in their course.

The unwanted sights of construction site and the dumping ground are seem to be mostly present in the paths 1-B and 2-B, while the paths 3-A and 3-B are affected only by the dumping ground. Other negatively weighted zones are not in any considerable line of sights. Among the positively weighted zones, the forest scenery is present mostly for the paths 1-A, 2-A, and 3-B.

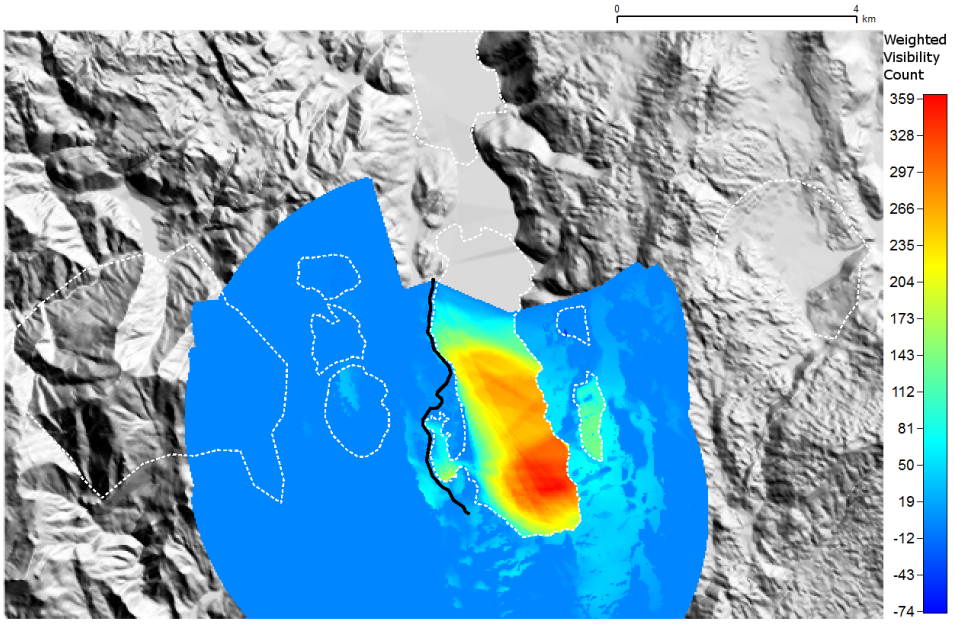


Figure 5.8: Visibility surface for the path 1-A.

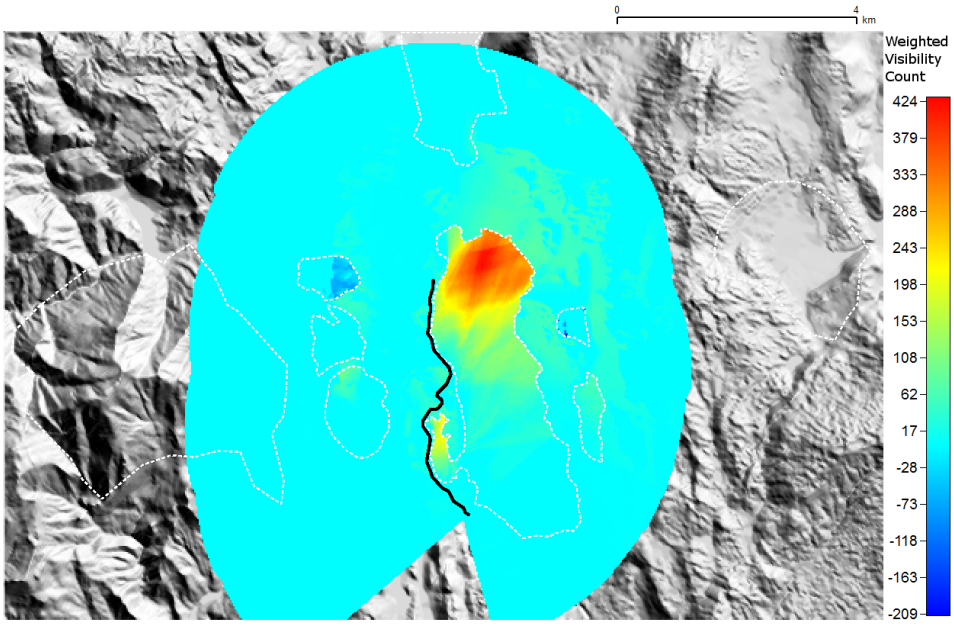


Figure 5.9: Visibility surface for the path 1-B.

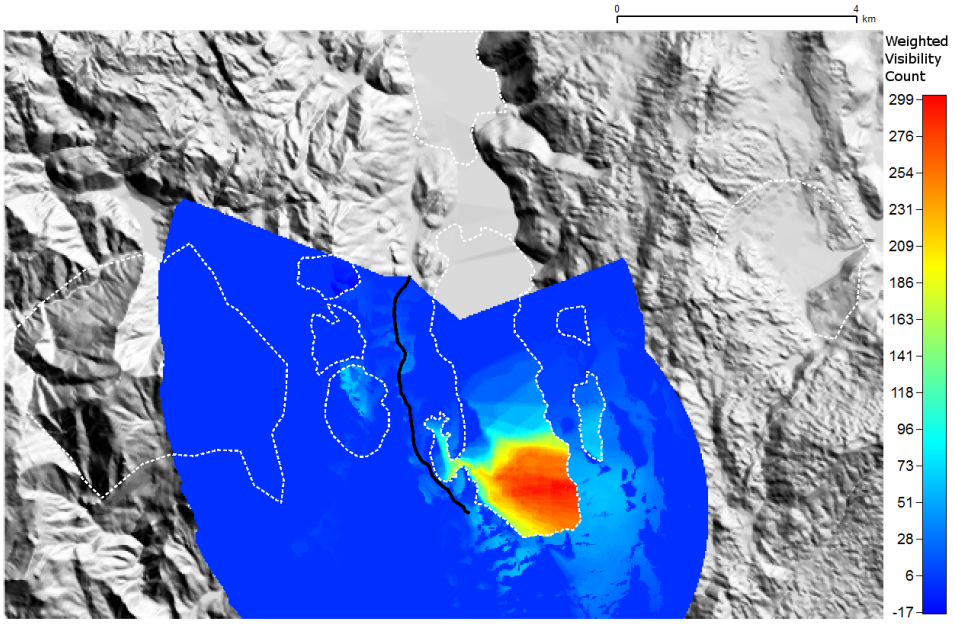


Figure 5.10: Visibility surface for the path 2-A.

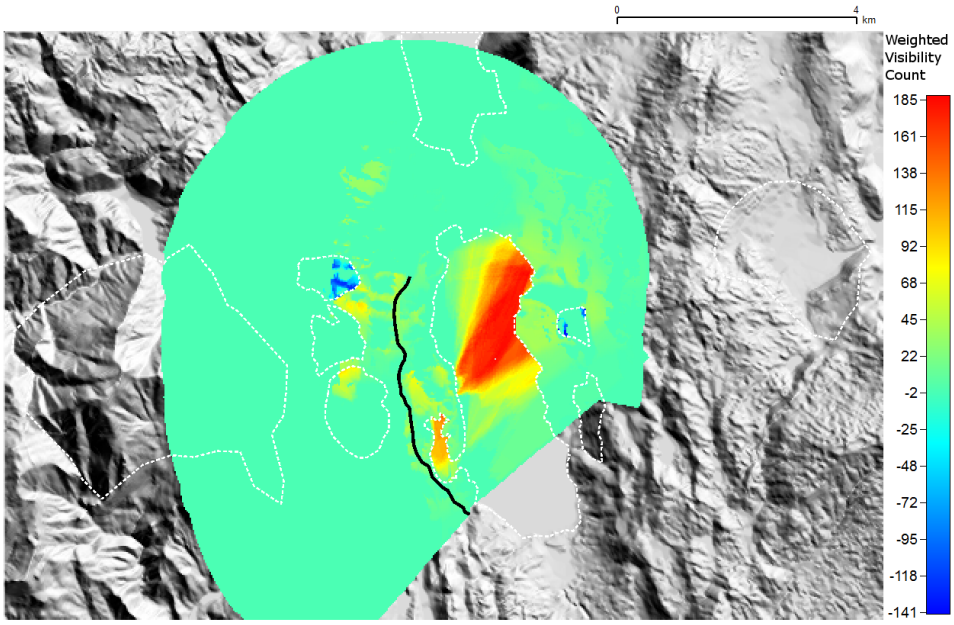


Figure 5.11: Visibility surface for the path 2-B.

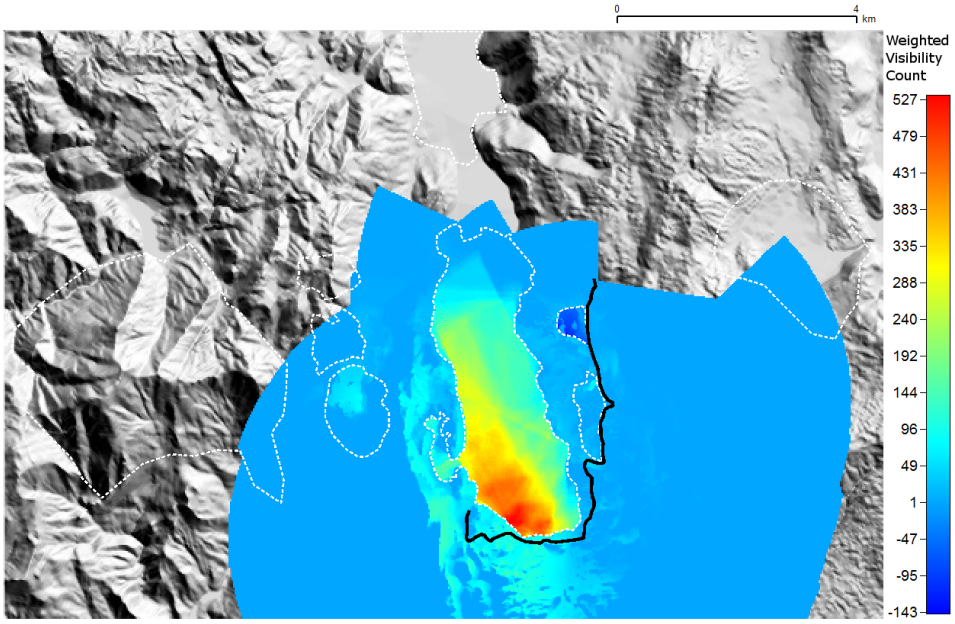


Figure 5.12: Visibility surface for the path 3-A.

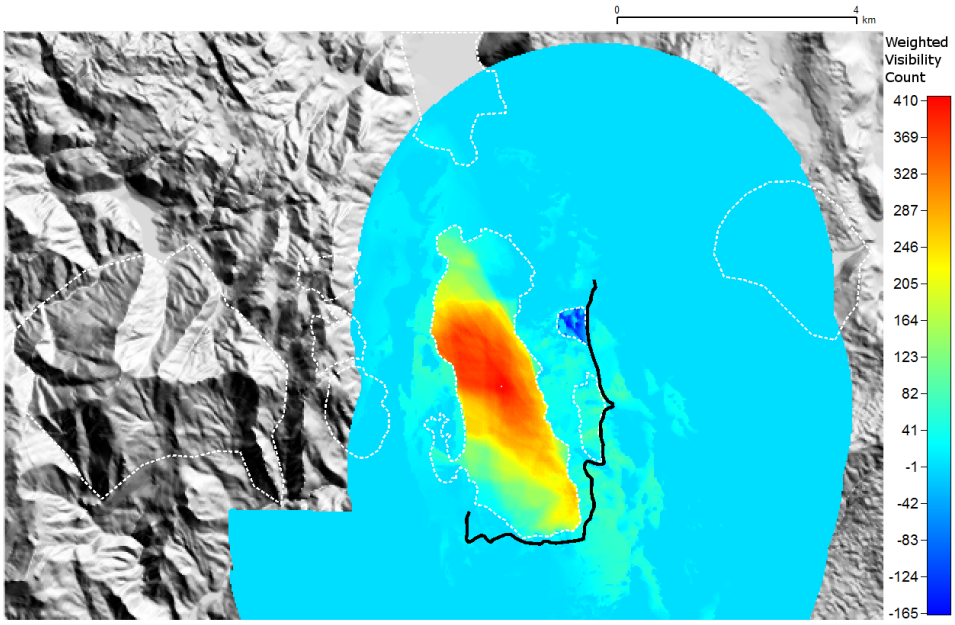


Figure 5.13: Visibility surface for the path 3-B.

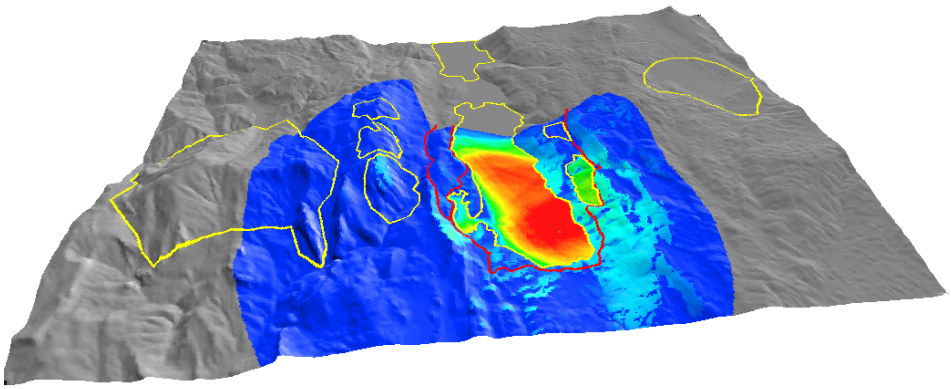


Figure 5.14: Visibility surface for the path 1-A overlayed over the shaded 3D view of the actual terrain.

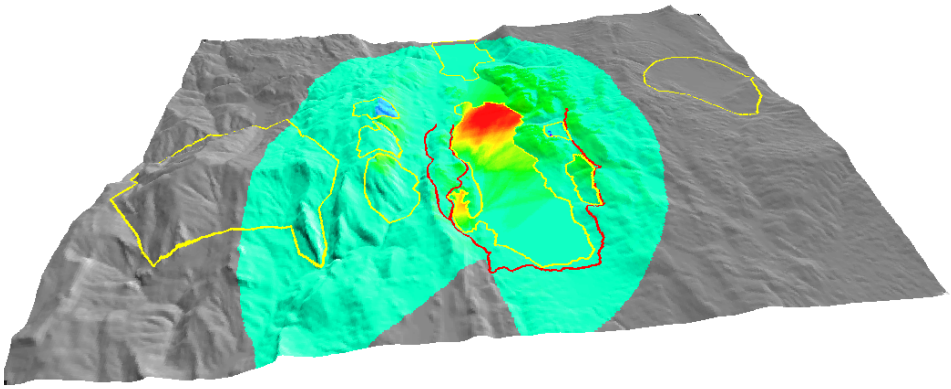


Figure 5.15: Visibility surface for the path 1-B overlayed over the shaded 3D view of the actual terrain.

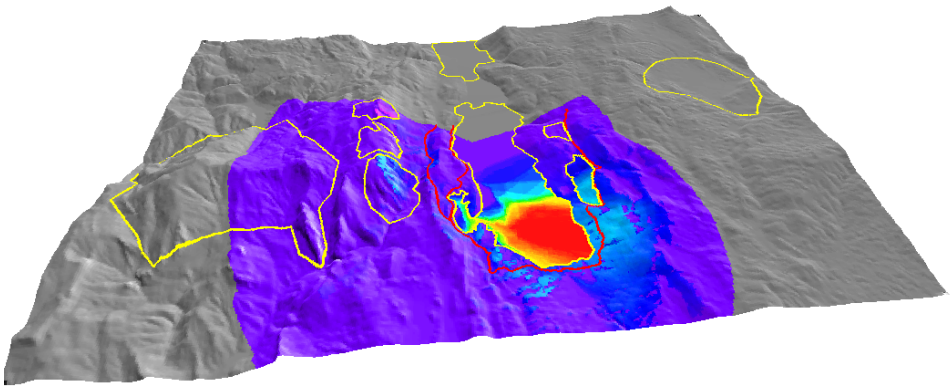


Figure 5.16: Visibility surface for the path 2-A overlayed over the shaded 3D view of the actual terrain.

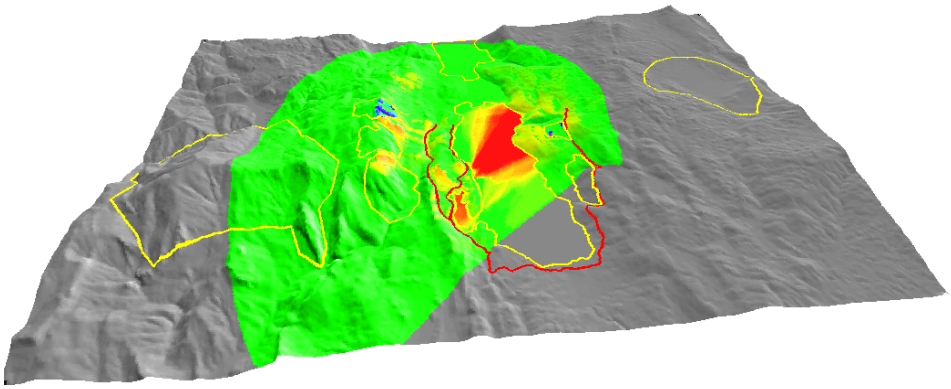


Figure 5.17: Visibility surface for the path 2-B overlayed over the shaded 3D view of the actual terrain.

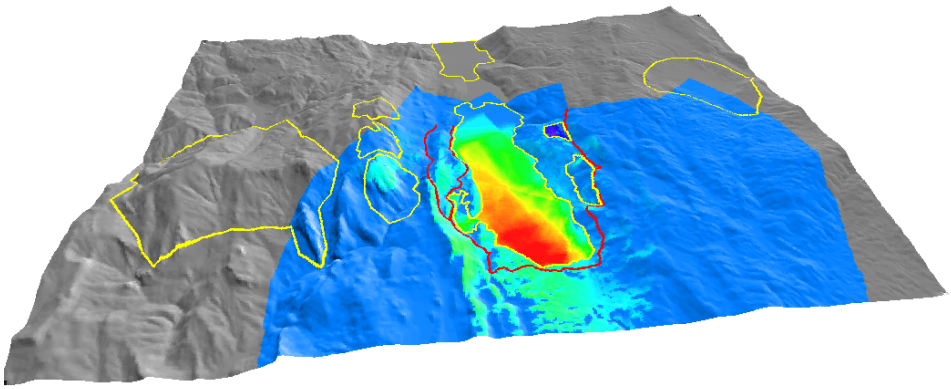


Figure 5.18: Visibility surface for the path 3-A overlayed over the shaded 3D view of the actual terrain.

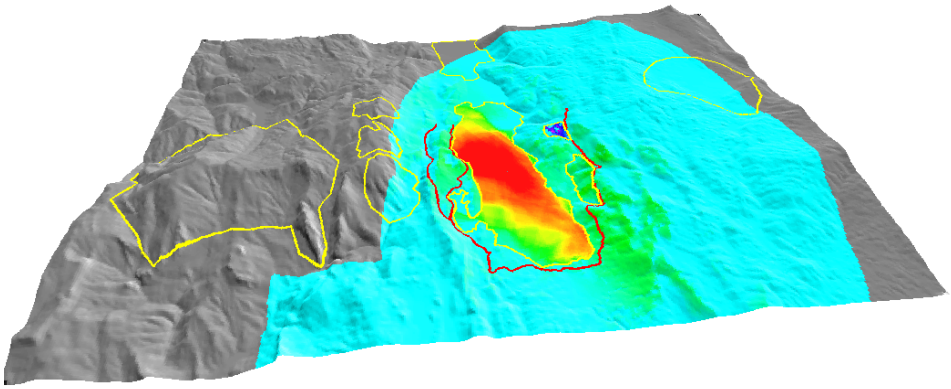


Figure 5.19: Visibility surface for the path 3-B overlayed over the shaded 3D view of the actual terrain.

Apart from the actual terrain, it is also informative to see the constructed visibility surfaces in a 3D view. 3D views of the six surfaces can be seen in Figures 5.20, 5.21, 5.22, 5.23, 5.24, and 5.25. In these views, the complex patterns of the visibility surface and the actual impact of an area on the path becomes apparent.

The raster paths which have accumulated weighted inverse visibility counts are visualized in two ways in the test case. First way is to display a color shaded raster path map on top of the actual terrain. These maps are shown for the test case in Figures 5.26, 5.27, 5.28, 5.29, 5.30, and 5.31. To increase the contrast, a black buffer is added for the paths. Obviously, the source path visibility values are much higher in average than the values for the targets.

Second way is to display a 2D diagram of the visibility value profile of the paths as in Figures 5.32, 5.33, 5.34, 5.35, 5.36, and 5.37. In these profile diagrams, the horizontal axis is used to display the pixel counts along the (rasterized) path directions. The vertical axis shows the weighted inverse visibility counts for the corresponding pixels. This diagram can be used to assess the visibility values for the different points along the path. It can be seen that the profiles for the paths are not smooth curves and usually have very sharp dips and peaks. This indicates a relatively quick changing (in terms of 'value') visual landscape for the observer following those paths.

Quantitative results for the test case are given in Tables 5.2, 5.3, and 5.4. The tables display the opposite directions of the same path side by side. Number of cells, minimums, maximums, mean values, and standard deviations are calculated for the target pixels (visibility surface) and the observer pixels (observer path). For the paths, the total and average path values are also given. Finally, the total number of visibility calculations for a direction and its total and average calculation times are given.

The average calculation time for a two-point visibility was around 700 and 800 microseconds for this test case on a notebook computer with a 2.66 GHz CPU and 4 GB of RAM. The total operation times were around one hour for the first two paths, and around one and a half hour for the third path. Total number of visibility

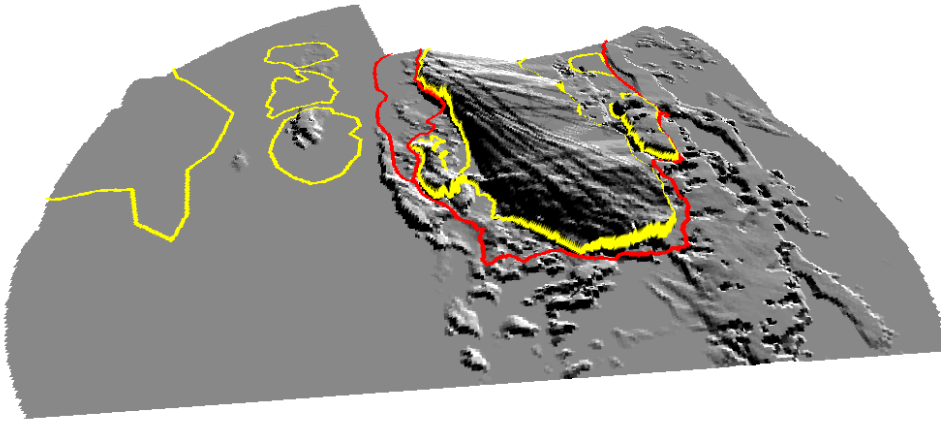


Figure 5.20: Visibility surface for the path 1-A in shaded 3D view.

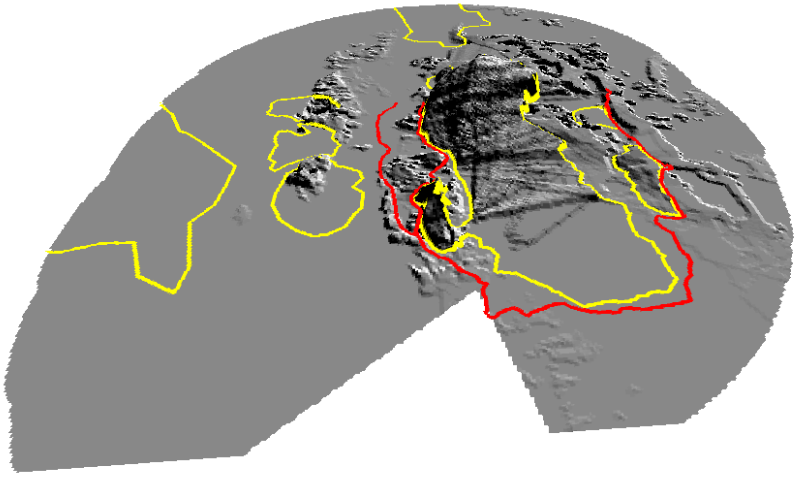


Figure 5.21: Visibility surface for the path 1-B in shaded 3D view.

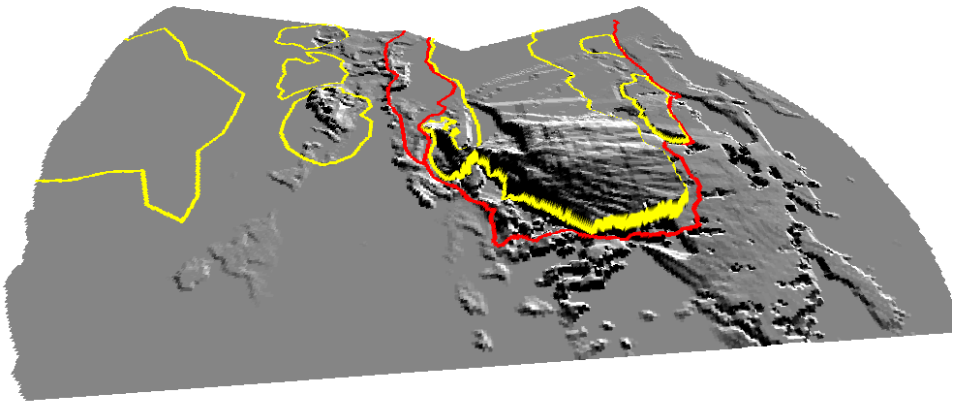


Figure 5.22: Visibility surface for the path 2-A in shaded 3D view.

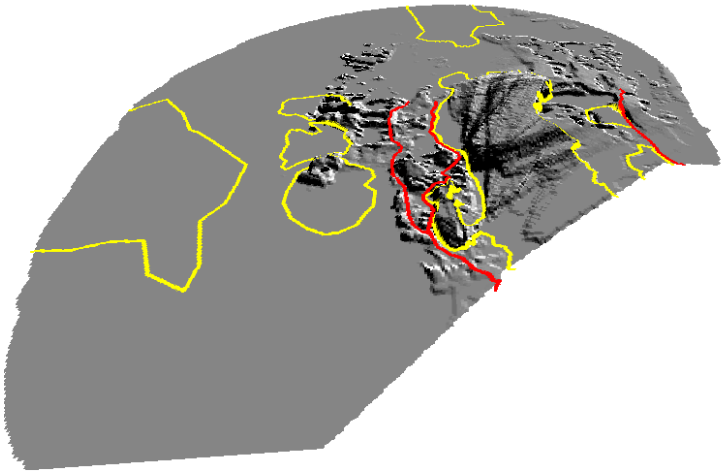


Figure 5.23: Visibility surface for the path 2-B in shaded 3D view.

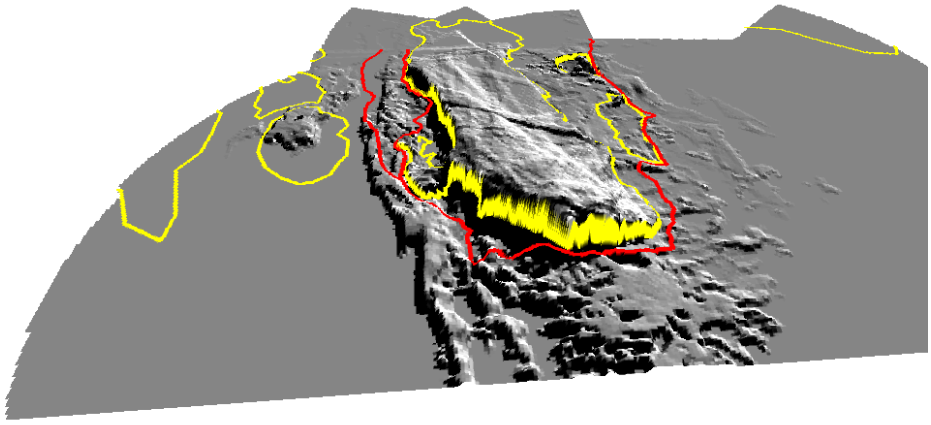


Figure 5.24: Visibility surface for the path 3-A in shaded 3D view.

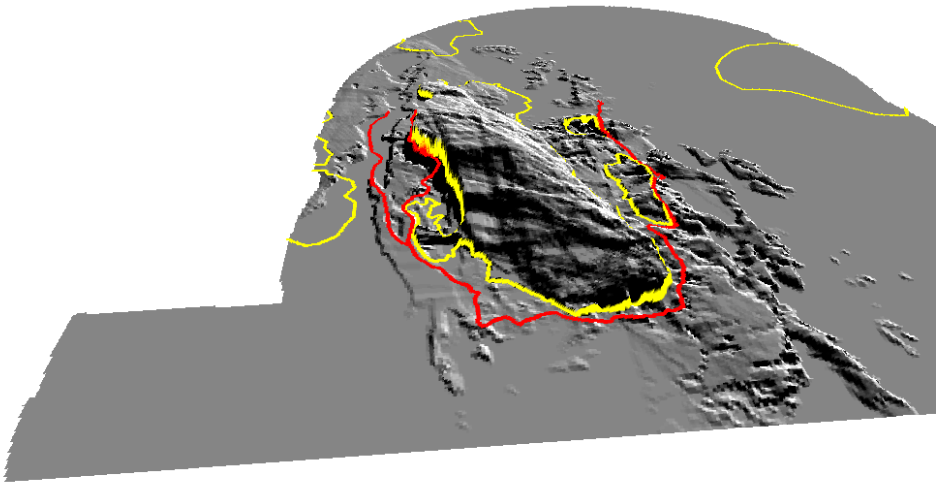


Figure 5.25: Visibility surface for the path 3-B in shaded 3D view.

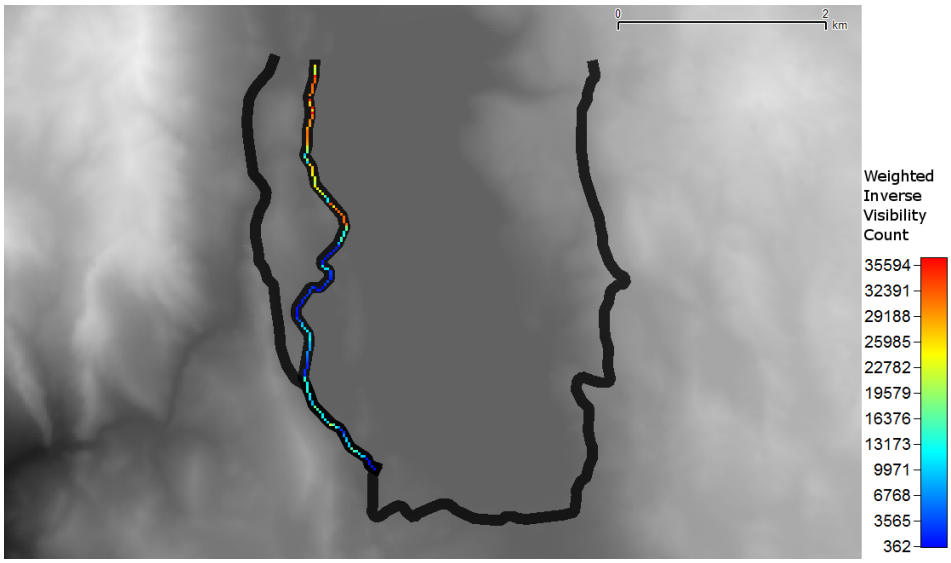


Figure 5.26: Weighted inverse visibility count for the path 1-A.

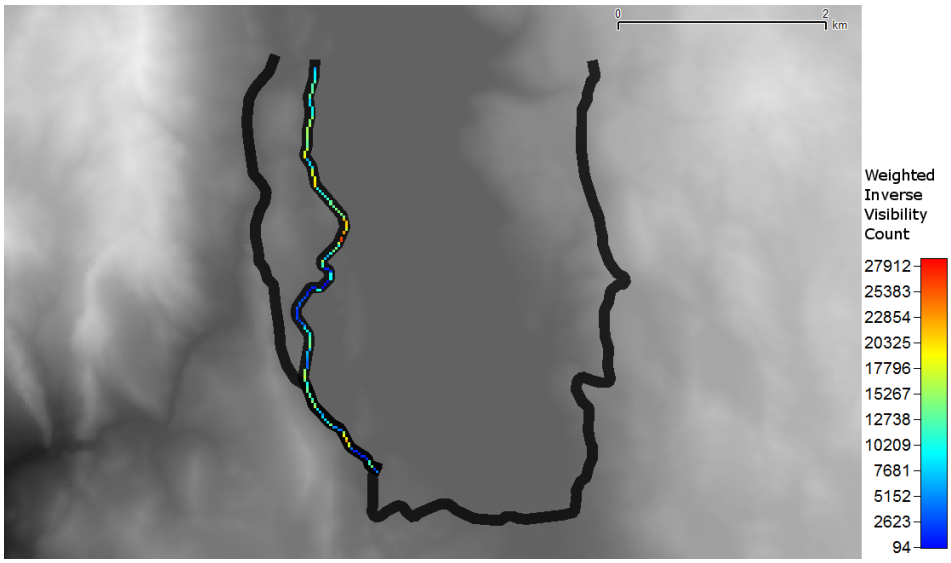


Figure 5.27: Weighted inverse visibility count for the path 1-B.

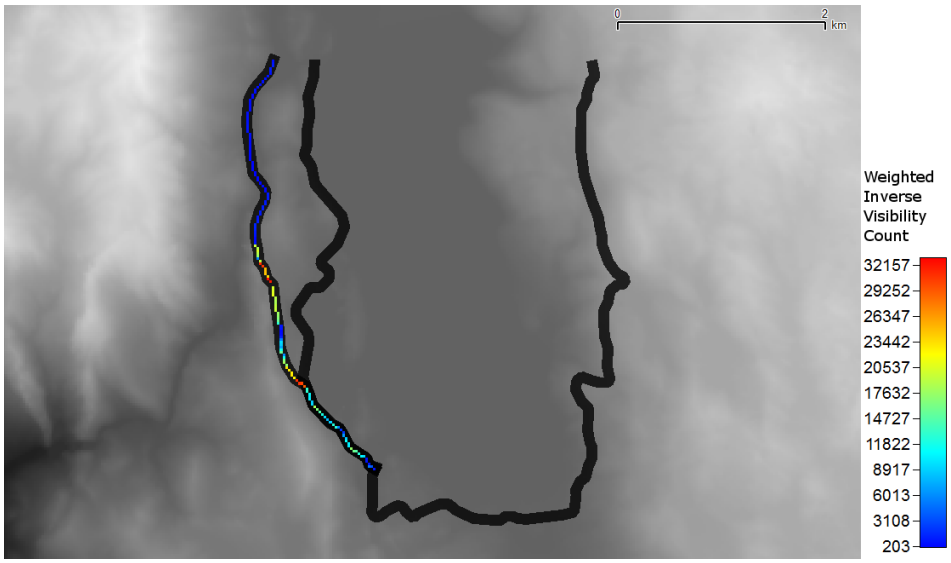


Figure 5.28: Weighted inverse visibility count for the path 2-A.

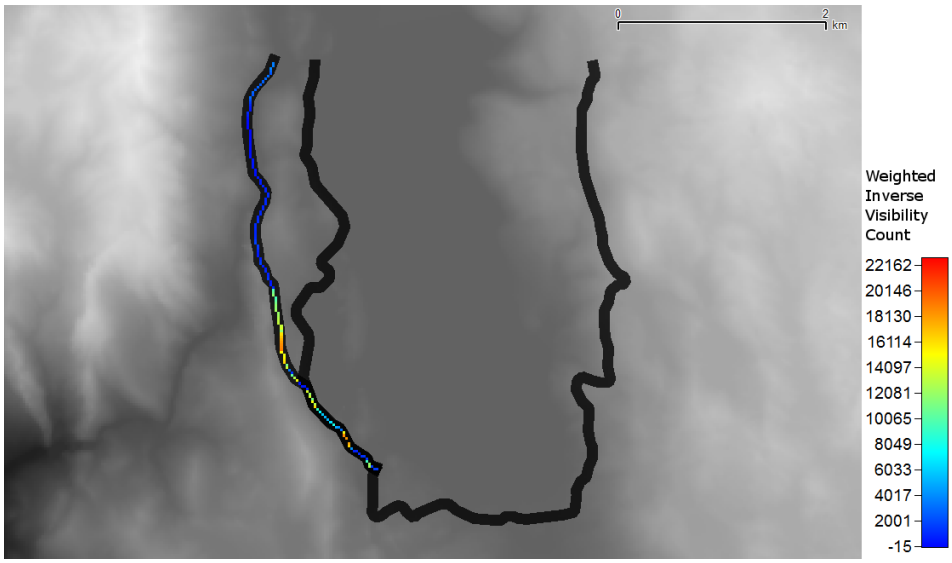


Figure 5.29: Weighted inverse visibility count for the path 2-B.

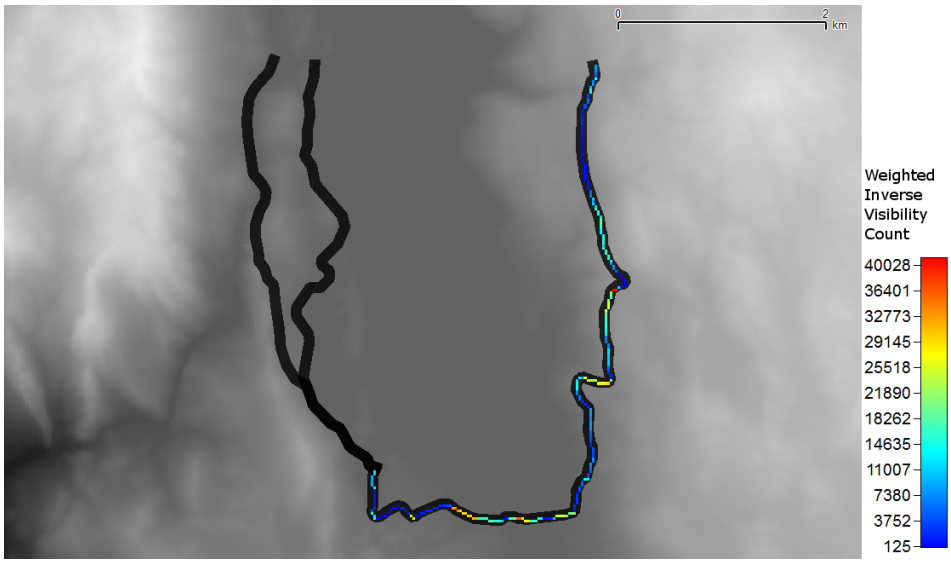


Figure 5.30: Weighted inverse visibility count for the path 3-A.

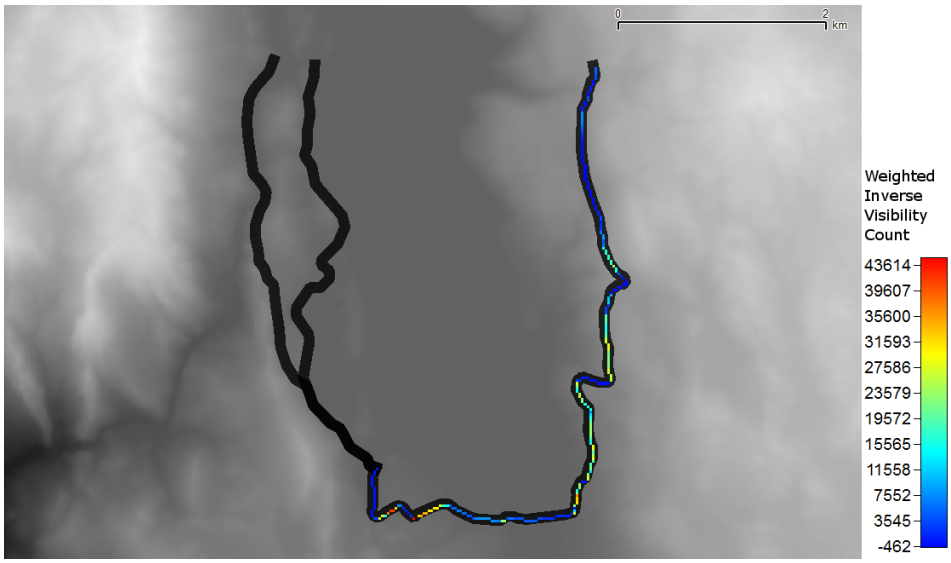


Figure 5.31: Weighted inverse visibility count for the path 1-A.

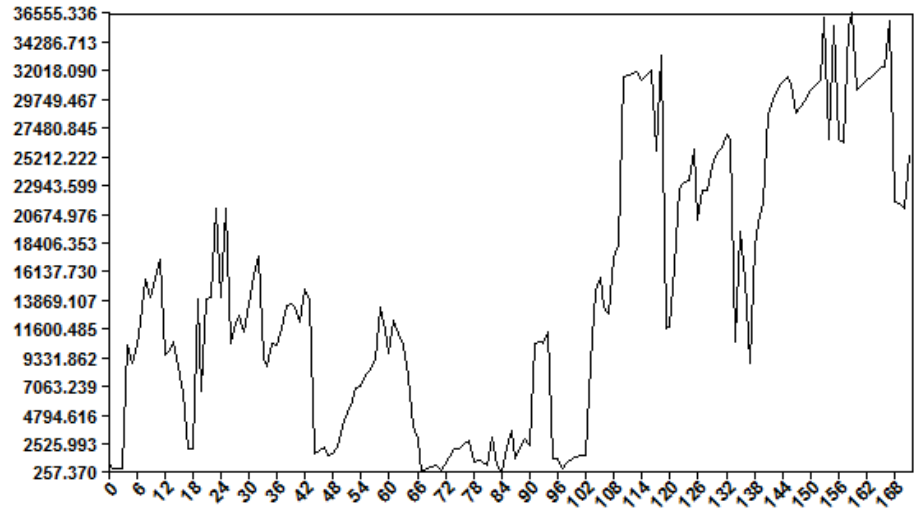


Figure 5.32: Profile diagram for the path 1-A. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

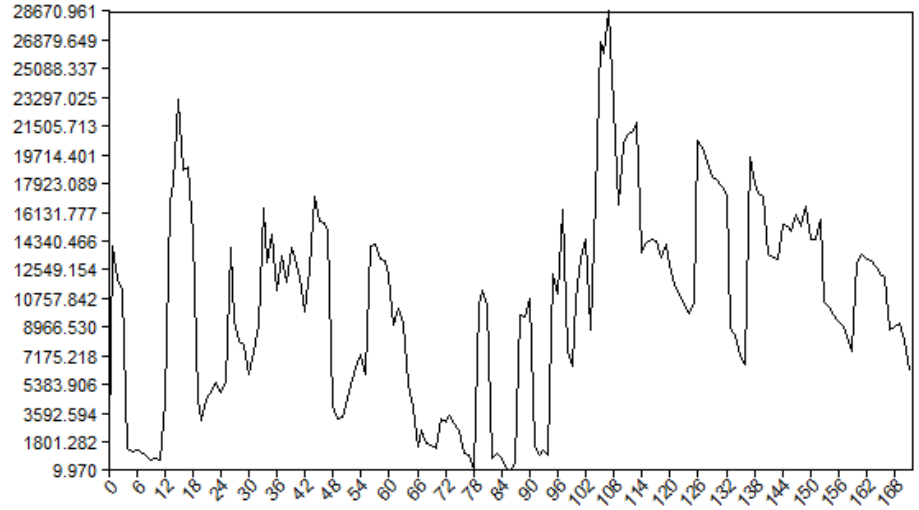


Figure 5.33: Profile diagram for the path 1-B. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

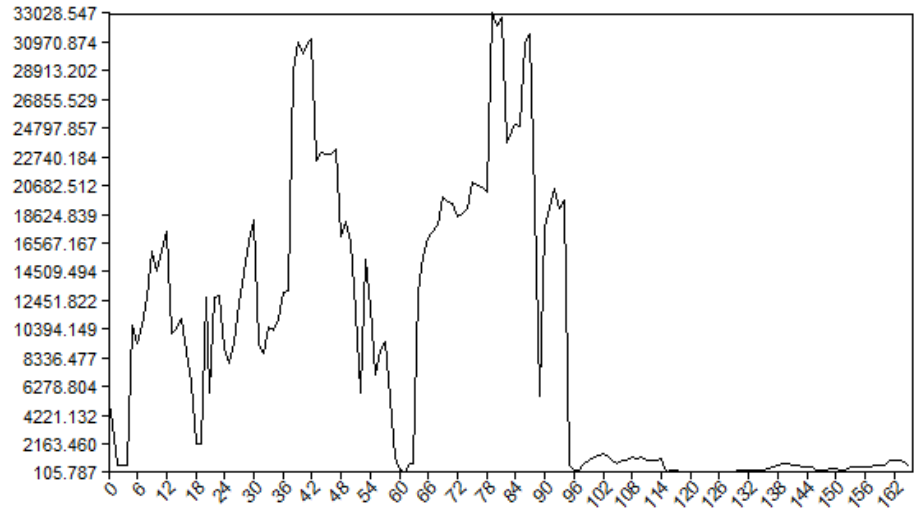


Figure 5.34: Profile diagram for the path 2-A. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

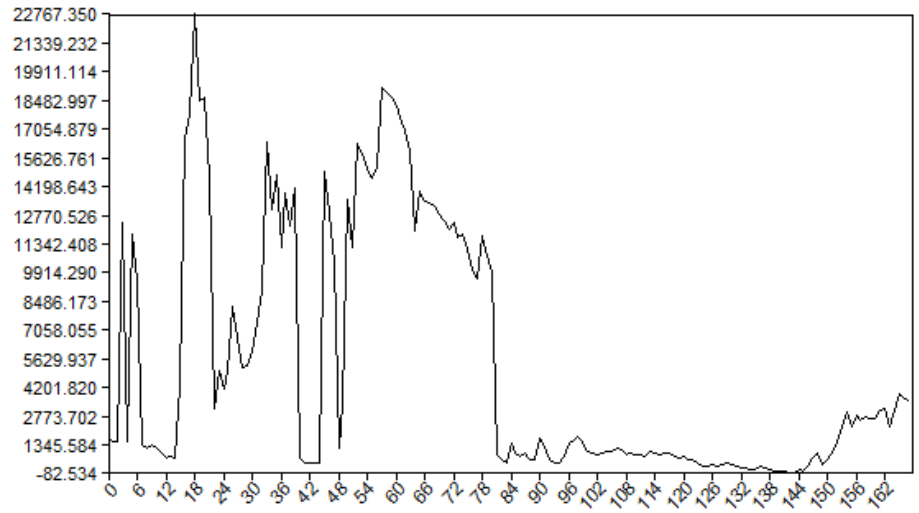


Figure 5.35: Profile diagram for the path 2-B. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

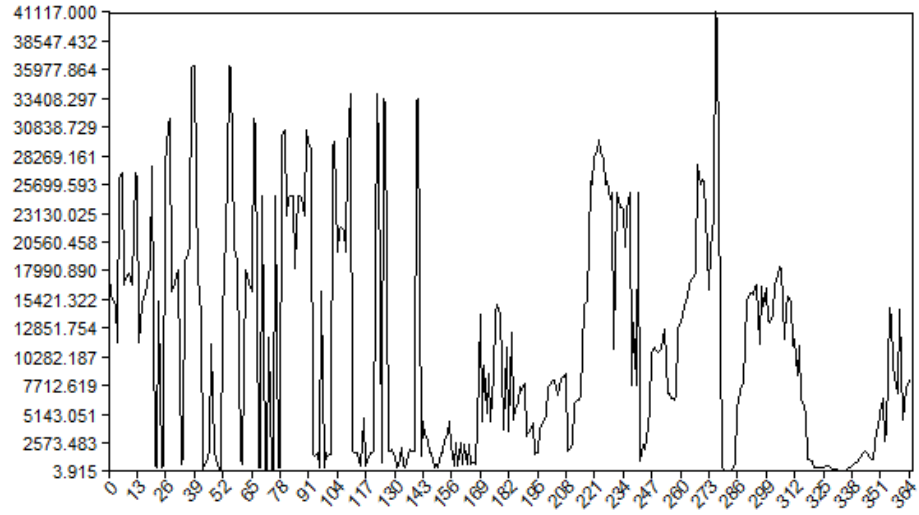


Figure 5.36: Profile diagram for the path 3-A. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

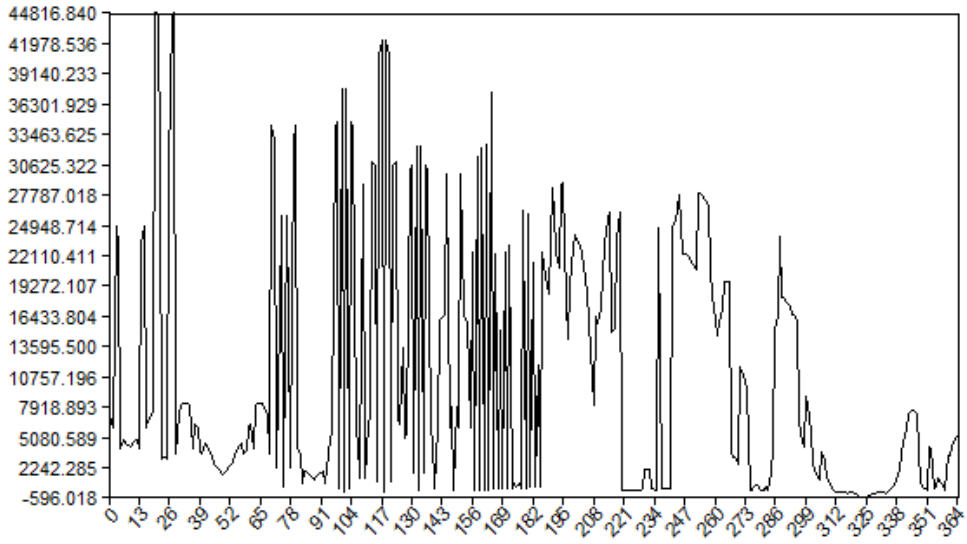


Figure 5.37: Profile diagram for the path 3-B. Horizontal axis represents the pixels along the path starting from the meeting point. Vertical axis shows the weighted cumulative value for the corresponding pixel.

calculations were around 4.5 to 8 million. Python language is generally considered to be slower in its computation speed than some other popular languages like the C. However, due to the rasterization based approach to the visibility algorithm that is used here, algorithm times remain within the usable and practical ranges.

If we ignore the edge effects, average path values provide a good quantitative comparison option between the alternative paths. Path 1-A has an average value of 14,540.59 while its opposite direction (1-B) value is 10,671.60. It can be said that, path 1 is more preferable when traversed in the direction 'A'. Similarly, path 2-A is more preferable to path 2-B (8,611.91 to 5,405.83). However, for path 3, the preferred direction would be the 'B' direction (10,150.97 to 10,467.54). If different paths are compared with each other in terms of their average values, path 1-A turns out to be the best path for the purposes of the defined test case. If total visibility values are more important than the averages, this time path 3 would be the preferred one.

Table 5.2: Analysis results for the first path.

	Direction 'A'		**Direction 'B'**	
	Visibility Surface	**Observer Path**	**Visibility Surface**	**Observer Path**
Number of Cells	78,012	165	105,849	165
Minimum Value	-75.52	255.26	-210.48	9.97
Maximum Value	363.10	36,555.34	429.42	28,670.96
Mean Value	30.75	14,386.74	16.64	10,634.01
Standard Deviation	72.43	11,218.91	52.56	6,232.86
Total Path Value	-	2,399,198.02	-	1,760,814.14
Average Path Value	-	14,540.59	-	10,671.60
Visibility Calculations	4,514,688		5,141,182	
Total Analysis Time	58 m 09.72 s		66 m 47.18 s	
Average Time	772.97 μs		779.43 μs	

Table 5.3: Analysis results for the second path.

	Direction 'A'		Direction 'B'	
	Visibility Surface	**Observer Path**	**Visibility Surface**	**Observer Path**
Number of Cells	78,755	163	94,329	163
Minimum Value	-17.96	105.79	-142.43	-82.53
Maximum Value	301.65	33,028.55	187.64	22,767.35
Mean Value	17.82	8,609.07	9.34	5,383.94
Standard Deviation	51.10	9,624.43	29.97	6,221.80
Total Path Value	-	1,403,741.49	-	881,150.35
Average Path Value	-	8,611.91	-	5,405.83
Visibility Calculations	4,466,160		5,075,769	
Total Analysis Time	55 m 03.93 s		59 m 56.32 s	
Average Time	739.77 μs		708.53 μs	

Table 5.4: Analysis results for the third path.

	Direction 'A'		Direction 'B'	
	Visibility Surface	**Observer Path**	**Visibility Surface**	**Observer Path**
Number of Cells	93,818	285	118,501	285
Minimum Value	-144.56	3.92	-166.27	-596.02
Maximum Value	532.49	41,117.00	414.75	44,816.84
Mean Value	30.84	10,122.93	25.17	10,448.87
Standard Deviation	78.68	9,471.61	71.66	11,270.91
Total Path Value	-	2,893,026.54	-	2,983,248.06
Average Path Value	-	10,150.97	-	10,467.54
Visibility Calculations	7,242,022		7,891,451	
Total Analysis Time	88 m 15.07 s		105 m 21.18 s	
Average Time	731.16 μs		801.02 μs	

CHAPTER 6

DISCUSSION

6.1 Discussion of the Approaches

The methodology developed in this study can be discussed in terms of the three approaches it combines. The multi-visibility properties of the paths, the definition of preference through the use of weighting systems, and the directional nature of moving observers have all been instrumental in the usability of this methodology.

6.1.1 Multi-Visibility for Paths

Although the visibility in its standard form is usually understood as the binary viewshed generation for a single point-type source, most useful and rich results are provided by multi-visibility analyses. One dimensional linear paths can be encountered for many types of multi-visibility cases. For example, a path can be used to represent a river and multi-visibility can be applied to reveal the percent of river that is visible from an area (an inverse visibility). These types of analyses treat the path as a collection of points that form a whole. As opposed to that, in this study, the path was treated as a sequential collection of positions (and directions) for an observer which moves in time.

The combination for multiple visibility results were in the form of aggregations over pixels. As discussed in the previous chapters, there are many type of set operations that can be applied, but only an aggregation can provide non-binary result maps. The aggregation has been used for both of the source and target pixels.

The cumulative results over the path reveal the overall quality of visibility from that particular observer location. Contrarily, the cumulative results over the target area are indications of the ratio of source path that has a visibility relationship with a particular target pixel. In that sense, these two different aggregations provide different but complementary information for the multi-visibility over a given case.

6.1.2 Weighted Targets

When the quality of a view or the visibility value of a location is considered, simple accumulative counts are not enough for many situations. For this reason, weighted target zones have been introduced in this study. The weighting system provides an opportunity to define the preference or value characteristics of the area. Obviously, the weights would only be applicable when the target is visible. Moreover, negative weights are used to define the zones that are not really preferred.

Two kinds of weighting are used in this study. One is the weights attached to specific target zones, and the other is the weighting effect of the defined distance factor. Although, these weights provide a firm base for the modeling of specific test cases, many different weighting criteria could also be defined. For example; the weights that are proportional to the height of the targets, the weights that take the slope and aspect into account, the weights that use the amount of sunlight, etc. Moreover, specific weights could also be defined for the source points.

Another factor would be whether the weights applied to the source or target aggregations, or for both. Applying the weights to the source path is usually desired, because the visibility value of a path would be affected by it. In addition, applying the weights to the target zones may also be desired, because it would provide clear visual information on the areas that are most affecting the source path. Therefore, these two approaches have been used at the same time in this study.

6.1.3 Directional Visibility

Since the study focuses on moving observers, the direction of movement should also be included in the methodology. In the basic sense, two directions are possible for a given path starting from one or the other end points. The direction of movement can be linked to the viewing direction for an observer in most cases as it is done in this study.

The direction is used to restrict the field of view for an observer in the methodology. In order to do that, a fixed viewing angle is defined for the observer which is assumed to be following the direction of the path. Although this approach can be a simplification for many of situations, it provides a basic and automated mechanism to include the direction in the analysis. It can be improved to include the speed and behavioral patterns of specific observers. Also, through directional visibility, opposite directions of the same path can be compared for the same test case.

6.2 Other Application Possibilities

Provided methodology can be used to assess a number of different situations. One obvious use case is the tourism or sight-seeing related applications as investigated in Chapter 5. In that case, visual preference of three paths have been compared in both directions to find out the best alternative for the walking courses.

The path can also be taken as a river, a railway, or a highway as all of these can be represented by paths. For example, the view from a boat following a river can be analyzed using this methodology. Another example would be when trying to find the best route for a new road construction near an ocean that would have the largest (or longest in time) view of the water body. Sometimes it may be desirable to avoid the view of certain areas or objects. The view of an industrial garbage area from different paths can be compared to find out the best alternative.

The visibility can also be used in reverse for certain tasks. When a point is visible from a certain location, the location would also be visible from that point (if the

observer height is also taken into account in the calculation). A reconnaissance type of operation in a military setting might need to minimize the visibility of people on a path from a set of watch towers. Finding the minimum visibility path in this setting would involve a least-cost path analysis on the visibility surface. However, if the alternative paths have already been defined, a comparison between them can be done using the given methodology. In this case, the tower areas should be marked with highly negative weight values.

Additionally, there may be a base station from where the observers should be visible in maximum amount of time for logistical support. This situation can also be modeled by defining positively weighted zones for the base stations. Another example for this case is the mountain climbers following a predefined path that are observed from the base camp by binoculars for safety. Naturally, it would be desirable to be able to follow the climbers from the base camp in maximum amount of time.

These are some of the use cases that the methodology defined in this study can be applied without much change. By the addition of other variables or enhancements, many more different scenarios can be analyzed or compared much more realistically. Some of these possible enhancements are discussed in Chapter 7.

6.3 Implementation Remarks

As described in Chapter 5, the software was implemented as a plugin to Quantum GIS host environment. This approach allowed the software to be used from any Quantum GIS installment as the plugins can be downloaded from Internet using the plugin manager of the host.

Python programming language provided fast development times, rich data types, and powerful manipulation mechanisms. The only disadvantage of it was the slower computation speed mostly due to its scripted nature. The speed was also restricted

by the use of R3 algorithm which has a cubic algorithmic complexity in time as described in Franklin and Ray (1994).

Although slower, R3 algorithm provides the most accurate results by using all of the data without any approximations. The use of other approximated algorithms can be considered to increase the speed at the expense of accuracy. The approximated R2 and XDraw algorithms are also defined by Franklin and Ray (1994) for this purpose which provides several order of magnitudes more speed. Izraelevitz (2003) tried to avoid some of the drawbacks of the XDraw algorithm by introducing a new method that is also fast but more accurate than the Xdraw.

Kaučič and Žalik (2002) compared several visibility algorithms including the R3, R2, and XDraw in terms of speed and accuracy. According to that study, for a 100 pixel view radius, the error ratio for R2 is 0.1% and for XDraw is 2.56% compared to the results for R3. However R2 and XDraw can provide a 30 to 90 times more speed in terms of calculation time.

In order to increase speed, some optimizations were applied in the implementation of the methodology. One such optimization is the cell-based approach used in the algorithm. All the computations are done on the grid locations provided by the resolution of the DEM, and not in between. To achieve this, the given vector path is rasterized using the Bresenham's algorithm and then the specific cell locations on this rasterized path are followed by the software. The target areas are also handled in a rasterized fashion further increasing the speed. Finally, the visibility computation is carried out also in a discrete way without using interpolation or other surface fitting methods on the in-between locations.

Another alternative for better efficiency in computation of visibility would be to use distributed parallel computing as discussed by Ware et al. (1998). Due to the independent nature of visibility algorithms such as R3, each target cell can be independently analyzed in parallel with the other cells. This property makes visibility analysis one of the most efficient candidates for distributed computing as there may be as many as computing resources as there are cells in the target area. However, it should be noted that many other more approximated fast visibility

algorithms are not really suitable for parallelization. These efficient algorithms usually use the result of the previous calculations in the successive analyses and must follow a specific order for the cells.

Although there is a small accuracy loss using discrete visibility calculations, it does not affect the results too much as observed in the comparison study discussed in Section 4.2. Furthermore, the effects of the errors in DEM data are much more influential in the results as studied by Nackaerts et al. (1999).

The DEM resolution is also important for the accuracy discussion. Obviously, more resolution (*i.e.,* smaller cell sizes) provides increased accuracy. However high resolution maps also need much more data space for storage and more time for computations. De Floriani and Magillo (2003) provide and overview of the effects of resolution in visibility analysis. The visibility results are usually affected by the locations near the observer point as small errors in those locations cause relatively high changes in the line-of-sight computations from the observer. One possible solution would be to use the high resolution data near the observers and gradually use lower resolutions while going further from the observer. However such an approach would be highly complex in its implementation.

6.4 Presentation of the Results

The main outputs of this study is a weighted visibility surface, a weighted path profile, and a total/average visibility value for the path. The software automatically constructs the visibility surface and path profile as separate raster maps in the current project. Also, a text file with the computed results and information is also written.

The raster maps can be visualized in many ways according to the specific goals of the study. It may be desirable to present the visibility surface in a color-shaded map over a 3D terrain. Another possibility is to construct contour maps with specific intervals to better understand the change of visibility values over the terrain. Furthermore, special queries can be applied to select out the specific portions of the

surface according to the values. The same arguments can be made for the path profiles as they are also given as raster maps.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

Visibility analysis has become one of the most used geospatial tools in recent years. As the quality of the digital elevation data and the available computational power increased, more and better uses for visibility have been found.

In this study, a combination of different approaches is applied to the standard visibility analysis in order to develop a visibility assessment system for the observers moving on directional paths. Through the use of weighted target zones, the visibility values which indicate the visual desirability of different paths are compared. Conversely, visibility effects of different target zones are also compared with respect to the specific source paths. The directional limitations of the field of view of observers are also been taken into account in an automated way in order to define a unified approach.

The resulting weighted visibility value surfaces for targets and weighted visibility value profiles for source paths provide an easy to understand, intuitive insight into the effects of visibility and visual quality on the scenarios being analyzed. Quantitative results further enhance this information with mathematically comparable data.

The conclusions derived from this study is summarized in the list below:

- Visibility surfaces provide a useful mechanism to understand the multi-visibility characteristics over a terrain.
- Weighted zones can be used to query for different visual preferences.

- Automation of some characteristics should be used to improve the practicality of visibility studies such as the automated calculation of the viewing angles on the paths.
- For quantitative comparison of different paths, total and average aggregations over weighted counts provide a good starting point.
- For larger data sets, some kind of approximation method should be applied in order to reach practical computation speeds.
- Open source public GIS software need to be enhanced with further visibility capabilities in order to be useful in this area.

Further research is possible for many of the aspects of this study. For example, a more general *moving observer* analysis can be introduced taking into account the speed, time, and the independent direction of view of different observers. Many real life scenarios (for example, a passenger in a tour bus) would need more parameters in this area.

The involvement of the enhanced visibility queries as discussed in the works of Fisher (1996a) can also be useful for some test cases. These enhanced queries rely on the availability of local and global offset visibility computations which can also be used in a weighting system.

Another research possibility is the improvement of the weighting system to better simulate real test cases. The visual effect of a target may be affected by different factors, such as the atmospheric conditions, lighting, the speed of the observer, the type of the target, etc. These types of studies usually deal with the visual impact of an object on an area. Hence, different objects such as the buildings and towers can also be incorporated into the study.

Error assessment and modeling can also be applied in order to reveal the effects of the DEM errors on the results. This can be used as an additional weighting criteria when constructing the visibility surfaces.

Further research can also be conducted on the algorithms used for the visibility computations. Several approximated algorithms exist as mentioned in Section

2.3, which provide acceptable levels of accuracy while being less computationally expensive.

The approaches used in this study can also be combined with least-cost analysis methods. The path with the highest average or total visibility value can be computed and displayed for a given terrain. However, this type of calculations would be highly expensive in terms of computational resources.

REFERENCES

Anile, M., Furno, P., Gallo, G., and Massolo, A. (2003). A fuzzy approach to visibility maps creation over digital terrains. *Fuzzy Sets and Systems*, 135(1):63–80.

Bishop, I. (2003). Assessment of visual qualities, impacts, and behaviours, in the landscape, by using measures of visibility. *Environment and Planning B: Planning and Design*, 30(5):677–688.

Bresenham, J. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

Caldwell, D., Mineter, M., Dowers, S., and Gittings, B. (2003). Analysis and visualization of visibility surfaces. In *Proceedings of the International Conference on GeoComputation*, pages 751–763. GeoComputation International Steering Group.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2008). *Computational Geometry: Algorithms and Applications*. Published by Springer-Verlag, third edition.

de By, R. A., editor (2001). *Principles of Geographic Information Systems*. Published by The International Institute for Aerospace Survey and Earth Sciences (ITC), second edition.

de By, R. A. and Kainz, W. (2001). Geographic information and spatial data types. In de By, R. A., editor, *Principles of Geographic Information Systems*, chapter 2, pages 37–66. Published by The International Institute for Aerospace Survey and Earth Sciences (ITC), second edition.

De Floriani, L. and Magillo, P. (2003). Algorithms for visibility computation on terrains: a survey. *Environment and Planning B: Planning and Design*, 30(5):709–728.

Ervin, S. and Steinitz, C. (2003). Landscape visibility computation: Necessary, but not sufficient. *Environment and Planning B: Planning and Design*, 30(5):757–766.

Felleman, J. (1979). *Landscape Visibility Mapping: Theory and Practice*. Published by the School of Landscape Architecture, State University of New York, College of Environmental Science and Forestry.

Fisher, P. (1994). Stretching the viewshed. In *Proceedings of the Symposium on Spatial Data Handling*, pages 725–738. International Society of Photogrammetry and Remote Sensing (ISPRS).

Fisher, P. (1996a). Extending the applicability of viewsheds in landscape planning. *PE & RS: Photogrammetric Engineering & Remote Sensing*, 62(11):1297–1302.

Fisher, P. (1996b). Reconsideration of the viewshed function in terrain modelling. *Geographical Systems*, 3:33–58.

Floriani, L. D. and Magillo, P. (1993). Algorithms for visibility computation on digital terrain models. In *Proceedings of the SIGAPP Symposium on Applied Computing*, pages 380–387. Association for Computing Machinery (ACM).

Franklin, W. and Ray, C. (1994). Higher isn't necessarily better: Visibility algorithms and experiments. In *Proceedings of the Symposium on Spatial Data Handling*, pages 751–763. International Society of Photogrammetry and Remote Sensing (ISPRS).

Gesch, D., Oimoen, M., Greenlee, S., Nelson, C., Steuck, M., and Tyler, D. (2002). The National Elevation Dataset. *PE & RS: Photogrammetric Engineering & Remote Sensing*, 68(1):5–11.

Izraelevitz, D. (2003). A fast algorithm for approximate viewshed computation. *PE & RS: Photogrammetric Engineering & Remote Sensing*, 69(7):767–774.

Kaučič, B. and Žalik, B. (2002). Comparison of viewshed algorithms on regular spaced points. In *Proceedings of the SIGGRAPH Spring Conference on Computer Graphics*, pages 177–183. Association for Computing Machinery (ACM).

Kim, Y., Rana, S., and Wise, S. (2004). Exploring multiple viewshed analysis using terrain features and optimisation techniques. *Computers & Geosciences*, 30(9-10):1019–1032.

Konecny, G. (2003). *Geoinformation: Remote Sensing, Photogrammetry and Geographic Information Systems*. Published by Taylor & Francis Inc, first edition.

Lee, J. (1989). *Coverage and Visibility Problems on Topographic Surfaces*. PhD thesis, University of Western Ontario.

Llobera, M. (2003). Extending GIS-based visual analysis: The concept of visualscapes. *International Journal of Geographical Information Science*, 17(1):25–48.

Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (2005). *Geographical Information Systems and Science*. Published by John Wiley & Sons Inc, second edition.

Lu, M., Zhang, J., Lv, P., and Fan, Z. (2008). Least visible path analysis in raster terrain. *International Journal of Geographical Information Science*, 22(6):645–656.

Maloy, M. and Dean, D. (2001). An accuracy assessment of various GIS-based viewshed delineation techniques. *PE & RS: Photogrammetric Engineering & Remote Sensing*, 67(11):1293–1298.

Mark, D. (1978). Concepts of data structure for digital terrain models. In *Proceedings of the Digital Terrain Models (DTM) Symposium*, pages 24–31. American Society of Photogrammetry & American Congress on Surveying and Mapping (ASP-ACSM).

Nackaerts, K., Govers, G., and Orshoven, J. (1999). Accuracy assessment of probabilistic visibilities. *International Journal of Geographical Information Science*, 13(7):709–721.

NASA JPL (2010a). ASTER Global Digital Elevation Map (GDEM). `http://asterweb.jpl.nasa.gov/gdem.asp`. [last accessed on 14-March-2010].

NASA JPL (2010b). ASTER Global Digital Elevation Map (GDEM) access and download. http://asterweb.jpl.nasa.gov/gdem-wist.asp. [last accessed on 14-March-2010].

OSGeo (2010). GeoTIFF official information and revision 1.0 specification. http://trac.osgeo.org/geotiff/. [last accessed on 23-January-2010].

Peucker, T., Fowler, R., Little, J., and Mark, D. (1978). The triangulated irregular network (TIN). In *Proceedings of the Digital Terrain Models (DTM) Symposium*, pages 516–540. American Society of Photogrammetry & American Congress on Surveying and Mapping (ASP-ACSM).

Quantum GIS Development Team (2010). Quantum GIS coding and compilation guide version 1.5 'tethys'. http://download.osgeo.org/qgis/doc/manual/qgis-1.5.0_coding-compilation_guide_en.pdf. [last accessed on 01-August-2010].

Riggs, P. and Dean, D. (2007). An investigation into the causes of errors and inconsistencies in predicted viewsheds. *Transactions in GIS*, 11(2):175–196.

Sherman, G. E. (2008). *Desktop GIS: Mapping the Planet with Open Source Tools*. Published by Pragmatic Bookshelf, first edition.

Tomlin, C. (1983). *Digital Cartographic Modeling Techniques in Environmental Planning*. PhD thesis, Yale University.

Travis, M., Elsner, G., Iverson, W., and Johnson, C. (1975). VIEWIT: Computation of seen areas, slope, and aspect for land-use planning. *USDA Forest Service, General Technical Report*, (PSW-11).

USGS (2010a). National Elevation Dataset (NED). http://ned.usgs.gov/. [last accessed on 11-May-2010].

USGS (2010b). Shuttle Radar Topography Mission (SRTM). http://srtm.usgs.gov/. [last accessed on 11-May-2010].

USGS (2010c). Spatial Data Transfer Standard (SDTS). http://mcmcweb.er.usgs.gov/sdts/. [last accessed on 12-May-2010].

van Zyl, J. (2001). The Shuttle Radar Topography Mission (SRTM). *Acta Astronautica*, 48(5-12):559–565.

Ware, J., Kidner, D., and Rallings, P. (1998). Parallel distributed viewshed analysis. In *Proceedings of the Symposium on Advances in Geographic Information Systems*, pages 151–156. Association for Computing Machinery (ACM).

Wheatley, D. (1995). Cumulative viewshed analysis: a GIS-based method for investigating intervisibility, and its archaeological application. *Archaeology and Geographical Information Systems: A European Perspective*, pages 171–185.

Wu, Y., Bishop, I., Hossain, H., and Sposito, V. (2006). Using gis in landscape visual quality assessment. *Applied GIS*, 2(3):18–18.

Yoeli, P. (1985). The making of intervisibility maps with computer and plotter. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 22(3):88–103.

APPENDIX A

SOURCE CODE FOR THE SOFTWARE

In this chapter, source codes for the software implementation of the methodology are given. The codes are presented without any artificial line wraps and with line breaks using the syntax of the Python language. The files with the extensions of '.py' are Python files. File extensions of '.ui' indicates the Qt XML files used to defining GUI windows. The syntax is highlighted using bold text style according to the keywords of the Python and XML languages for easier readability. Appropriate comments have been added in code syntax throughout the source code to explain specific parts.

A.1 __init__.py

This file is loaded with the plugin module by Quantum GIS in the beginning. It contains basic information like name and description.

```
1  # -*- coding: utf-8 -*-
2
3  VERSION = "1.0"
4
5  #******************************************************************************
6  #
7  # The information about this plugin can be found in the 'vap.py' file...
8  #
9  #******************************************************************************
10
11 def name():
```

```
12          return "Visibility Analysis for Paths"
13
14  def description():
15          return "This plugin calculates weighted multi-visibility surfaces and "+\
16                                  "path value profiles for the directional paths"
17
18  def version():
19          return VERSION
20
21  def qgisMinimumVersion():
22          return "1.5"
23
24  def authorName():
25          return "Cagil Seker"
26
27  def classFactory( iface ):
28          from vap import VAP
29          return VAP( iface )
```

A.2 vap.py

This file holds the main class for the plugin. The initialization routine that prepares the plugin GUI resides here.

```
1   # -*- coding: utf-8 -*-
2
3   #*****************************************************************************
4   #
5   # Visibility Analysis for Paths - version 1.0
6   # ---------------------------------------------------------
7   # This software works as a plugin for the Quantum GIS software v1.5+.
8   # It has been developed as supplementary to the thesis titled
9   # "Weighted Multi-Visibility Analysis on Directional Paths".
10  #
11  # Copyright (C) 2010 Cagil Seker (cagils@gmail.com)
12  #
13  # This source is free software; you can redistribute it and/or modify it under
14  # the terms of the GNU General Public License as published by the Free
```

```
15  # Software Foundation; either version 2 of the License, or (at your option)
16  # any later version.
17  #
18  # This code is distributed in the hope that it will be useful, but WITHOUT ANY
19  # WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
20  # FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
21  # details.
22  #
23  # A copy of the GNU General Public License is available on the World Wide Web
24  # at <http://www.gnu.org/copyleft/gpl.html>. You can also obtain it by writing
25  # to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
26  # MA 02111-1307, USA.
27  #
28  #****************************************************************************
29
30  from PyQt4.QtCore import *
31  from PyQt4.QtGui import *
32  from qgis.core import *
33  from qgis.gui import *
34
35  import resources_rc
36  import dialogs
37  import vap_io
38
39  from . import VERSION
40
41
42  # Visibility Analysis Plugin (VAP) main class
43  class VAP:
44
45
46      def __init__(self, iface):
47          """Initializes the plugin while QGIS is starting"""
48
49          self.iface = iface
50          self.canvas = iface.mapCanvas()
51
52
53      def initGui(self):
54          """Initializes the GUI after __init__ while QGis is starting"""
```

```
55
56          self.mainDialog = dialogs.MainDialog(self.iface.mainWindow(),
57                                                              self.canvas)
58
59          # create the main action and bind run method
60          self.action = QAction( QIcon(":/VAP_icon.png") , "Visibility Analysis",
61                                                      self.iface.mainWindow() )
62          self.action.setWhatsThis("Performs visibility analysis for a DEM "+\
63                                                              "raster layer")
64          self.action.setStatusTip("Visibility Analysis Plugin")
65          QObject.connect(self.action, SIGNAL("triggered()"), self.run)
66
67          # add VAP to the QGIS interface
68          self.iface.addToolBarIcon(self.action)
69          self.iface.addPluginToMenu("&Visibility Analysis", self.action)
70          self.iface.registerMainWindowAction(self.action, "F7")
71
72
73      def unload(self):
74          """Unloads the plugin from the QGIS. This is called when the plugin
75          is unloaded from the plugins menu and while QGIS is closing."""
76
77          # remove VAP from the QGIS interface
78          self.iface.removePluginMenu("&Visibility Analysis", self.action)
79          self.iface.removeToolBarIcon(self.action)
80          self.iface.unregisterMainWindowAction(self.action)
81          # remove the key binding
82
83
84      # this is called when the plugin starts
85      def run(self):
86          """Runs the plugin"""
87
88          self.mainDialog.initDialog()
89          self.mainDialog.show()
```

A.3 vap_io.py

This file mainly stores the helper methods and classes that deal with the input and output operations for data files and map layers.

```
# -*- coding: utf-8 -*-

#******************************************************************************
#
# The information about this plugin can be found in the 'vap.py' file...
#
#******************************************************************************

from __future__ import with_statement

from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
from qgis.gui import *

import osgeo.gdal as gdal
import osgeo.gdalconst as gdalconst

import numpy
import os
import math

from string import rfind


def getLayers(filterMethod):
    """Returns the sorted list of (LayerId, MapLayer) tuples
    filtered by the filterMethod."""

    layersDict = QgsMapLayerRegistry.instance().mapLayers()
    # this returns a dictionary
    layers = sorted(layersDict.items(), key=lambda k: k[1].name())

    filteredLayers = filter(filterMethod, layers)
```

```
35      return filteredLayers
36
37
38  def layerFilterRaster(layer):
39      return layer[1].type() == QgsMapLayer.RasterLayer
40
41
42  def layerFilterVector(layer):
43      return layer[1].type() == QgsMapLayer.VectorLayer
44
45
46  def layerFilterAll(layer):
47      return True
48
49
50  def layerFilterPoly(layer):
51      return (layer[1].type() == QgsMapLayer.VectorLayer) and \
52                                  (layer[1].geometryType() == QGis.Polygon)
53
54
55  def layerFilterLine(layer):
56      return (layer[1].type() == QgsMapLayer.VectorLayer) and \
57                                  (layer[1].geometryType() == QGis.Line)
58
59
60  def drawAB(pathLayer, dialog):
61      if pathLayer.wkbType() != QGis.WKBLineString:
62          return False
63      if pathLayer.selectedFeatureCount() != 1:
64          return False
65
66      selFeat = pathLayer.selectedFeatures()
67      path = selFeat[0]
68      pathPoints = path.geometry().asPolyline()
69      firstPoint = pathPoints[0]
70      lastPoint = pathPoints[-1]
71
72      dialog.charMark1 = CharMarker(dialog.canvas, "A", True, firstPoint)
73      dialog.charMark2 = CharMarker(dialog.canvas, "B", True, lastPoint)
74
```

```
75      return True
76
77  # The class to mark the A and B characters on the path
78  class CharMarker(QgsMapCanvasItem):
79
80
81      def __init__(self, canvas, char, withBox = True,
82                                              initPos = QgsPoint(0.0, 0.0)):
83          QgsMapCanvasItem.__init__(self, canvas)
84
85          self.deleted = False
86          self.font = QFont()
87          self.font.setPointSize(8)
88          self.qRect = QFontMetricsF(self.font).boundingRect("."+char+".")
89          self.qRect.moveCenter(QPointF(0.0, 0.0))
90          self.char = char
91
92          self.withBox = withBox
93          self.setVisible(False)
94          self.setPosition(initPos)
95
96
97      def delete(self):
98          self.deleted = True
99          self.setVisible(False)
100
101
102     def setPosition(self, pos):
103         self.mapPos = pos
104         self.canvasPos = self.toCanvasCoordinates(pos)
105         self.setPos(self.canvasPos)
106         self.setVisible(True)
107
108
109     def paint(self, painter, option, widget):
110         if not self.deleted:
111             if self.withBox:
112                 pen = QPen(QColor(0, 0, 0))
113                 pen.setWidth(1)
114                 painter.setPen(pen)
```

```
115                 painter.setOpacity(0.7)
116                 painter.fillRect(self.qRect, QColor(0, 255, 0))
117                 painter.setOpacity(1)
118                 painter.drawRect(self.qRect)
119 
120             painter.setFont(self.font)
121             painter.drawText(self.qRect, Qt.AlignCenter, self.char)
122 
123 
124     def boundingRect(self):
125         return self.qRect
126 
127 
128     def updatePosition(self):
129         self.setPos(self.toCanvasCoordinates(self.mapPos))
130 
131 # The class to mark the currently analyzed location
132 class XMarker(CharMarker):
133 
134 
135     def __init__(self, canvas, resolution, initPos = QgsPoint(0.0, 0.0)):
136         QgsMapCanvasItem.__init__(self, canvas)
137 
138         self.deleted = False
139         self.canvas = canvas
140         self.resolution = resolution
141 
142         self.calculateRect()
143 
144         self.setVisible(False)
145         self.setPosition(initPos)
146 
147 
148     def calculateRect(self):
149             scale = int(self.calculateScale())
150             if scale % 2 != 0:
151                 scale += 1
152             radius = math.sqrt(2) * scale
153             outScale = int(scale + radius + 10)
154             if outScale % 2 != 0:
```

```
155                 outScale += 1
156             self.qRectPixel = QRectF(QRect(0, 0, scale, scale))
157             if scale < 10:
158                 outScale = 24
159             self.qRect = QRectF(QRect(0, 0, outScale, outScale))
160             self.qRect.moveCenter(QPointF(int(scale/2), int(scale/2)))
161 
162 
163     def paint(self, painter, option, widget):
164         if not self.deleted:
165             self.calculateRect()
166 
167             #penBorder = QPen(QColor(200, 0, 0))
168             #penBorder.setWidth(2)
169 
170             brushPixel = QBrush(QColor(0, 220, 0))
171             brushOuter = QBrush(QColor(0, 0, 200))
172 
173             painter.setOpacity(0.3)
174             painter.setBrush(brushOuter)
175             painter.setPen(Qt.black)
176             painter.drawEllipse(self.qRect)
177 
178             painter.setOpacity(0.8)
179             painter.setBrush(brushPixel)
180             painter.setPen(Qt.NoPen)
181             painter.drawRect(self.qRectPixel)
182 
183 
184     def calculateScale(self):
185         zoom = self.canvas.scale()
186         dpi = self.canvas.mapRenderer().outputDpi()
187         dpm = float(dpi) * 100.0 / 2.54
188         scale = int(dpm / float(zoom) * self.resolution)
189         return scale
190 
191 # This class is a simple storage for the input data
192 class VapInput(object):
193 
194 
```

```
195     def __init__(self):
196         self.elevationLayer = None
197         self.pathLayer = None
198         self.obsHeight = 0.0
199         self.startPoint = "A"
200         self.viewAngle = 360.0
201         self.distLimit = 100
202         self.polyLayer = None
203         self.weightAttrIndex = None
204         self.weightAttrName = ""
205         self.limitPoly = False
206         self.distFactor = 0.0
207         self.descText = ""
208 
209 # This class handles the input/output type of operations
210 class RasterIO(object):
211 
212     # open and prepare the elevation raster layer
213     def __init__(self, rasterLayer, dialog):
214         self.rasterLayer = rasterLayer
215         self.layerPath = str(self.rasterLayer.source())
216 
217         srcBasename, srcExtension = os.path.splitext(self.layerPath)
218         self.outFileName = srcBasename + "_cva" + ".tiff"
219         self.pathFileName = srcBasename + "_path" + ".tiff"
220 
221         self.dialog = dialog
222 
223         # use GDAL library to open raster data
224         self.sourceDataset = gdal.Open(self.layerPath, gdal.GA_ReadOnly)
225         if self.sourceDataset is None:
226             return None
227 
228         self.rasterXSize = self.sourceDataset.RasterXSize
229         self.rasterYSize = self.sourceDataset.RasterYSize
230         bandCount = self.sourceDataset.RasterCount
231 
232         band = self.sourceDataset.GetRasterBand(1)
233 
234         # read all of the band in a NumPy type of array
```

```
235         self.data = band.ReadAsArray(0, 0, self.rasterXSize, self.rasterYSize)
236 
237         # construct empty arrays for the output layers
238         self.cumulativeArray = \
239             numpy.zeros(shape=(self.rasterYSize, self.rasterXSize),
240                                                             dtype=float)
241         self.cumulativeArray.fill(-99999)
242         self.pathCumulativeArray = \
243             numpy.zeros(shape=(self.rasterYSize, self.rasterXSize),
244                                                             dtype=float)
245         self.pathCumulativeArray.fill(-99999)
246 
247         self.gdalDriver = self.sourceDataset.GetDriver()
248 
249     # read a specific coordinate from the DEM layer
250     def getValue(self, x, y):
251 
252         value = self.data[y, x] # math matrix notation is row number first!
253 
254         return value
255 
256     # add a value to the visibility count for the targets
257     def addValue(self, xy, value):
258         (x, y) = xy
259         oldValue = self.cumulativeArray[y, x]
260         if oldValue == -99999:
261             oldValue = 0
262         self.cumulativeArray[y, x] = oldValue + value
263 
264     # add a value to the visibility count for the path
265     def addPathValue(self, xy, value):
266         (x, y) = xy
267         oldValue = self.pathCumulativeArray[y, x]
268         if oldValue == -99999:
269             oldValue = 0
270         self.pathCumulativeArray[y, x] = oldValue + value
271 
272     # create the raster map on the file system for output
273     def createRasterLayer(self):
274         fileName = self.outFileName
```

```
275
276         while (os.path.isfile(fileName)):
277             base, ext = os.path.splitext(fileName)
278             i_under = rfind(base,"_")
279             if i_under == -1:
280                 fileName = base + "_1" + ext
281                 continue
282             if i_under+1 == len(base):
283                 fileName = base + "1" + ext
284                 continue
285             rest = base[i_under+1:]
286             try: # to see if this is a number...
287                 num = int(rest)
288             except ValueError: # if not, treat it as part of the name
289                 fileName = base + "_1" + ext
290                 continue
291
292             fileName = base[:i_under+1] + str(num+1) + ext
293             continue
294
295         self.targetDataset = None
296         try:
297             self.targetDataset = self.gdalDriver.CreateCopy(fileName,
298                                                             self.sourceDataset,0)
299         except IOError, e:
300             err = str(type(e)) + " : " + str(e)
301             self.dialog.showError(err)
302             return False
303
304         # write the output array onto the file
305         self.targetDataset.GetRasterBand(1).WriteArray(self.cumulativeArray)
306
307         self.targetDataset = None
308
309         layerName, ext = os.path.splitext(os.path.basename(fileName))
310
311         rOutLayer = QgsRasterLayer(fileName, layerName)
312         self.outLayer = rOutLayer
313
314         rOutLayer.setDrawingStyle(QgsRasterLayer.SingleBandGray)
```

```
315         rOutLayer.setTransparency(255)
316         rOutLayer.setNoDataValue(-99999)
317         #(min, max) = rOutLayer.computeMinimumMaximumEstimates(1)
318         stats = rOutLayer.bandStatistics(1)
319         rOutLayer.setMinimumValue(1, stats.minimumValue)
320         rOutLayer.setMaximumValue(1, stats.maximumValue)
321         rOutLayer.setUserDefinedGrayMinimumMaximum(True)
322         rOutLayer.setColorShadingAlgorithm(QgsRasterLayer.UserDefinedShader)
323         rOutLayer.setContrastEnhancementAlgorithm(
324                         QgsContrastEnhancement.StretchToMinimumMaximum, True)
325 
326         #rOutLayer.setColorShadingAlgorithm(QgsRasterLayer.ColorRampShader)
327         #lst = [ (0, QColor(0,255,0)), (255, QColor(255,255,0)) ]
328         #shader = rOutLayer.rasterShader()
329         #func = shader.rasterShaderFunction()
330         #shader.setColorRampType(QgsColorRampShader.INTERPOLATED)
331         #shader.setColorRampItemList(lst)
332 
333         registry = QgsMapLayerRegistry.instance()
334         registry.addMapLayer(rOutLayer)
335 
336         return True
337 
338     # create the raster map for the path on the file system
339     def createPathRasterLayer(self):
340         fileName = self.pathFileName
341 
342         while (os.path.isfile(fileName)):
343             base, ext = os.path.splitext(fileName)
344             i_under = rfind(base,"_")
345             if i_under == -1:
346                 fileName = base + "_1" + ext
347                 continue
348             if i_under+1 == len(base):
349                 fileName = base + "1" + ext
350                 continue
351             rest = base[i_under+1:]
352             try: # to see if this is a number...
353                 num = int(rest)
354             except ValueError: # if not, treat it as part of the name
```

```
355                 fileName = base + "_1" + ext
356                 continue
357
358             fileName = base[:i_under+1] + str(num+1) + ext
359             continue
360         self.pathDataset = None
361         try:
362             self.pathDataset = self.gdalDriver.CreateCopy(fileName,
363                                                 self.sourceDataset, 0)
364         except IOError, e:
365             err = str(type(e)) + " : " + str(e)
366             self.dialog.showError(err)
367             return False
368
369         # write the array map onto the file
370         self.pathDataset.GetRasterBand(1).WriteArray(self.pathCumulativeArray)
371
372         self.pathDataset = None
373         #self.sourceDataset = None
374
375         layerName, ext = os.path.splitext(os.path.basename(fileName))
376
377         rOutLayer = QgsRasterLayer(fileName, layerName)
378         self.outPathLayer = rOutLayer
379
380         rOutLayer.setDrawingStyle(QgsRasterLayer.SingleBandPseudoColor)
381         rOutLayer.setTransparency(255)
382         rOutLayer.setNoDataValue(-99999)
383         #(min, max) = rOutLayer.computeMinimumMaximumEstimates(1)
384         stats = rOutLayer.bandStatistics(1)
385         rOutLayer.setMinimumValue(1, stats.minimumValue)
386         rOutLayer.setMaximumValue(1, stats.maximumValue)
387         rOutLayer.setUserDefinedGrayMinimumMaximum(True)
388         rOutLayer.setColorShadingAlgorithm(QgsRasterLayer.PseudoColorShader)
389         rOutLayer.setContrastEnhancementAlgorithm(
390                         QgsContrastEnhancement.StretchToMinimumMaximum, True)
391
392         registry = QgsMapLayerRegistry.instance()
393         registry.addMapLayer(rOutLayer)
394
```

```
395         return True
396 
397     # determine the suitable file name for the outputs
398     def findResultsFile(self):
399         srcBasename, srcExtension = os.path.splitext(self.layerPath)
400         fileName = "_results" + ".txt"
401 
402         while (os.path.isfile(fileName)):
403             base, ext = os.path.splitext(fileName)
404             i_under = rfind(base,"_")
405             if i_under == -1:
406                 fileName = base + "_1" + ext
407                 continue
408             if i_under+1 == len(base):
409                 fileName = base + "1" + ext
410                 continue
411             rest = base[i_under+1:]
412             try: # to see if this is a number...
413                 num = int(rest)
414             except ValueError: # if not, treat it as part of the name
415                 fileName = base + "_1" + ext
416                 continue
417 
418             fileName = base[:i_under+1] + str(num+1) + ext
419             continue
420 
421         self.outTextFileName = fileName
422 
423 
424     def writeResultsFile(self, results):
425         f = open(self.outTextFileName, "w")
426         with f:
427             f.write(results)
428 
429         return
```

A.4 vap_analyze.py

This file holds the Analyzer class that is responsible for the main visibility analysis operations.

```
# -*- coding: utf-8 -*-

#*****************************************************************************
#
# The information about this plugin can be found in the 'vap.py' file...
#
#*****************************************************************************

import vap_io
from qgis.core import *
from qgis.gui import *
from itertools import izip
import time
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import math

import dialogs

# create and start the analyzer
def startAnalyzer(dialog, vapInput):
    analyzer = Analyzer(dialog, vapInput)
    analyzer.start()
    return


# a helper method to generate a list in successive pairs
# used for path segments
def pairWise(list):
    listIter = iter(list)
    first = listIter.next()
    for second in listIter:
        yield first, second
        first = second
```

```
35
36
37  # rotates a point on a coordinate system by an angle
38  def rotatePoint(x, y, angleD):
39      # if angle is positive and y-axis is positive on the upside direction,
40      # then this rotates the point clock-wise
41      # angle should be between -180 and 180 degrees
42      if angleD < -180.0 or angleD > 180.0:
43          return None
44
45      if angleD == 0.0:
46          return x, y
47
48      angleRad = abs(angleD) * math.pi / 180.0
49
50      if(angleD > 0.0):
51          xr = x * math.cos(angleRad) - y * math.sin(angleRad)
52          yr = x * math.sin(angleRad) + y * math.cos(angleRad)
53      else:
54          xr = x * math.cos(angleRad) + y * math.sin(angleRad)
55          yr = -x * math.sin(angleRad) + y * math.cos(angleRad)
56
57      return xr, yr
58
59
60  # Bresenham's rasterization algorithm. Optimized for performance
61  def bresenham(includeTarget, x0, y0, x1, y1):
62      steep = abs(y1 - y0) > abs(x1 - x0)
63      if steep:
64          x0, y0 = y0, x0
65          x1, y1 = y1, x1
66      reverse = x0 > x1
67      deltaX = abs(x1 - x0)
68      deltaY = abs(y1 - y0)
69      error = int(deltaX / 2)
70      y = y0
71      if y0 < y1:
72          yStep = 1
73      else:
74          yStep = -1
```

```
75      if reverse:
76          r = xrange(x0, x1-includeTarget, -1)
77      else:
78          r = xrange(x0, x1+includeTarget)
79      for x in r:
80          if steep:
81              yield (y, x)
82          else:
83              yield (x, y)
84          error -= deltaY
85          if error < 0:
86              y += yStep
87              error += deltaX
88
89
90  # The main class for the analyzer
91  class Analyzer(object):
92
93
94      # get the input variables
95      def __init__(self, dialog, vapInput):
96          self.dialog = dialog
97          self.vapInput = vapInput
98
99          self.extent = self.vapInput.elevationLayer.extent()
100         self.width = self.vapInput.elevationLayer.width()
101         self.height = self.vapInput.elevationLayer.height()
102         self.resolution = self.vapInput.elevationLayer.rasterUnitsPerPixel()
103
104         if self.vapInput.viewAngle == 360.0:
105             self.fullView = True
106         else:
107             self.fullView = False
108
109         self.mtp = QgsMapToPixel(self.resolution, self.height,
110                                  self.extent.yMinimum(), self.extent.xMinimum())
111
112         self.path = self.vapInput.pathLayer.selectedFeatures()[0]
113         self.pathPoints = self.findPathPoints(self.path)
114         self.currentObsHeight = self.vapInput.obsHeight
```

```
115
116         self.rasterIO = vap_io.RasterIO(self.vapInput.elevationLayer, dialog)
117
118         self.notFinished = True
119
120
121     # main method for the visibility analysis
122     def start(self):
123         self.dialog.setStatus(\
124                         "Visibility analysis process has been started...")
125         self.dialog.showProgressBar(0)
126         t0 = time.time()
127         self.totalPathValue = 0
128         self.totalObservers = 0
129         self.totalTargets = 0
130
131         # get the segments iterator from the path
132         rPathSegmentsIter = self.rasterSegmentGenerator(self.pathPoints)
133
134         viewAngle = self.vapInput.viewAngle
135
136         # get the polygons iterator from the weight zone layer
137         polyListIter = self.polyGenerator(self.vapInput.polyLayer)
138         self.polyList = list(polyListIter)
139
140         # prepare the mark for the observer location
141         observerMark = vap_io.XMarker(self.dialog.canvas, self.resolution)
142
143         # outer loop 1 for every line segment of the path...
144         for rSegmentCoordsF in rPathSegmentsIter:
145
146             # transform the coordinates into integer cell coordinates
147             rSegmentCoords = map(int, map(round, rSegmentCoordsF))
148
149             # use the Bresenham to find the cells along the segment
150             rSegmentPointsIter = bresenham(0, *rSegmentCoords)
151
152             # outer loop 2 for every cell along the segment...
153             for rObserver in rSegmentPointsIter:
154                 self.totalObservers += 1
```

```
155                 self.showMark(rObserver, observerMark)
156
157                 # 'v...' = map coordinates
158                 # 'r...' = integer raster coordinates
159                 # '...F' = floating raster coordinates
160                 vObserver = self.mtp.toMapCoordinates(*rObserver)
161                 self.rasterIO.addPathValue(rObserver, 0)
162
163                 if not self.fullView:
164                     # calculate the view field polygon for the direction
165                     self.viewFieldPolyGeo = self.viewFieldFromSegment(
166                                                 rObserver, *rSegmentCoordsF)
167
168                 # generate all of the target points for the current cell
169                 rTargetPointsIter = self.rasterTargetGenerator(vObserver)
170
171                 # inner loop for every target cell...
172                 for rTarget in rTargetPointsIter:
173                     if self.dialog.progress.wasCanceled():
174                         self.dialog.progress.reset()
175                         observerMark.delete()
176                         del observerMark
177                         self.dialog.setStatus("")
178                         return -1
179                     self.totalTargets += 1
180                     qApp.processEvents(QEventLoop.AllEvents)
181
182                     # calculate visibility between observer and target
183                     visible = self.visibility(rObserver, rTarget)
184
185                     # find the weight of the target and whether
186                     # it is a polygon
187                     weight, wasPolygon = self.findWeight(rTarget)
188
189                     # calculate distance factor
190                     df = float(self.vapInput.distFactor)
191                     dist = float(self.resolution *
192                         math.sqrt(QgsPoint(*rObserver).sqrDist(*rTarget)))
193
194                     if (not self.vapInput.limitPoly) or (wasPolygon):
```

```
                        if visible:
                            if df <> 0.0:
                                weight /= 2**(dist/df)
                            # add the weighted counts
                            self.rasterIO.addValue(rTarget, weight)
                            self.rasterIO.addPathValue(rObserver, weight)
                            self.totalPathValue += weight
                        else:
                            # not visible
                            self.rasterIO.addValue(rTarget, 0)

        self.dialog.progress.reset()
        t1 = time.time()
        self.timeDiff = t1 - t0

        observerMark.delete()
        del observerMark
        self.dialog.setStatus("")

        # create the output files
        self.rasterIO.createRasterLayer()
        self.rasterIO.createPathRasterLayer()
        self.sourceDataset = None
        self.rasterIO.findResultsFile()

        # calculate the average path value
        averagePathValue = self.totalPathValue / float(self.totalObservers)

        timeDiffStr = str("%.2f" % self.timeDiff)
        self.resultsStr = "Results for Weighted Multi Visibility Analysis on "\
             "Directional Paths (WMVADP)\n"\
             "==========================================\n"\
             "Descriptive Text: " + self.vapInput.descText + "\n"\
             "Elevation Map: " + self.vapInput.elevationLayer.name() + ""\
             " (" + self.vapInput.elevationLayer.source() + ")\n"\
             "Path Layer: " + self.vapInput.pathLayer.name() + ""\
             " (" + self.vapInput.pathLayer.source() + ")\n"\
             "Observer Height: " + str(self.vapInput.obsHeight) + "\n"\
             "Starting Point: '" + self.vapInput.startPoint + "'\n"\
             "View Angle: " + str(self.vapInput.viewAngle) + "\n"\
```

```
235             "Distance Limit: " + str(self.vapInput.distLimit) + "\n"\
236             "Polygon Layer: " + self.vapInput.polyLayer.name() + ""\
237             " (" + self.vapInput.polyLayer.source() + ")\n"\
238             "Weight Attribute: " + self.vapInput.weightAttrName + "\n"\
239             "Limit to Polygons: " + str(self.vapInput.limitPoly)  + "\n"\
240             "Distance Factor: " + str(self.vapInput.distFactor) + "\n"\
241             "-----------------------------------------------\n"\
242             "Total Path Value: " + str(self.totalPathValue) + "\n"\
243             "Observer Point Count: " + str(self.totalObservers) + "\n"\
244             "Target Points Analyzed: " + str(self.totalTargets) + "\n"\
245             "Average Path Value: " + str(averagePathValue) + "\n"\
246             "Analysis Time: " + timeDiffStr + " seconds \n"\
247             "Output Layer for the Targets: " + \
248             self.rasterIO.outLayer.name() + "\n"\
249             "Output Layer for the Path: " + \
250             self.rasterIO.outPathLayer.name() + "\n"\
251             "Results Text File : " + self.rasterIO.outTextFileName + "\n"\
252             "-----------------------------------------------\n"
253
254         self.rasterIO.writeResultsFile(self.resultsStr)
255         self.showResults()
256         return
257
258
259     # show the results dialog
260     def showResults(self):
261         self.resultsDialog = dialogs.ResultsDialog(self.dialog)
262         self.resultsDialog.initDialog()
263         self.resultsDialog.editResults.setText(self.resultsStr)
264         self.resultsDialog.show()
265
266         return
267
268
269     # This helper method is used for the view field polygon
270     # generation. The border lines of the map area
271     # are numbered and we use these numbers to find out
272     # which borders should be included in the view field
273     def findLineNo(self, p):
274         corner = 0
```

```
275         line = None
276         if int(p.x()) == self.width:
277             line = 1
278             if int(p.y()) == 0:
279                 line = 2 # skip corner
280         elif int(p.y()) == 0:
281             line = 2
282             if int(p.x()) == 0:
283                 line = 3 # skip corner
284         elif int(p.x()) == 0:
285             line = 3
286             if int(p.y()) == self.height:
287                 line = 0 # skip corner
288         elif int(p.y()) == self.height:
289             line = 0
290 
291         return line
292 
293 
294     # calculate the view field using the current segment and
295     # observer location...
296     # we use a new coordinate system with the origin at
297     # the observer for the rotation calculations
298     def viewFieldFromSegment(self, rObserver, x0, y0, xp, yp):
299         x, y = rObserver
300 
301         # if the start and finish are same...
302         if x0 == y0 and xp == yp:
303             return None
304 
305         if xp-x0 == 0.0:
306             range = -10 if yp < y0 else 10
307             ypFar = yp + range
308             xpFar = xp
309             x = x0
310         elif yp-y0 == 0.0:
311             range = -10 if xp < x0 else 10
312             ypFar = yp
313             xpFar = xp + range
314             y = y0
```

```
315             else:
316                 a = (yp-y0) / float((xp-x0))
317                 b = yp-a*float(xp)
318 
319                 if abs(xp-x0) > abs(yp-y0):
320                     y = a * x + b
321                     range = -10 if xp < x0 else 10
322                     xpFar = xp + range
323                     ypFar = a * xpFar + b
324                 else:
325                     x = (y - b)/a
326                     range = -10 if yp < y0 else 10
327                     ypFar = yp + range
328                     xpFar = (ypFar - b) / a
329 
330             x0, y0 = x, y
331             txp, typ = xpFar-x0, ypFar-y0
332             rtxp, rtyp = txp, typ
333             angle = self.vapInput.viewAngle
334             rotateAngle = angle
335             if angle > 180.0:
336                 rotateAngle = angle - 360.0
337                 rtxp, rtyp = -rtxp, -rtyp
338 
339             # find the limits of the view field based on the angle
340             txv1, tyv1 = rotatePoint(rtxp, rtyp, rotateAngle/2.0)
341             txv2, tyv2 = rotatePoint(rtxp, rtyp, -rotateAngle/2.0)
342 
343             tborders = QgsRectangle(-x0, -y0, self.width-x0, self.height-y0)
344             tXLimit1, tYLimit1, tXLimit2, tYLimit2 = None, None, None, None
345             borderPoint1, borderPoint2 = None, None
346 
347             # check for the overflows...
348             if txv1 > 0.0:
349                 tXLimit1 = tborders.xMaximum()
350             elif txv1 < 0.0:
351                 tXLimit1 = tborders.xMinimum()
352 
353             if tyv1 > 0.0:
354                 tYLimit1 = tborders.yMaximum()
```

```
355         elif tyv1 < 0.0:
356             tYLimit1 = tborders.yMinimum()
357 
358         if txv2 > 0.0:
359             tXLimit2 = tborders.xMaximum()
360         elif txv2 < 0.0:
361             tXLimit2 = tborders.xMinimum()
362 
363         if tyv2 > 0.0:
364             tYLimit2 = tborders.yMaximum()
365         elif tyv2 < 0.0:
366             tYLimit2 = tborders.yMinimum()
367 
368         if tXLimit1 is None:
369             borderPoint1 = QgsPoint(float(0.0+x0), float(tYLimit1+y0))
370         if tYLimit1 is None:
371             borderPoint1 = QgsPoint(float(tXLimit1+x0), float(0.0+y0))
372         if tXLimit2 is None:
373             borderPoint2 = QgsPoint(float(0.0+x0), float(tYLimit2+y0))
374         if tYLimit2 is None:
375             borderPoint2 = QgsPoint(float(tXLimit2+x0), float(0.0+y0))
376 
377         # find the coordinates of the view field at the intersections
378         if borderPoint1 is None:
379             tcoef1 = tyv1 / float(txv1)
380             if tcoef1 >= 1.0:
381                 txBorder1 = tYLimit1 / tcoef1
382                 if abs(txBorder1) >= abs(tXLimit1):
383                     tyBorder1 = tXLimit1 * tcoef1
384                     borderPoint1 = QgsPoint(float(tXLimit1+x0), tyBorder1+y0)
385                 else:
386                     borderPoint1 = QgsPoint(txBorder1+x0, float(tYLimit1+y0))
387             else:
388                 tyBorder1 = tXLimit1 * tcoef1
389                 if abs(tyBorder1) >= abs(tYLimit1):
390                     txBorder1 = tYLimit1 / tcoef1
391                     borderPoint1 = QgsPoint(txBorder1+x0, float(tYLimit1+y0))
392                 else:
393                     borderPoint1 = QgsPoint(float(tXLimit1+x0), tyBorder1+y0)
394 
```

```
395         if borderPoint2 is None:
396             tcoef2 = tyv2 / float(txv2)
397             if tcoef2 >= 1.0:
398                 txBorder2 = tYLimit2 / tcoef2
399                 if abs(txBorder2) >= abs(tXLimit2):
400                     tyBorder2 = tXLimit2 * tcoef2
401                     borderPoint2 = QgsPoint(float(tXLimit2+x0), tyBorder2+y0)
402                 else:
403                     borderPoint2 = QgsPoint(txBorder2+x0, float(tYLimit2+y0))
404             else:
405                 tyBorder2 = tXLimit2 * tcoef2
406                 if abs(tyBorder2) >= abs(tYLimit2):
407                     txBorder2 = tYLimit2 / tcoef2
408                     borderPoint2 = QgsPoint(txBorder2+x0, float(tYLimit2+y0))
409                 else:
410                     borderPoint2 = QgsPoint(float(tXLimit2+x0), tyBorder2+y0)
411 
412         cornersList = [QgsPoint(self.width, self.height),
413                 QgsPoint(self.width, 0), QgsPoint(0, 0), QgsPoint(0,
414                                                         self.height)]
415         p1Line = self.findLineNo(borderPoint1)
416         p2Line = self.findLineNo(borderPoint2)
417         if p2Line < p1Line:
418             p2Line += 4
419         if (p2Line == p1Line) and (angle > 180.0): # angle looped over
420             p2Line += 4
421 
422         cornersList.extend(cornersList)
423         vfCorners = cornersList[p1Line:p2Line]
424 
425         originPoint = QgsPoint(float(x0), float(y0))
426 
427         # construct the view field boundary coordinates
428         viewField = [originPoint, borderPoint1] + vfCorners + \
429                                         [borderPoint2, originPoint]
430 
431         # transform to map coordinates
432         viewFieldOnMap = map(self.mtp.toMapCoordinates,
433                 [p.x() for p in viewField], [p.y() for p in viewField])
434 
```

```
435         # construct the view field polygon
436         viewFieldPolyGeo = QgsGeometry.fromPolygon([viewFieldOnMap])
437 
438         return viewFieldPolyGeo
439 
440 
441     # get the center point of a map location
442     def getPointCenter(self, vPoint):
443         x = vPoint.x()
444         y = vPoint.y()
445         res = self.resolution
446         return QgsGeometry.fromPoint(QgsPoint(x+res/2.0, y-res/2.0))
447 
448 
449     # get the boundary rectange of a map cell
450     def getPointRectPoly(self, vPoint):
451         x = vPoint.x()
452         y = vPoint.y()
453         res = self.resolution
454         points = [QgsPoint(x, y), QgsPoint(x+res, y), QgsPoint(x+res, y-res),
455                                                             QgsPoint(x, y-res)]
456         return QgsGeometry.fromPolygon([points])
457 
458 
459     # depending on the polygon intersection
460     # find the weight of the target location
461     def findWeight(self, rPoint):
462         vPoint = self.mtp.toMapCoordinates(*rPoint)
463         #pointCenter = self.getPointCenter(vPoint)
464         #pointRectPoly = self.getPointRectPoly(vPoint)
465         for poly in self.polyList:
466             polyGeom = QgsGeometry(poly.geometry())
467             if polyGeom.intersects(QgsGeometry.fromPoint(vPoint)):
468                 weight = self.getWeightFromPoly(poly)
469                 return weight, True
470 
471         weight = 1
472         return weight, False
473 
474 
```

```
475     # read the weight value from the polygon
476     def getWeightFromPoly(self, poly):
477         index = self.vapInput.weightAttrIndex
478         atMap = poly.attributeMap()
479         (weight, result) = atMap[index].toDouble()
480         if not result:
481             return None
482         return weight
483 
484 
485     def showMark(self, rPoint, mark):
486         mark.setPosition(self.mtp.toMapCoordinates(*rPoint))
487         self.dialog.canvas.update()
488         qApp.processEvents(QEventLoop.AllEvents)
489 
490 
491     # generates path segments in order
492     def rasterSegmentGenerator(self, pathPoints):
493         pathSegmentsG = pairWise(pathPoints)
494 
495         for segmentStart, segmentEnd in pathSegmentsG:
496             rSegmentStartX, rSegmentStartY = \
497                         self.getRasterCoordsFromPoint(segmentStart)
498             rSegmentEndX, rSegmentEndY = \
499                         self.getRasterCoordsFromPoint(segmentEnd)
500             yield (rSegmentStartX, rSegmentStartY, rSegmentEndX, rSegmentEndY)
501 
502 
503     # generates polygons for weight zones
504     def polyGenerator(self, polyLayer):
505         #polyLayer.featureCount()
506         vProvider = polyLayer.dataProvider()
507         vProvider.select(vProvider.attributeIndexes(),
508                                                 QgsRectangle(), True, True)
509         currentPolygon = QgsFeature()
510         while vProvider.nextFeature(currentPolygon):
511             yield QgsFeature(currentPolygon)
512 
513 
514     # this method generates all of the map cells.
```

```
515     # can be used to calculate total viewsheds
516     def rasterTotalTargetGenerator(self):
517         rXStart, rYStart = 0, 0
518         rXEnd, rYEnd = self.width-1, self.height-1
519 
520         for y in xrange(rYStart, rYEnd+1):
521             for x in xrange(rXStart, rXEnd+1):
522                 yield (x, y)
523 
524 
525     # generate the target locations for the current observer
526     # targets are based on the radius, map boundaries,
527     # and the view field at that location
528     def rasterTargetGenerator(self, vObserver):
529         radius = self.vapInput.distLimit
530         res = int( (radius/self.resolution) / 2)
531         center = QgsGeometry.fromPoint(vObserver)
532         circle = center.buffer(radius, res)
533         if not self.fullView:
534             viewField = self.viewFieldPolyGeo
535             areaIntersection = circle.intersection(viewField)
536         else:
537             areaIntersection = circle
538 
539         # areaIntersection is the polygonal area
540         # of the intersection of view circle and view field
541         if areaIntersection is None:
542             return
543         bound = areaIntersection.boundingBox()
544 
545         xMin, yMax = bound.xMinimum(), bound.yMaximum()
546         xMax, yMin = bound.xMaximum(), bound.yMinimum()
547         rTopLeftFloat = self.mtp.transform(xMin, yMax)
548         rBottomRightFloat = self.mtp.transform(xMax, yMin)
549         rXStart, rYStart = int(round(rTopLeftFloat.x())),
550                                             int(round(rTopLeftFloat.y()))
551         rXEnd, rYEnd = int(round(rBottomRightFloat.x())),
552                                             int(round(rBottomRightFloat.y()))
553 
554         if rXStart < 0:
```

```
555             rXStart = 0
556         if rYStart < 0:
557             rYStart = 0
558         if rXEnd > self.width - 1:
559             rXEnd = self.width - 1
560         if rYEnd > self.height - 1:
561             rYEnd = self.height - 1
562 
563         for y in xrange(rYStart, rYEnd+1):
564             for x in xrange(rXStart, rXEnd+1):
565                 vPoint = self.mtp.toMapCoordinates(x, y)
566                 #pointRectPoly = self.getPointRectPoly(vPoint)
567                 #pointCenter = self.getPointCenter(vPoint)
568                 if areaIntersection.intersects(QgsGeometry.fromPoint(vPoint)):
569                     yield (x, y)
570 
571 
572     # get the points that construct a vector path
573     def findPathPoints(self, path):
574         pathPoints = path.geometry().asPolyline()
575         if self.vapInput.startPoint == "B":
576             pathPoints.reverse()
577         return pathPoints
578 
579 
580     def calculateStep(self, start, end, delta):
581         return (end-start)/float(delta)
582 
583 
584     def getRasterCoordsFromPoint(self, point):
585         rPointFloatF = self.mtp.transform(point)
586         rPointXF, rPointYF = rPointFloatF.x(), rPointFloatF.y()
587         return rPointXF, rPointYF
588 
589 
590     # this method calculates the visibility of a target
591     # from an observer location, both given in
592     # raster coordinates
593     def visibility(self, rObserver, rTarget):
594         (rObserverX, rObserverY) = rObserver
```

```
595         (rTargetX, rTargetY) = rTarget
596 
597         lineIter = bresenham(0, rObserverX, rObserverY, rTargetX, rTargetY)
598         try:
599             lineIter.next()
600         except StopIteration:
601             return True
602 
603         delta = max(abs(rTargetY - rObserverY), abs(rTargetX - rObserverX))
604 
605         observerGroundElev = self.rasterIO.getValue(rObserverX, rObserverY)
606         observerElev = observerGroundElev + self.currentObsHeight
607         targetElev = self.rasterIO.getValue(rTargetX, rTargetY)
608 
609         losElevStep = self.calculateStep(observerElev, targetElev, delta)
610         losElev = observerElev
611         for (x, y) in lineIter:
612             losElev += losElevStep
613             rValue = self.rasterIO.getValue(x, y)
614             if rValue > losElev:
615                 return False
616 
617         return True
```

A.5 ui_results.ui

This XML file is the Qt windowing toolkit file for the results window. It was produced graphically by the Qt Designer software.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>Results</class>
4   <widget class="QDialog" name="Results">
5    <property name="geometry">
6     <rect>
7      <x>0</x>
8      <y>0</y>
9      <width>678</width>
```

```
    <height>595</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_2">
   <item>
    <widget class="QTextEdit" name="editResults"/>
   </item>
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout">
     <item>
      <spacer name="horizontalSpacer">
       <property name="orientation">
        <enum>Qt::Horizontal</enum>
       </property>
       <property name="sizeHint" stdset="0">
        <size>
         <width>40</width>
         <height>20</height>
        </size>
       </property>
      </spacer>
     </item>
     <item>
      <widget class="QPushButton" name="buttonClose">
       <property name="text">
        <string>Close</string>
       </property>
      </widget>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections/>
</ui>
```

A.6 ui_input.ui

This XML file is the Qt windowing toolkit file for the main window. It was produced graphically by the Qt Designer software.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>431</width>
    <height>563</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <property name="autoFillBackground">
   <bool>true</bool>
  </property>
  <widget class="QGroupBox" name="grpSource">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>120</y>
     <width>411</width>
     <height>151</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <pointsize>10</pointsize>
     <weight>75</weight>
     <bold>true</bold>
    </font>
   </property>
```

```
35    <property name="autoFillBackground">
36     <bool>true</bool>
37    </property>
38    <property name="title">
39     <string>Source Selection</string>
40    </property>
41    <property name="flat">
42     <bool>false</bool>
43    </property>
44    <property name="checkable">
45     <bool>false</bool>
46    </property>
47    <widget class="QComboBox" name="cmbPathLayer">
48     <property name="geometry">
49      <rect>
50       <x>140</x>
51       <y>30</y>
52       <width>251</width>
53       <height>22</height>
54      </rect>
55     </property>
56     <property name="font">
57      <font>
58       <pointsize>8</pointsize>
59       <weight>50</weight>
60       <bold>false</bold>
61      </font>
62     </property>
63    </widget>
64    <widget class="QLabel" name="labPathLayer">
65     <property name="geometry">
66      <rect>
67       <x>20</x>
68       <y>30</y>
69       <width>91</width>
70       <height>21</height>
71      </rect>
72     </property>
73     <property name="font">
74      <font>
```

```
75        <pointsize>8</pointsize>
76        <weight>50</weight>
77        <bold>false</bold>
78       </font>
79      </property>
80      <property name="text">
81       <string>Path Layer:</string>
82      </property>
83     </widget>
84     <widget class="Line" name="line">
85      <property name="enabled">
86       <bool>true</bool>
87      </property>
88      <property name="geometry">
89       <rect>
90        <x>110</x>
91        <y>30</y>
92        <width>20</width>
93        <height>111</height>
94       </rect>
95      </property>
96      <property name="frameShadow">
97       <enum>QFrame::Sunken</enum>
98      </property>
99      <property name="orientation">
100      <enum>Qt::Vertical</enum>
101     </property>
102    </widget>
103    <widget class="QLabel" name="labObsHeight">
104     <property name="geometry">
105      <rect>
106       <x>20</x>
107       <y>60</y>
108       <width>91</width>
109       <height>21</height>
110      </rect>
111     </property>
112     <property name="font">
113      <font>
114       <pointsize>8</pointsize>
```

```
115       <weight>50</weight>
116       <bold>false</bold>
117      </font>
118     </property>
119     <property name="text">
120      <string>Observer Height:</string>
121     </property>
122    </widget>
123    <widget class="QLineEdit" name="editObsHeight">
124     <property name="geometry">
125      <rect>
126       <x>140</x>
127       <y>60</y>
128       <width>81</width>
129       <height>20</height>
130      </rect>
131     </property>
132     <property name="font">
133      <font>
134       <pointsize>8</pointsize>
135       <weight>50</weight>
136       <bold>false</bold>
137      </font>
138     </property>
139    </widget>
140    <widget class="QLabel" name="labStartPoint">
141     <property name="geometry">
142      <rect>
143       <x>20</x>
144       <y>90</y>
145       <width>91</width>
146       <height>21</height>
147      </rect>
148     </property>
149     <property name="font">
150      <font>
151       <pointsize>8</pointsize>
152       <weight>50</weight>
153       <bold>false</bold>
154      </font>
```

```
155      </property>
156      <property name="text">
157       <string>Starting Point:</string>
158      </property>
159     </widget>
160     <widget class="QLabel" name="labViewAngle">
161      <property name="geometry">
162       <rect>
163        <x>20</x>
164        <y>120</y>
165        <width>91</width>
166        <height>21</height>
167       </rect>
168      </property>
169      <property name="font">
170       <font>
171        <pointsize>8</pointsize>
172        <weight>50</weight>
173        <bold>false</bold>
174       </font>
175      </property>
176      <property name="text">
177       <string>View Angle:</string>
178      </property>
179     </widget>
180     <widget class="QLineEdit" name="editViewAngle">
181      <property name="geometry">
182       <rect>
183        <x>140</x>
184        <y>120</y>
185        <width>81</width>
186        <height>20</height>
187       </rect>
188      </property>
189      <property name="font">
190       <font>
191        <pointsize>8</pointsize>
192        <weight>50</weight>
193        <bold>false</bold>
194       </font>
```

```
195      </property>
196      <property name="alignment">
197       <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
198      </property>
199     </widget>
200     <widget class="QLabel" name="label">
201      <property name="geometry">
202       <rect>
203        <x>224</x>
204        <y>118</y>
205        <width>16</width>
206        <height>16</height>
207       </rect>
208      </property>
209      <property name="font">
210       <font>
211        <weight>50</weight>
212        <bold>false</bold>
213        <kerning>true</kerning>
214       </font>
215      </property>
216      <property name="text">
217       <string>o</string>
218      </property>
219     </widget>
220     <widget class="QFrame" name="frame">
221      <property name="geometry">
222       <rect>
223        <x>139</x>
224        <y>87</y>
225        <width>221</width>
226        <height>27</height>
227       </rect>
228      </property>
229      <property name="frameShape">
230       <enum>QFrame::Box</enum>
231      </property>
232      <property name="frameShadow">
233       <enum>QFrame::Sunken</enum>
234      </property>
```

```
    <property name="midLineWidth">
     <number>0</number>
    </property>
    <widget class="QPushButton" name="buttonAB">
     <property name="geometry">
      <rect>
       <x>106</x>
       <y>3</y>
       <width>108</width>
       <height>20</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <pointsize>8</pointsize>
       <weight>50</weight>
       <bold>false</bold>
      </font>
     </property>
     <property name="text">
      <string>Display A-B on map</string>
     </property>
     <property name="default">
      <bool>false</bool>
     </property>
     <property name="flat">
      <bool>false</bool>
     </property>
    </widget>
    <widget class="QRadioButton" name="rbStartPointA">
     <property name="geometry">
      <rect>
       <x>7</x>
       <y>4</y>
       <width>41</width>
       <height>17</height>
      </rect>
     </property>
     <property name="font">
      <font>
```

```
275         <weight>50</weight>
276         <bold>false</bold>
277        </font>
278       </property>
279       <property name="text">
280        <string>A</string>
281       </property>
282      </widget>
283      <widget class="QRadioButton" name="rbStartPointB">
284       <property name="geometry">
285        <rect>
286         <x>57</x>
287         <y>4</y>
288         <width>41</width>
289         <height>17</height>
290        </rect>
291       </property>
292       <property name="font">
293        <font>
294         <weight>50</weight>
295         <bold>false</bold>
296        </font>
297       </property>
298       <property name="text">
299        <string>B</string>
300       </property>
301      </widget>
302     </widget>
303     <widget class="QCheckBox" name="cb360">
304      <property name="geometry">
305       <rect>
306        <x>240</x>
307        <y>121</y>
308        <width>151</width>
309        <height>20</height>
310       </rect>
311      </property>
312      <property name="font">
313       <font>
314        <pointsize>8</pointsize>
```

```
315       <weight>50</weight>
316       <bold>false</bold>
317      </font>
318     </property>
319     <property name="text">
320      <string>No restriction (360o view)</string>
321     </property>
322    </widget>
323   </widget>
324   <widget class="QGroupBox" name="grpTarget">
325    <property name="geometry">
326     <rect>
327      <x>10</x>
328      <y>290</y>
329      <width>411</width>
330      <height>181</height>
331     </rect>
332    </property>
333    <property name="font">
334     <font>
335      <pointsize>10</pointsize>
336      <weight>75</weight>
337      <bold>true</bold>
338     </font>
339    </property>
340    <property name="autoFillBackground">
341     <bool>true</bool>
342    </property>
343    <property name="title">
344     <string>Target Selection</string>
345    </property>
346    <property name="flat">
347     <bool>false</bool>
348    </property>
349    <widget class="Line" name="line_2">
350     <property name="enabled">
351      <bool>true</bool>
352     </property>
353     <property name="geometry">
354      <rect>
```

```
355       <x>110</x>
356       <y>30</y>
357       <width>20</width>
358       <height>141</height>
359      </rect>
360     </property>
361     <property name="frameShadow">
362      <enum>QFrame::Sunken</enum>
363     </property>
364     <property name="orientation">
365      <enum>Qt::Vertical</enum>
366     </property>
367    </widget>
368    <widget class="QLabel" name="labDistLimit">
369     <property name="geometry">
370      <rect>
371       <x>20</x>
372       <y>30</y>
373       <width>91</width>
374       <height>21</height>
375      </rect>
376     </property>
377     <property name="font">
378      <font>
379       <pointsize>8</pointsize>
380       <weight>50</weight>
381       <bold>false</bold>
382      </font>
383     </property>
384     <property name="text">
385      <string>Distance Limit:</string>
386     </property>
387    </widget>
388    <widget class="QLineEdit" name="editDistLimit">
389     <property name="geometry">
390      <rect>
391       <x>140</x>
392       <y>30</y>
393       <width>81</width>
394       <height>20</height>
```

```
395       </rect>
396      </property>
397      <property name="font">
398       <font>
399        <pointsize>8</pointsize>
400        <weight>50</weight>
401        <bold>false</bold>
402       </font>
403      </property>
404     </widget>
405     <widget class="QLabel" name="labPolyLayer">
406      <property name="geometry">
407       <rect>
408        <x>20</x>
409        <y>60</y>
410        <width>91</width>
411        <height>21</height>
412       </rect>
413      </property>
414      <property name="font">
415       <font>
416        <pointsize>8</pointsize>
417        <weight>50</weight>
418        <bold>false</bold>
419       </font>
420      </property>
421      <property name="text">
422       <string>Polygon Layer:</string>
423      </property>
424     </widget>
425     <widget class="QComboBox" name="cmbPolyLayer">
426      <property name="geometry">
427       <rect>
428        <x>140</x>
429        <y>60</y>
430        <width>251</width>
431        <height>22</height>
432       </rect>
433      </property>
434      <property name="font">
```

```
435       <font>
436        <pointsize>8</pointsize>
437        <weight>50</weight>
438        <bold>false</bold>
439       </font>
440      </property>
441     </widget>
442     <widget class="QLabel" name="labWeightAttr">
443      <property name="geometry">
444       <rect>
445        <x>20</x>
446        <y>90</y>
447        <width>91</width>
448        <height>21</height>
449       </rect>
450      </property>
451      <property name="font">
452       <font>
453        <pointsize>8</pointsize>
454        <weight>50</weight>
455        <bold>false</bold>
456       </font>
457      </property>
458      <property name="text">
459       <string>Weight Attribute:</string>
460      </property>
461     </widget>
462     <widget class="QComboBox" name="cmbWeightAttr">
463      <property name="geometry">
464       <rect>
465        <x>140</x>
466        <y>90</y>
467        <width>251</width>
468        <height>22</height>
469       </rect>
470      </property>
471      <property name="font">
472       <font>
473        <pointsize>8</pointsize>
474        <weight>50</weight>
```

```
475       <bold>false</bold>
476      </font>
477     </property>
478    </widget>
479    <widget class="QLabel" name="labLimitPoly">
480     <property name="geometry">
481      <rect>
482       <x>20</x>
483       <y>120</y>
484       <width>91</width>
485       <height>21</height>
486      </rect>
487     </property>
488     <property name="font">
489      <font>
490       <pointsize>8</pointsize>
491       <weight>50</weight>
492       <bold>false</bold>
493      </font>
494     </property>
495     <property name="text">
496      <string>Limit to Polygons:</string>
497     </property>
498    </widget>
499    <widget class="QCheckBox" name="cbLimitPoly">
500     <property name="geometry">
501      <rect>
502       <x>140</x>
503       <y>120</y>
504       <width>70</width>
505       <height>21</height>
506      </rect>
507     </property>
508     <property name="font">
509      <font>
510       <pointsize>8</pointsize>
511       <weight>50</weight>
512       <bold>false</bold>
513      </font>
514     </property>
```

```
 <property name="text">
  <string/>
 </property>
</widget>
<widget class="QLabel" name="labDistFactor">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>150</y>
   <width>91</width>
   <height>21</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>8</pointsize>
   <weight>50</weight>
   <bold>false</bold>
  </font>
 </property>
 <property name="text">
  <string>Distance Factor:</string>
 </property>
</widget>
<widget class="QLineEdit" name="editDistFactor">
 <property name="geometry">
  <rect>
   <x>140</x>
   <y>150</y>
   <width>81</width>
   <height>20</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>8</pointsize>
   <weight>50</weight>
   <bold>false</bold>
  </font>
 </property>
```

```
555     </widget>
556     <widget class="QLabel" name="label_2">
557      <property name="geometry">
558       <rect>
559        <x>227</x>
560        <y>150</y>
561        <width>171</width>
562        <height>21</height>
563       </rect>
564      </property>
565      <property name="font">
566       <font>
567        <pointsize>8</pointsize>
568        <weight>50</weight>
569        <bold>false</bold>
570        <kerning>true</kerning>
571       </font>
572      </property>
573      <property name="text">
574       <string>map units ('0' to disable)</string>
575      </property>
576     </widget>
577    </widget>
578    <widget class="QLabel" name="iconVAP">
579     <property name="geometry">
580      <rect>
581       <x>10</x>
582       <y>10</y>
583       <width>41</width>
584       <height>41</height>
585      </rect>
586     </property>
587     <property name="layoutDirection">
588      <enum>Qt::LeftToRight</enum>
589     </property>
590     <property name="frameShape">
591      <enum>QFrame::Panel</enum>
592     </property>
593     <property name="frameShadow">
594      <enum>QFrame::Sunken</enum>
```

```
595     </property>
596     <property name="text">
597      <string/>
598     </property>
599     <property name="pixmap">
600      <pixmap resource="resources.qrc">:/VAP_icon.png</pixmap>
601     </property>
602     <property name="scaledContents">
603      <bool>false</bool>
604     </property>
605     <property name="margin">
606      <number>4</number>
607     </property>
608    </widget>
609    <widget class="QLabel" name="labelTitle">
610     <property name="geometry">
611      <rect>
612       <x>60</x>
613       <y>10</y>
614       <width>361</width>
615       <height>41</height>
616      </rect>
617     </property>
618     <property name="palette">
619      <palette>
620       <active>
621        <colorrole role="WindowText">
622         <brush brushstyle="SolidPattern">
623          <color alpha="255">
624           <red>85</red>
625           <green>85</green>
626           <blue>255</blue>
627          </color>
628         </brush>
629        </colorrole>
630        <colorrole role="Text">
631         <brush brushstyle="SolidPattern">
632          <color alpha="255">
633           <red>85</red>
634           <green>0</green>
```

```
635           <blue>127</blue>
636          </color>
637         </brush>
638        </colorrole>
639       </active>
640       <inactive>
641        <colorrole role="WindowText">
642         <brush brushstyle="SolidPattern">
643          <color alpha="255">
644           <red>85</red>
645           <green>85</green>
646           <blue>255</blue>
647          </color>
648         </brush>
649        </colorrole>
650        <colorrole role="Text">
651         <brush brushstyle="SolidPattern">
652          <color alpha="255">
653           <red>85</red>
654           <green>0</green>
655           <blue>127</blue>
656          </color>
657         </brush>
658        </colorrole>
659       </inactive>
660       <disabled>
661        <colorrole role="WindowText">
662         <brush brushstyle="SolidPattern">
663          <color alpha="255">
664           <red>120</red>
665           <green>120</green>
666           <blue>120</blue>
667          </color>
668         </brush>
669        </colorrole>
670        <colorrole role="Text">
671         <brush brushstyle="SolidPattern">
672          <color alpha="255">
673           <red>120</red>
674           <green>120</green>
```

```
675           <blue>120</blue>
676          </color>
677         </brush>
678        </colorrole>
679       </disabled>
680      </palette>
681     </property>
682     <property name="font">
683      <font>
684       <pointsize>10</pointsize>
685       <weight>75</weight>
686       <bold>true</bold>
687      </font>
688     </property>
689     <property name="frameShape">
690      <enum>QFrame::NoFrame</enum>
691     </property>
692     <property name="frameShadow">
693      <enum>QFrame::Plain</enum>
694     </property>
695     <property name="lineWidth">
696      <number>1</number>
697     </property>
698     <property name="text">
699      <string>Weighted Multi-Visibility Analysis on Directional Paths</string>
700     </property>
701    </widget>
702    <widget class="QPushButton" name="buttonAnalyze">
703     <property name="geometry">
704      <rect>
705       <x>346</x>
706       <y>531</y>
707       <width>75</width>
708       <height>23</height>
709      </rect>
710     </property>
711     <property name="sizePolicy">
712      <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
713       <horstretch>0</horstretch>
714       <verstretch>0</verstretch>
```

```
715      </sizepolicy>
716     </property>
717     <property name="text">
718      <string>Analyze</string>
719     </property>
720     <property name="flat">
721      <bool>false</bool>
722     </property>
723    </widget>
724    <widget class="QPushButton" name="buttonHelp">
725     <property name="geometry">
726      <rect>
727       <x>90</x>
728       <y>531</y>
729       <width>75</width>
730       <height>23</height>
731      </rect>
732     </property>
733     <property name="text">
734      <string>Help</string>
735     </property>
736    </widget>
737    <widget class="QComboBox" name="cmbElevationLayer">
738     <property name="geometry">
739      <rect>
740       <x>150</x>
741       <y>70</y>
742       <width>251</width>
743       <height>22</height>
744      </rect>
745     </property>
746     <property name="font">
747      <font>
748       <pointsize>8</pointsize>
749       <weight>50</weight>
750       <bold>false</bold>
751      </font>
752     </property>
753    </widget>
754    <widget class="QLabel" name="labElevationLayer">
```

```
755     <property name="geometry">
756      <rect>
757       <x>20</x>
758       <y>70</y>
759       <width>111</width>
760       <height>21</height>
761      </rect>
762     </property>
763     <property name="font">
764      <font>
765       <pointsize>10</pointsize>
766       <weight>75</weight>
767       <bold>true</bold>
768      </font>
769     </property>
770     <property name="text">
771      <string>Elevation Map:</string>
772     </property>
773    </widget>
774    <widget class="Line" name="divisor">
775     <property name="geometry">
776      <rect>
777       <x>10</x>
778       <y>90</y>
779       <width>411</width>
780       <height>31</height>
781      </rect>
782     </property>
783     <property name="orientation">
784      <enum>Qt::Horizontal</enum>
785     </property>
786    </widget>
787    <widget class="QPushButton" name="buttonAbout">
788     <property name="geometry">
789      <rect>
790       <x>10</x>
791       <y>531</y>
792       <width>75</width>
793       <height>23</height>
794      </rect>
```

```
 </property>
 <property name="text">
  <string>About</string>
 </property>
</widget>
<widget class="Line" name="divisor_2">
 <property name="geometry">
  <rect>
   <x>10</x>
   <y>517</y>
   <width>411</width>
   <height>16</height>
  </rect>
 </property>
 <property name="orientation">
  <enum>Qt::Horizontal</enum>
 </property>
</widget>
<widget class="QLabel" name="labDescText">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>490</y>
   <width>111</width>
   <height>21</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>10</pointsize>
   <weight>75</weight>
   <bold>true</bold>
  </font>
 </property>
 <property name="text">
  <string>Descriptive Text:</string>
 </property>
</widget>
<widget class="Line" name="divisor_3">
 <property name="geometry">
```

```
    <rect>
     <x>10</x>
     <y>466</y>
     <width>411</width>
     <height>31</height>
    </rect>
   </property>
   <property name="orientation">
    <enum>Qt::Horizontal</enum>
   </property>
  </widget>
  <widget class="QLineEdit" name="editDescText">
   <property name="geometry">
    <rect>
     <x>150</x>
     <y>489</y>
     <width>251</width>
     <height>21</height>
    </rect>
   </property>
  </widget>
 </widget>
 <resources>
  <include location="resources.qrc"/>
 </resources>
 <connections/>
</ui>
```

A.7 dialogs.py

This file deals with the dialog related operations of the plugin. Event handling for the GUI and input construction for the analysis are done here.

```
# -*- coding: utf-8 -*-

#/***************************************************************************
#
# The information about this plugin can be found in the 'vap.py' file...
```

```
6   #
7   #*****************************************************************************
8
9   import time
10
11  from PyQt4.QtCore import *
12  from PyQt4.QtGui import *
13  from qgis.core import *
14  from qgis.gui import *
15
16  import qgis.utils
17
18  from . import VERSION
19  from ui_input import Ui_Dialog
20  from ui_results import Ui_Results
21  import vap_io
22  import vap_analyze
23
24
25  class MainDialog(QDialog, Ui_Dialog):
26
27
28      def __init__(self, mainWindow, canvas):
29          flags = Qt.WindowTitleHint | Qt.WindowSystemMenuHint |\
30                                                  Qt.FramelessWindowHint
31          QDialog.__init__(self, mainWindow, flags)
32          self.setupUi(self)
33          self.canvas = canvas
34          self.mainWindow = mainWindow
35
36
37      def show(self):
38          QDialog.show(self)
39
40
41      def reject(self):
42          self.disconnectSignals()
43          self.clearAB()
44          QDialog.reject(self)
45
```

```
46
47      def accept(self):
48          self.disconnectSignals()
49          QDialog.accept(self)
50
51
52      def connectSignals(self):
53          QObject.connect(self.cmbPolyLayer, SIGNAL("currentIndexChanged(int)"),
54                                                      self.updateFieldsCombo)
55          QObject.connect(self.cmbPathLayer, SIGNAL("currentIndexChanged(int)"),
56                                                      self.clearAB)
57          QObject.connect(self.buttonAB, SIGNAL("clicked()"), self.redrawAB)
58          QObject.connect(self.buttonHelp, SIGNAL("clicked()"), self.showHelp)
59          QObject.connect(self.buttonAbout, SIGNAL("clicked()"), self.showAbout)
60          QObject.connect(self.buttonAnalyze, SIGNAL("clicked()"),
61                                                       self.analyzeClicked)
62          QObject.connect(self.cb360, SIGNAL("stateChanged()"),
63                                                       self.cb360Changed)
64
65
66      def disconnectSignals(self):
67          QObject.disconnect(self.cmbPolyLayer,
68              SIGNAL("currentIndexChanged(int)"), self.updateFieldsCombo)
69          QObject.disconnect(self.cmbPathLayer,
70              SIGNAL("currentIndexChanged(int)"), self.clearAB)
71          QObject.disconnect(self.buttonAB, SIGNAL("clicked()"), self.redrawAB)
72          QObject.disconnect(self.buttonHelp, SIGNAL("clicked()"), self.showHelp)
73          QObject.disconnect(self.buttonAbout, SIGNAL("clicked()"),
74                                                              self.showAbout)
75          QObject.disconnect(self.buttonAnalyze, SIGNAL("clicked()"),
76                                                            self.analyzeClicked)
77          QObject.disconnect(self.cb360, SIGNAL("stateChanged()"),
78                                                              self.cb360Changed)
79
80
81      def initDialog(self):
82          """Initializes all the components of the main dialog."""
83
84          self.disconnectSignals()
85          self.buttonAnalyze.setDefault(True)
```

```
86          self.cmbElevationLayer.clear()
87          demLayers = vap_io.getLayers(vap_io.layerFilterRaster)
88          for key, layer in demLayers:
89              self.cmbElevationLayer.addItem(layer.name(), key)
90
91          self.cmbPathLayer.clear()
92          pathLayers = vap_io.getLayers(vap_io.layerFilterLine)
93          for key, layer in pathLayers:
94              self.cmbPathLayer.addItem(layer.name(), key)
95
96          self.editObsHeight.setText("0")
97
98          self.rbStartPointA.setChecked(True)
99
100         self.editViewAngle.setText("120")
101         self.cb360.setChecked(False)
102
103         self.editDistLimit.setText("100")
104
105         self.cmbPolyLayer.clear()
106         polyLayers = vap_io.getLayers(vap_io.layerFilterPoly)
107         for key, layer in polyLayers:
108             self.cmbPolyLayer.addItem(layer.name(), key)
109
110         self.cbLimitPoly.setChecked(False)
111
112         self.editDistFactor.setText("0")
113
114         if self.cmbPolyLayer.count():
115             self.updateFieldsCombo(self.cmbPolyLayer.currentIndex())
116         else:
117             self.cmbWeightAttr.clear()
118
119         self.editDescText.setText("")
120
121         self.connectSignals()
122
123         self.abFound = False
124
125
```

```
126     def updateFieldsCombo(self, index):
127         """Updates the list of attributes for the weight attribute
128         combobox when a new polygon layer is selected."""
129 
130         curPolyLayer = self.getLayerFromCombo(self.cmbPolyLayer, index)
131 
132         if curPolyLayer is not None:
133             polyFields = curPolyLayer.dataProvider().fields().items()
134         else:
135             return
136 
137         self.cmbWeightAttr.clear()
138         for key, field in polyFields:
139             if field.type() == QVariant.Double:
140                 self.cmbWeightAttr.addItem(field.name() + " (" +
141                                            field.typeName() + ")", key)
142 
143 
144     def cb360Changed(self):
145         if self.cb360.isChecked():
146             self.editViewAngle.setDisabled(True)
147         else:
148             self.editViewAngle.setDisabled(False)
149 
150 
151     def redrawAB(self):
152         self.clearAB()
153         pathLayer = self.getLayerFromCombo(self.cmbPathLayer)
154         if not vap_io.drawAB(pathLayer, self):
155             self.showError(\
156               "Path error: can not find the selected path on the path layer.")
157         else:
158             self.abFound = True
159 
160 
161     def showHelp(self):
162         qgis.utils.showPluginHelp()
163 
164 
165     def showAbout(self):
```

```
166         aboutMessage = QString("Visibility Analysis for Paths " + VERSION +\
167                                "\nby Cagil Seker (cagils@gmail.com) @2010")
168         QMessageBox.about(self, "About VAP", aboutMessage)
169 
170 
171     def showError(self, errorMessage):
172         QMessageBox.critical(self, "Error", errorMessage)
173 
174 
175     def showInfo(self, infoMessage):
176         QMessageBox.information(self, "Information", infoMessage)
177 
178 
179     def setStatus(self, message):
180         self.mainWindow.statusBar().showMessage(message)
181 
182 
183     def clearAB(self):
184         try:
185             self.charMark1.delete()
186             self.charMark2.delete()
187             del self.charMark1
188             del self.charMark2
189         except AttributeError:
190             None
191 
192         self.abFound = False
193 
194 
195     def analyzeClicked(self):
196         if not self.inputCheckDialog():
197             return
198 
199         vapInput = self.constructInput()
200 
201         #self.accept()
202         #self.clearAB()
203 
204         vap_analyze.startAnalyzer(self, vapInput)
205         self.clearAB()
```

```
206
207
208     def showProgressBar(self, range):
209         self.progress = QProgressDialog("Visibility analysis in progress...",
210                                                         "Stop", 0, range, self)
211         self.progress.setWindowModality(Qt.NonModal);
212         self.progress.reset()
213         self.progress.show()
214
215
216     def updateProgressBar(self, value):
217         self.progress.setValue(value)
218
219
220     def inputCheckDialog(self):
221         cmbList = [self.cmbElevationLayer, \
222                 self.cmbPathLayer, self.cmbPolyLayer, self.cmbWeightAttr]
223         if [cmb.count() for cmb in cmbList].count(0):
224             self.showError("Please select suitable layers / fields"+\
225                                                         " for the plugin.")
226             return False
227
228         if self.getLayerFromCombo(self.cmbElevationLayer).bandCount() != 1:
229             self.showError("Please select an elevation layer that has "+\
230                                                             "only one band.")
231             return False
232
233         try:
234             obsHeight = float(self.editObsHeight.text())
235         except ValueError:
236             self.showError("Observer height must be a number.")
237             return False
238
239         if not self.cb360.isChecked():
240             try:
241                 viewAngle = float(self.editViewAngle.text())
242                 if viewAngle < 5 or viewAngle > 350:
243                     self.showError("Viewing angle must be between 5 and 355.")
244                     return False
245             except ValueError:
```

```
246                 self.showError("Viewing angle must be a number.")
247                 return False
248
249         try:
250             distLimit = float(self.editDistLimit.text())
251             if distLimit < 0:
252                 self.showError(\
253                     "Distance limit (radius) must be a positive number.")
254                 return False
255         except ValueError:
256             self.showError("Distance limit must be a number.")
257             return False
258
259         try:
260             distFactor = float(self.editDistFactor.text())
261             if distFactor < 0:
262                 self.showError(\
263             "Distance factor must be a positive multiplier for the weight.")
264                 return False
265         except ValueError:
266             self.showError("Distance factor must be a number.")
267             return False
268
269         if not self.abFound:
270             self.showError("Path error: please first display A-B endpoints.")
271             return False
272
273         return True
274
275
276     def constructInput(self):
277         input = vap_io.VapInput()
278
279         input.elevationLayer = self.getLayerFromCombo(self.cmbElevationLayer)
280         input.pathLayer = self.getLayerFromCombo(self.cmbPathLayer)
281         input.obsHeight = float(self.editObsHeight.text())
282         input.startPoint = "B" if self.rbStartPointB.isChecked() else "A"
283         if self.cb360.isChecked():
284             input.viewAngle = 360.0
285         else:
```

```
286             input.viewAngle = float(self.editViewAngle.text())
287         input.distLimit = float(self.editDistLimit.text())
288         input.polyLayer = self.getLayerFromCombo(self.cmbPolyLayer)
289         input.weightAttrIndex = int(self.getItemDataFromCombo(\
290                                                 self.cmbWeightAttr))
291         input.weightAttrName = str(self.getItemTextFromCombo(\
292                                                 self.cmbWeightAttr))
293         input.limitPoly = self.cbLimitPoly.isChecked()
294         input.distFactor = float(self.editDistFactor.text())
295         input.descText = str(self.editDescText.text().toLocal8Bit())
296 
297         return input
298 
299 
300     def getItemDataFromCombo(self, cmb, index=None):
301         if not index:
302             index = cmb.currentIndex()
303         return cmb.itemData(index).toString()
304 
305 
306     def getItemTextFromCombo(self, cmb, index=None):
307         if not index:
308             index = cmb.currentIndex()
309         return cmb.itemText(index)
310 
311 
312     def getLayerFromCombo(self, cmb, index=None):
313         layerId = self.getItemDataFromCombo(cmb, index)
314         return QgsMapLayerRegistry.instance().mapLayer(layerId)
315 
316 
317     def selectOutputFile(self):
318         """Opens the file save dialog for output file selection by the user"""
319 
320         # open the file save dialog for output file.
321         # currently only TIFF files are supported
322         outFileName = QFileDialog.getSaveFileName(self,
323             "Save raster file", ".", "GTiff (*.tif *.tiff *.TIF *.TIFF)")
324 
325         if outFileName.isEmpty():
```

```
326             return
327
328         if not outFileName.toLower().endsWith(".tiff"):
329             if not outFileName.toLower().endsWith(".tif"):
330                 outFileName += ".tiff"
331
332         # display the selected file in the field
333         return outFileName
334
335
336 class ResultsDialog(QDialog, Ui_Results):
337
338
339     def __init__(self, mainWindow):
340         flags = Qt.WindowTitleHint | Qt.WindowSystemMenuHint
341         QDialog.__init__(self, mainWindow, flags)
342         self.setupUi(self)
343         self.mainWindow = mainWindow
344
345
346     def show(self):
347         QDialog.show(self)
348
349
350     def reject(self):
351         self.disconnectSignals()
352         QDialog.reject(self)
353
354
355     def accept(self):
356         self.disconnectSignals()
357         QDialog.accept(self)
358
359
360     def connectSignals(self):
361         QObject.connect(self.buttonClose, SIGNAL("clicked()"), self.accept)
362
363
364     def disconnectSignals(self):
365         QObject.disconnect(self.buttonClose, SIGNAL("clicked()"), self.accept)
```

```
366
367
368     def initDialog(self):
369         self.disconnectSignals()
370         self.editResults.setText("")
371         self.connectSignals()
```