2014

Comparison of Ground-to-Air Visibility Analysis Methods

Erica Pfister-Altschul
University of South Carolina - Columbia

COMPARISON OF GROUND-TO-AIR VISIBILITY ANALYSIS METHODS

by

Erica Pfister-Altschul

Bachelor of Science
Massachusetts Institute of Technology, 2000

Master of Engineering
University of South Carolina, 2011

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in

Geography

College of Arts and Sciences

University of South Carolina

2014

Accepted by:

Michael Hodgson, Director of Thesis

Sarah Battersby, Reader

Chris Emrich, Reader

Lacy Ford, Vice Provost and Dean of Graduate Studies

DEDICATION

For Yingvyn, Bebil, Biyal, and Qinigle, vos gelernt vee tzu zen zich.

ACKNOWLEDGEMENTS

This research was supported in part by the Department of Homeland Security grant for the RESPT project. More importantly, Dr. Michael Hodgson, Dr. Sarah Battersby, and Dr. Chris Emrich provided me with thoughtful advice and insightful comments as my thesis committee.

But most of all, this work would not have been possible without the support and enthusiasm of my family and friends. Appreciating their unwavering belief in me was the most valuable part of this process. Brett, Lillian, Reuven, Benji, and Carol: thanks for helping me learn what I wanted to do when I grow up.

ABSTRACT

When a disaster occurs, remotely sensed imagery is critical for emergency responders. Aircraft collect digital images of damaged areas to assist with damage assessment and response planning. Such airborne imagery can be transmitted directly from the plane to ground antennae and internet-connected dispersal, allowing for faster acquisition of data. However, air-to-ground transmission of images requires near-constant visibility between the aircraft transmitter and ground station antenna. This research uses GIS-based models to identify the ground station locations that can reliably receive data from aircraft, using a variety of visibility analysis methods and a comparison of their performance. A custom algorithm is demonstrated to perform significantly faster than commercially available software tools.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Rapid emergency response is critical after a disaster occurs. Remotely sensed imagery can be a great asset in such situations. Quick action is facilitated by having imagery to help assess impacted areas. Once the extent of the damage is known, the appropriate preparations can be made to ensure that the right kind of aid is sent to where it is most needed.

There are multiple remote sensing technology options available for disaster response imagery. Satellites provide useful coverage, particularly in remote regions, but they also have limitations. Their orbital planes are fixed, and thus imagery can only be captured at predetermined overpasses (for example. every three days in the late morning). Weather conditions or time of day may obscure sensors. In contrast, aircraft can be positioned wherever needed, fly below heavy cloud cover, fly at all hours of the day, and capture oblique imagery. Programs such as the Civil Air Patrol, with over 450 planes, provide federally-directed coverage to capture imagery of areas affected by disasters, facilitating rapid impact assessment.

Image delivery time can be improved even more by transmitting airborne imagery directly from a plane to a ground antenna. Rather than waiting for the flight to land, image analysis can begin almost immediately, and response actions can be implemented more quickly as a result. Microwave transmission of imagery data for disaster response from the air is a relatively new application, and is still being explored.

Successful air-to-ground transmission of data has some basic requirements. The ground antenna needs to have power and a hard-wired internet connection. Most importantly, since the transmission is via microwave, there must be near-constant visibility between the aircraft and the antenna to reliably receive all the data. Therefore, the choice of location for the ground antenna should be considered beforehand to ensure visibility of the entire flight path.

The REmote Sensing Planning Tool (RESPT) is a set of web-based decision support tools that provide guidance on acquiring and analyzing remotely sensed imagery for diverse applications. One component of RESPT is the Ground Station Siting Model (GSSM), a tool that allows an emergency response team to find optimal locations to position a ground receiving antenna for image downlinks. It requires three inputs: candidate ground antenna sites, the flight

path of a remote sensing aircraft, and an elevation surface model of the region. From this data, the GSSM can provide a ranking of candidate site suitability.

There are two basic phases of evaluating ground station sites: screening and visibility analysis. During the planning phase of the emergency response cycle, before any disaster has actually occurred, screening of potential locations can take place to determine if they meet basic suitability criteria. Emergency responders will need to gain access to the antenna site in the event of a disaster. Suitable locations should be owned by a government agency, or usage agreements must be established in advance to ensure that responders will be allowed into a building to set up a ground antenna. The site needs to have a reliable supply of electricity; power outages (a common side effect of disasters) should be planned for and may be circumvented by providing an on-site generator. Finally, the antenna needs to have a hard-wired internet connection – a wireless (Wi-Fi) connection is not adequate.

In addition, the antenna needs to be able to "see" the majority of the airplane flight path. For an initial rough estimate, it can be assumed that locations at higher elevations are probably able to see a larger proportion of the surrounding sky than sites that are at lower elevations. Also, sites surrounded by vegetation and other buildings would have reduced sky visibility. However, the question of visibility cannot be fully answered until a flight path has actually

been selected. The visibility analysis is run on locations which meet the preliminary screening criteria and enables the final selection of the most appropriate antenna site.

The primary goal of this research was to develop a GIS model that could measure visibility between candidate ground antenna sites and a moving aircraft. No known ground station modeling solution existed for determining appropriate ground station locations for receiving airborne transmissions. Therefore, this research has explored variety of modeling approaches that can identify ground station locations capable of reliably receiving data from aircraft.

A visibility analysis can be addressed through a variety of modeling approaches. Three metrics -- efficiency, sensitivity, and accuracy -- were used to compare candidate algorithms. A visibility analysis could be conducted either prior to an actual emergency during planning stages, or after an event has occurred. The GSSM tool was assumed to be an emergency response tool, and therefore determining the best location for a ground antenna must be done in a short period of time.

This research will be useful within the context of disaster response efforts. More broadly, it can also be applied to any new visibility modeling problems that involve an airborne target. Air-to-ground visibility has not been researched in the GIScience literature except for fixed-orbit satellites or solar planes.

In time-sensitive disaster response events, response time is critical and the processing time of visibility analysis algorithms is the most significant factor as to whether a model is suitable. Many modeling approaches may be accurate enough to answer the question. In order to facilitate rapid disaster response, a threshold of one hour was assumed to be the maximum acceptable processing time. If a visibility analysis consistently takes longer than sixty minutes to select the best ground antenna site, then it may not provide any time savings over driving to the local airport to physically retrieve the imagery data and is therefore not a suitable approach.

This thesis compares the performance of three alternative visibility analysis methods within an embedded GIS environment and a loosely-coupled GIS environment. The potential solutions each underwent validation and sensitivity analyses, providing information about the efficiency and accuracy of alternative approaches. Simultaneously, some innovative work was done with visibility modeling. Typical assumptions about viewshed models have been inverted to work as a ground-to-air visibility measurement rather than ground-to-ground. Also, a new open-source algorithm for rapid line-of-sight has been written.

CHAPTER 2

LITERATURE REVIEW

Visibility analysis has a wide range of possible applications, and GIS has been used to answer visibility questions for decades. One fundamental requirement that is shared by all methods of visibility analysis is knowledge of local natural or man-made terrain features. Visibility is analyzed along straight "lines of sight" by evaluating whether they are blocked by terrain features (see Figure 2.1). Intervisibility is always assumed: if the observer can see a target object, then the target object is able to see the observer.

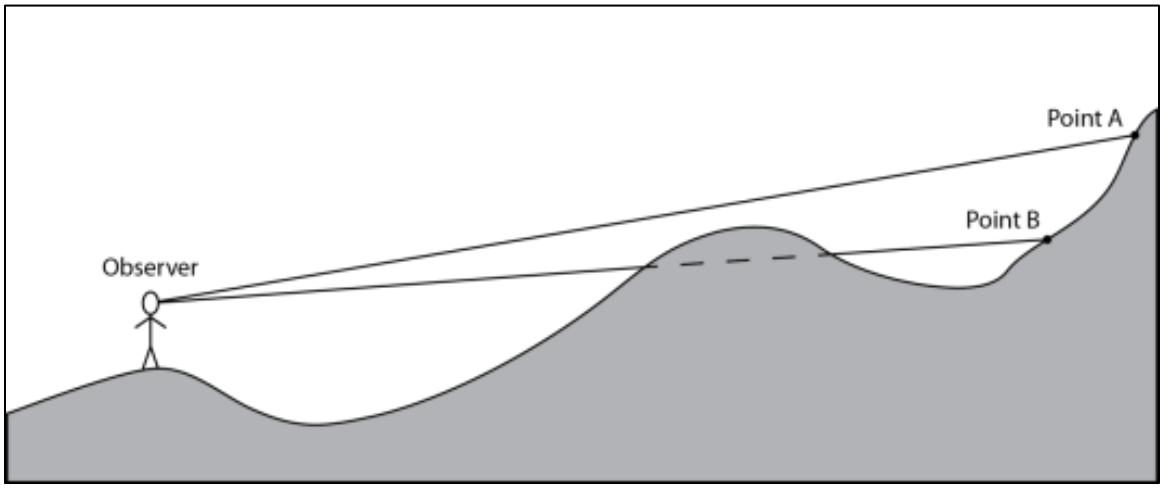

Figure 2.1: A schematic diagram of the visibility analysis process. Point A is visible to the observer, while Point B is not.

Predicting visibility was one of the oldest motivations for research into terrain mapping. Military planning requires visibility analyses when considering such activities as moving troops, performing reconnaissance, or assaulting a target. Depending on the situation, the goal may be either to see as much enemy activity or territory as possible, or be seen as little as possible by the enemy (Bruzese 1989). The military's need for accurate intelligence provided the impetus for the development of many geographic and spatial analysis techniques that are now widely used in civilian research and planning. Terrain mapping and analysis were critically important, and contributed to a wide range of modern GIS procedures (O'Sullivan 1983).

In addition to the military, the United States Department of Agriculture's Forest Service contributed significantly to the development of early visibility research. Viewshed analyses were used both to delineate scenic views from given vantage points (with the goal of protecting the surrounding landscapes), and to find optimal sites for fire towers that could observe as much of the surrounding forest as possible from a single vantage point. The Forest Service worked with visibility analyses long before computer analyses were possible (Show et al. 1937), and their expertise and interest led to some of the earliest computer algorithms for visibility analysis and terrain modeling (Amidon and Elsner 1968; Travis et al. 1975; Mees 1976).

As GIS became increasingly available, the number of disciplines using visibility algorithms significantly increased. Archaeologists used visibility analyses to understand historical landscapes, exploring intervisibility of sites of interest (Wheatley 1995) or determining visibility of a megalithic site from surrounding population centers (Ruggles, Medyckyj-Scott, and Gruffydd 1993). Wildlife population counts have used visibility analyses to calculate visible area, improving the accuracy of estimates of the spatial area in which given species were seen and counted (Maichak and Schuler 2004). Land developers and urban planners regularly consider view and visibility as part of the overall experience of a place, using the visibility of surrounding features to evaluate the nearby landscape and its aesthetic experience (Lynch 1976). The growing interest in preventing terrorism and crime means that interest in visibility is moving into fields such as surveillance monitoring, where it is used to optimize video camera coverage (Murray et al. 2007; Kim, Murray, and Xiao 2008) or estimate possible sniper positions (VanHorn and Mosurinjohn 2010).

A derived visibility surface can be used as an input to other GIS analyses as well. The military application of minimizing visibility evolved into a "least visible path" (LVP) analysis, which is essentially a least-cost path (LCP) based on terrain visibility analysis. LCP analysis requires a "friction surface" or "cost surface," which is a raster that describes the cost to move through each cell in

terms of movement difficulty, speed limits, dollar costs, or other measured constraints on travel. An LVP approach uses visibility as the cost factor and finds the path over the surface which has the least visual exposure (Lu et al. 2008). In addition to military uses, LVP can be used for civilian activities such as routing power lines to minimize their visibility (Bagli, Geneletti, and Orsi 2011). The LVP can be inverted to instead find the most visible path, also called a "scenic path" analysis. This can be used, for example, to plan a hiking trail that provides maximum visibility of the surrounding landscape (Lee and Stucky 1998).

The sample of studies described in the preceding paragraphs is not intended to be comprehensive. Rather, it provides a picture of the scope of research questions and the number of researchers using well-established methods of visibility analysis in new applications. Even as uses for visibility analysis increase, they can still be broadly categorized into three basic analysis types: sky view, isovist, and viewshed. The best choice for a visibility analysis question depends on many factors, including the relative positions of observer and target objects, the complexity of nearby terrain or surface elevation, and the scope of analysis or area of interest.

2.1 CATEGORIES OF VISIBILITY ANALYSIS

2.1.1 SKY VIEW

A *sky view*, also known as a sky visibility viewshed, is "the angular distribution of sky visibility versus obstruction" (Fu and Rich 2000). Simply stated, a sky view describes the visible portion of the hemisphere of the sky. It is derived by first calculating the horizon angles from the observer's vantage point (usually at sixteen evenly-spaced points around the horizon to simplify calculation). The angles are then converted into a hemispherical coordinate system, and are then used to derive which parts of the sky are visible and which are obstructed (see Figure 2.2).

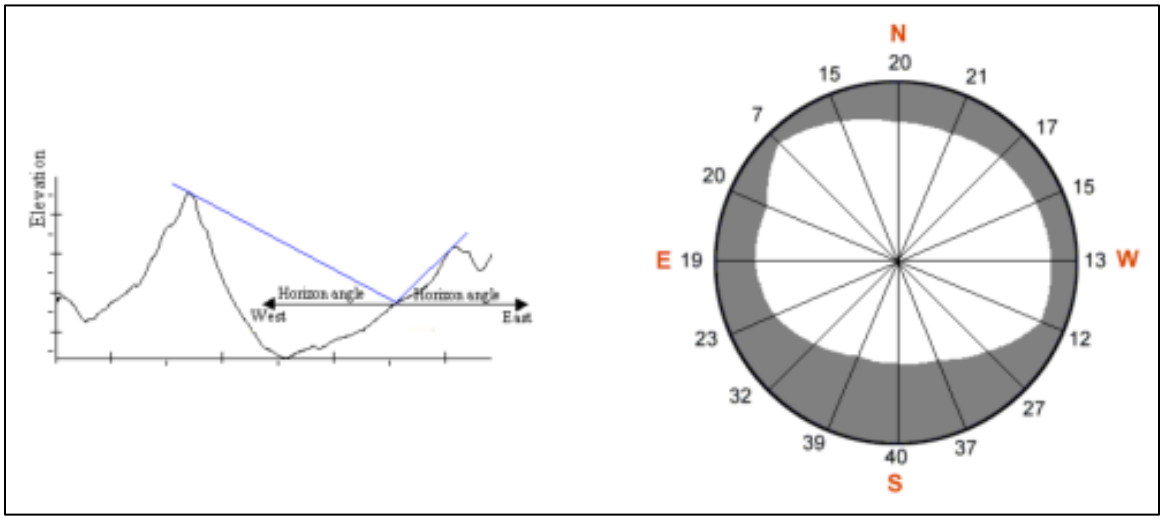

Figure 2.2: In a sky view, horizon angles are calculated for each direction (left), and the resulting output (right) shows how much of the sky is visible and how much is obscured (Fu and Rich 2000). The use of sixteen directions is somewhat arbitrary: a sample of the infinite directions available simplifies the analysis.

Multiple research questions can use sky view analysis. It is most typically used for solar radiation models (Dozier and Frew 1990), tracking the position of the sun throughout the day, and using that data (i.e., whether the sun is directly visible or not) to calculate direct versus diffuse radiation percentages. Visibility of any objects in space, such as GPS satellites (Beesley 2002) can be measured. Applications have even been found in archaeology, to analyze what prehistoric astronomers would have been able to see at megalithic sites (Ruggles, Medyckyj-Scott, and Gruffydd 1993).

2.1.2 ISOVIST

The definition of the term *isovist* was first formalized in 1979: "the set of all points visible from a given vantage point in space and with respect to an environment" (Benedikt 1979). The term is most commonly used in the context of theories of perception, space, and visual environments relating to architecture, urban planning, landscape design, and other fields concerned with human perception and man-made spaces. In addition to evaluating the physical measurements of a visible space, an isovist can also include descriptions of the visual experience, detailing what objects and spaces would be seen by the observer.

An isovist is a two-dimensional polygon that represents the space visible from a given vantage point, derived using a horizontal plane at eye level to

identify visual obstacles and thereby delineate visible space around the observer (see Figure 2.3). The two-dimensional nature of an isovist is more of an artifact than a fundamental requirement. Studies have explored the possibility of expanding the concept to three-dimensional analysis (Benedikt 1979; Morello and Ratti 2009), which would more accurately represent the actual human experience from a given vantage point. The restriction to two dimensions is due to historical limitations on computer processing speed and storage, which prevented more complex analyses from being generally feasible.

Isovists can be performed either in an internal space, evaluating visibility within a set of rooms, or an external space, evaluating visibility in an urban setting. Since the two-dimensional isovist does not account for verticality, instead stopping the "visible" boundary at the first eye-level obstacle, it cannot identify whether an object further from the observer would be visible or not. For example, a small tree would not completely block the view of a large building, even though the isovist analysis of that space would indicate the building was outside the visible space.

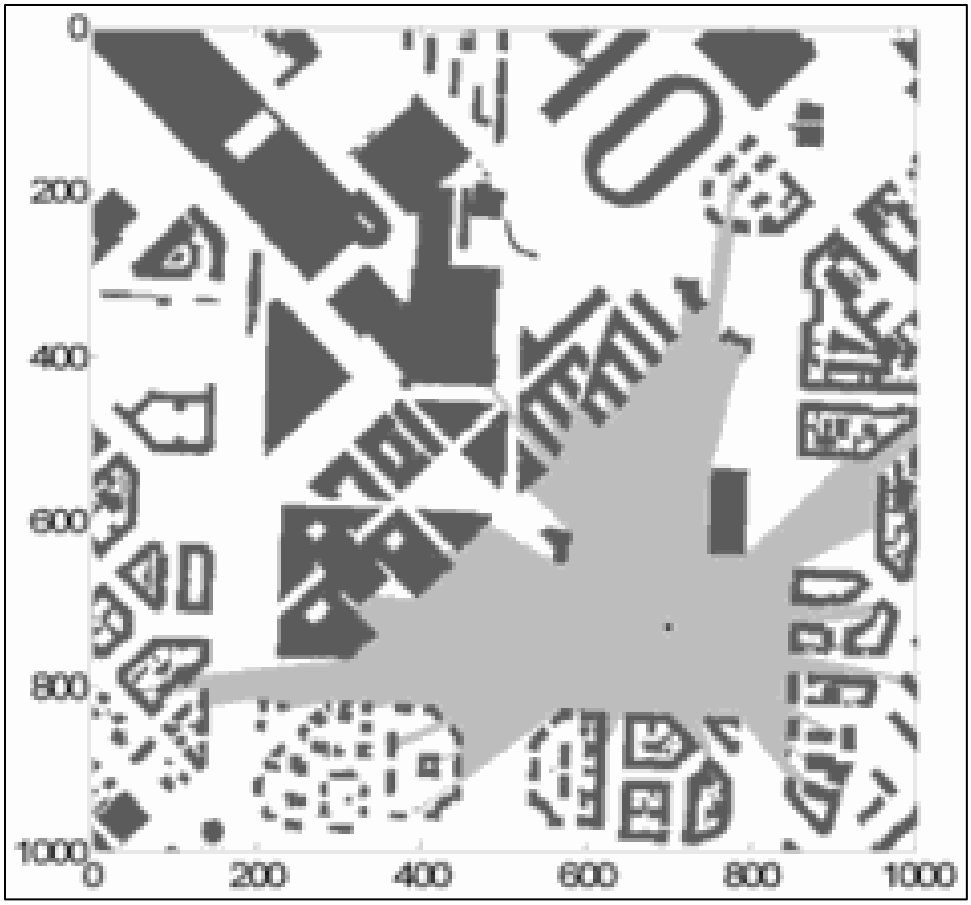

Figure 2.3: An example of an isovist in an urban setting. White is flat ground; dark grey is buildings; and light grey is the visible area from the vantage point (Morello and Ratti 2009).

2.1.3 VIEWSHED

The third major category of visibility analysis, *viewshed*, is arguably the most commonly used visibility analysis in geography and many other disciplines. Inspired by the term "watershed," which describes an area of land where all the water drains to a given point, "viewshed" describes an area of land which can be seen from a given point. Unlike a sky view analysis, a viewshed considers objects on (or very near) the ground. The most basic type, known as a

binary viewshed, measures the visibility of every point in the surrounding terrain and codes it as either "visible" or "not visible" (see Figure 2.4). Unlike an isovist, which only considers the immediately adjacent polygon of visibility, a viewshed looks all the way to the furthest horizon and can identify non-contiguous visible areas. Visibility is not limited to a contiguous spatial area, but is evaluated individually for each point (e.g., each cell in a regular tessellation) in the entire analysis region. This makes viewshed analysis more suitable for hilly or mountainous natural landscapes, since it can explore the full visibility of an area of interest.

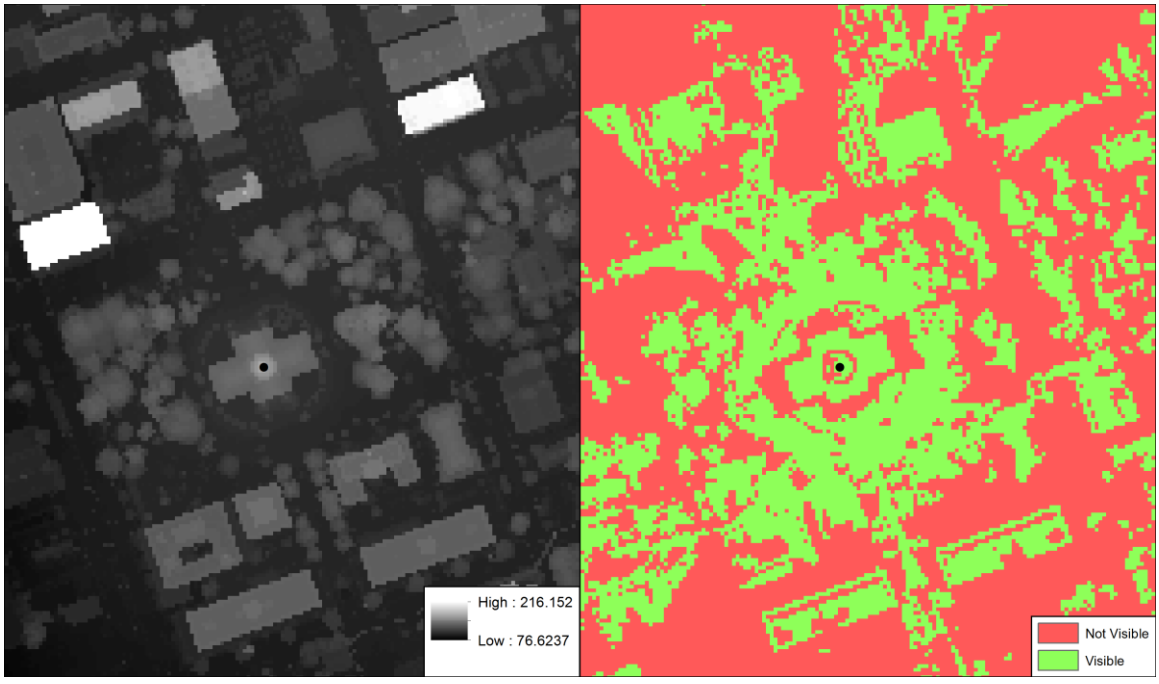

Figure 2.4: An example of the binary viewshed of an observer point on top of the State House in Columbia, SC.

Viewsheds also have more flexibility than other visibility analyses. Since the visibility of all the surrounding terrain is measured during the analysis, viewsheds can simultaneously evaluate visibility of multiple target objects (for example, ten potential wind farm sites can be coded as "visible" or "not visible" depending on the visibility of the terrain at their proposed locations). Multiple binary viewsheds can be added together to create a cumulative viewshed, allowing the analysis to consider multiple *observers* in addition to multiple targets (a characteristic that is not found in sky view or isovists). Viewshed algorithms also typically account for the height above the ground, or "offset," of the observer and/or the target objects in visibility calculation – although, despite relatively early development of such features (Mees 1978), not all viewshed analyses included offset options for many years (Fisher 1996). Finally, the designation of whether a point is "observer" or "target" is more flexible than in sky view or isovist, giving a GIS model more flexibility in its parameters and analysis.

2.2 SOURCES OF INACCURACY

While viewshed analysis may be the most widely used type of visibility analysis, its popularity is not because it is more accurate than alternative approaches. In fact, a comparison of field-surveyed and GIS-predicted viewsheds found that the average level of agreement was "only slightly higher than 50

percent" at best (Maloy and Dean 2001). There are multiple factors that may contribute to a lack of accuracy in visibility analysis.

Three primary sets of challenges were identified in 1990, when viewsheds were beginning to increase in popularity and application (Felleman and Griffin 1990). First, any inaccuracies in the elevation surface will propagate to any subsequent analysis (such as a viewshed). Such errors can come from a multitude of possible sources, and cannot always be prevented. Second, the "'black box' nature of … proprietary 'user friendly' GIS algorithms" provides no information about how the viewshed is calculated. Not only does this make pinpointing errors difficult, since the underlying assumptions of the algorithm are invisible, but different algorithms can produce different results. Third, an inexperienced user may inadvertently introduce error to the analysis by choosing inadequately detailed or up-to-date data. User error also occurs in reporting, as many studies tend to ignore or underestimate the errors in viewsheds. Therefore, despite the widespread availability of viewshed algorithms, results should be treated with some caution and their uncertainties acknowledged.

These challenges have been discussed specifically in the context of viewsheds, rather than visibility analyses in general. In part, this is because of the popularity of viewsheds, which are much more commonly used than sky views or isovists. Relatively few studies specifically discuss the accuracy of

non-viewshed visibility. However, the fundamental principles of both sky view and isovist analyses rely on an elevation model and lines of sight. We can therefore infer that the challenges are not specific to viewsheds.

2.2.1 THE ELEVATION SURFACE

All computer-based visibility analyses require a digital model of the terrain. There are two terms, somewhat interchangeable, that may be used in this context: digital elevation model (DEM), or digital surface model (DSM). Broadly speaking, a DSM incorporates surface features such as vegetation or buildings in addition to the base terrain data in a DEM. Either can be used in a visibility analysis, depending on the required level of accuracy and the goal of the analysis.

A DEM provides the underlying information for the visibility analysis. Without knowing the height and location of nearby features, it is impossible to determine which might cause obstructions. Any errors in the DEM will propagate to modeled products, including a viewshed or other visibility analysis, and therefore understanding the DEM error is an important first step in controlling error (Fisher 1991). Since DEMs are used in a wide variety of geographical modeling, there has been substantial research on sources of error in DEMs and how to estimate or ameliorate them.

The process of generating an elevation surface involves three basic tasks: gathering a sample of height measurements, creating a surface model from the data, and correcting errors or artifacts in the resulting digital model (Hengl and Evans 2009). Each step involves a number of choices about the most appropriate method. For example: when measuring the terrain height, what technology should be used and how closely spaced should points be sampled? The decisions at each stage in the process will have impacts on the overall accuracy of the model.

Sources of DEM error fall into categories that roughly correspond to the phases of DEM creation (Fisher and Tate 2006). Data-based error stems from variations in the accuracy or density of measured source data, which are dependent on the method of data generation. Method-based errors arise when creating the surface model. The processing and interpolation used to turn source data into a continuous elevation surface can introduce inaccuracies. Also, the characteristics of the terrain surface being modeled and its representation in a DEM can affect how well accurately the terrain is approximated.

The development and increasing use of new technologies for deriving DEMs such as radar, light detection and ranging (LiDAR), and digital photogrammetry has required some re-evaluation of the impacts of data collection and processing (Fisher and Tate 2006). While these active-sensing

technologies may reduce data-based error, the amount of processing required introduces new method-based considerations. For example, an elevation "point" cloud collected with LiDAR must be filtered to find the desired returns, determining whether a collected return is from bare ground, vegetation, buildings, or other surface constructions. Choices made during processing create additional opportunities for error. It is therefore not appropriate to assume that a fine-resolution model is more accurate than a coarse-resolution model. Active sensors are able to sample points at much higher density than other methods, potentially leading to a simple "newer is better" conclusion; however, the high resolution DEM may have greater uncertainty if its attribute values are less well understood (Wilson 2012). For example, if LiDAR points are provided with no metadata about the sensor type, time of flight, or other useful information, the post-processing of the point cloud data requires guesswork and can introduce new inaccuracies.

The most important consideration is that errors from source data or processing cannot always be eliminated. Therefore, the analysis of a DEM "must be cognizant of these errors" and take into account how inaccuracy may affect the subsequent analysis and eventual interpretation of the significance of their results (Wilson 2012).

2.2.2 RESOLUTION

A related source of error is the spatial resolution of the elevation surface. Choosing the "best" DEM resolution for any GIS analysis is a matter of balancing the need for accuracy and detail against processing speed and storage requirements. The tradeoff is not necessarily direct or predictable: while fine resolution DEMs generally represent terrain more accurately, the improvement is less significant in a relatively smooth landscape (Hengl 2006). In general, however, a reduction in resolution will generalize, mask, or eliminate important surface features. For example, building footprints may have significantly different shapes, and narrow obstacles such as trees or antenna towers may be hidden (see Figure 2.5). Visibility algorithms are left with a relatively inaccurate representation of reality, and the output results will reflect this.

Error resulting from coarser spatial resolution is separate from measurement error, since it is introduced after measurements of the surface are completed. It may arise either from processing choices when the surface is being generated, or from the user's choice of what available elevation surface to use in a visibility analysis. Considering spatial resolution separately from measurement recognizes its additional significance in time-sensitive visibility analyses, when resolution becomes a concern because of its influence on processing time.

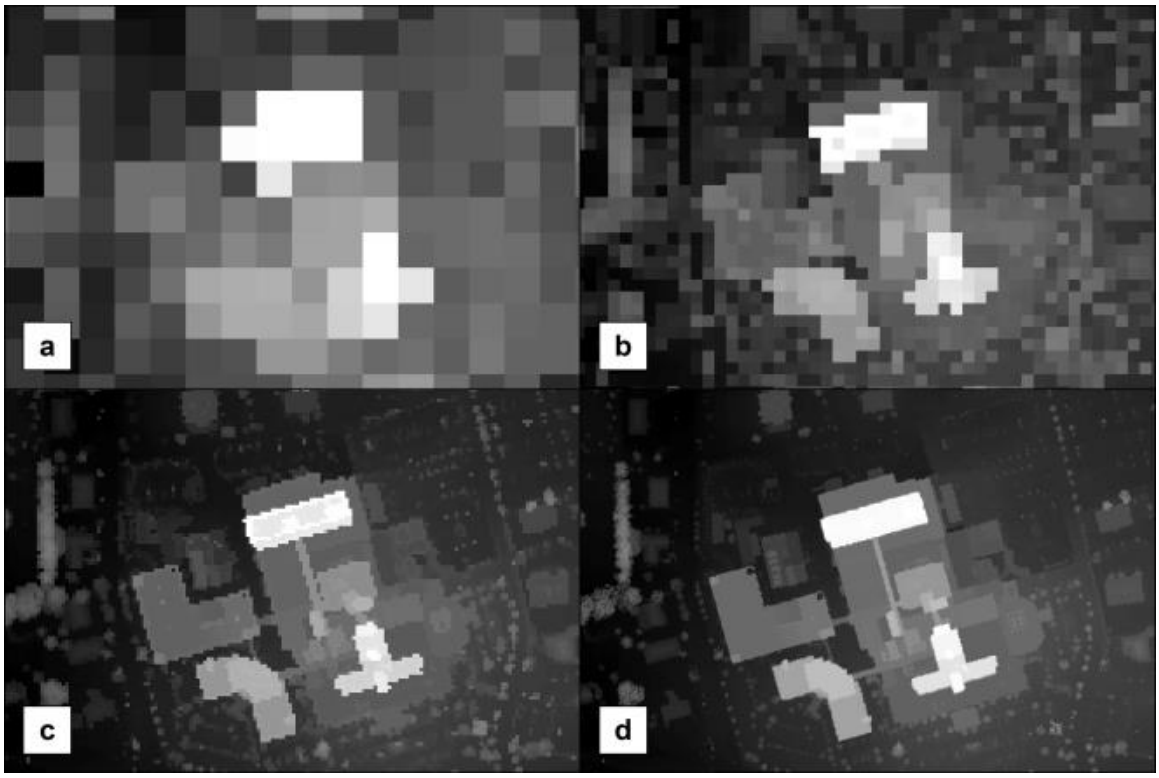

Figure 2.5: The Lexington Medical Center in West Columbia, SC in LiDAR-derived DSMs, at (a) 30 meter, (b) 10 meter, (c) 3 meter, and (d) 1 meter resolution.

2.2.3 THE ALGORITHM

Another possible source of error comes from the visibility algorithm itself. While there are three basic types of analysis, each one can be accomplished with a number of different algorithms. Any approach involves various assumptions and simplifications which will affect the results. In the case of algorithms which are proprietary parts of commercial software, the cause of inaccuracies must be guessed at, making quantification nearly impossible.

A good analogy for the influence of algorithm choices is vector-raster conversion. When converting data in a GIS between raster and vector formats,

the choice of grid cell size, grid position, or cell classification method can affect the shape and size of features (Congalton 1997). A viewshed algorithm is working with raster elevation data and vector line-of-sight data to calculate visibility, and the processing choices made by the original authors of an algorithm will not necessarily be consistent between software packages.

The effect of different algorithms has been demonstrated empirically by running a viewshed analysis on the exact same study area using four different software packages (IDRISI, MAPII, PMAP, and ARC-INFO). The resulting visible areas varied significantly due to "different, typically undocumented, simplifying assumptions" that the programmers of the various algorithms used (Felleman and Griffin 1990). Differences were unpredictable: some portions of the study area were more likely to have wide variation in visibility boundaries than others. Since each algorithm was a proprietary part of commercial software, the researchers were unable to further analyze the causes of error.

2.2.4 FUZZY VIEWSHEDS

An additional type of inaccuracy can result from only considering landscape topography in the calculation of line-of-sight geometry. Environmental interference, such as from fog, sunlight glare, or haze, can result in reduced visibility even when a target location should be visible according to the topography. This measurement of how distinct an observed target location

map appear has been termed a "fuzzy viewshed," and is distinct from an "uncertainty viewshed," which measures probable error in visibility assessment calculated from known DEM error (Fisher 1994). Atmospheric conditions that scatter or absorb visible light have different effects on electromagnetic energy at different wavelengths, however. Interference with the visual path from atmospheric haze or solar glare would not cause as significant a problem for microwave transmission, and therefore a fuzzy approach is not necessary in this context.

2.2.5 USER ERROR

The widespread availability of a viewshed tool in both commercial and open-source GIS software has led to a great deal of popularity, as can been seen from the wide range of research questions which use it. Widespread adoption and ease of use does not mean that results are automatically authoritative, but it can lead to a misperception of the tool's accuracy, implying a level of confidence that may not be justified.

In the field of landscape aesthetics, for example, a review of studies found that debate over the reliability of visibility analysis focused more on environmental psychology questions than on the underlying physical data. Planners considered visibility mapping to be a "simple, mechanical, highly replicable" process that was standardized and well understood (Felleman 1982).

A review of studies that had used visibility mapping in planning and impact reports found that fewer than half of the studies even documented their methods, and no studies discussed the accuracy of viewshed results (Felleman and Griffin 1990).

While the number of researchers using viewshed or other types of visibility analysis has increased in the decades since these studies, the tendency to accept viewshed results with little question has not changed much. A user who is unfamiliar with the challenges of a visibility analysis is more likely to choose whatever data is available rather than considering all possible choices. Visibility analyses may be based on elevation values that are out-of-date or at an inadequate resolution. Since such choices are one of the most significant contributors to error, ill-informed data selection is likely to lead to poor accuracy.

CHAPTER 3

METHODOLOGY

3.1 VISIBILITY ANALYSIS METHODS

This research tested three approaches for the ground-to-air visibility modeling problem. Two models were made with ArcMap tools using ModelBuilder in ESRI's ArcGIS 10.1, one based on the Viewshed tool (one of the most common visibility analysis methods) and one based on the Sight Line tools. The third approach was a custom algorithm written in Python, based on sight line analysis principles and using trigonometry to check for obstructions.

3.1.1 VIEWSHED MODEL (ARCMAP)

The Viewshed tool in ArcMap includes a number of optional parameters (see Figure 3.1) which are necessary to accurately model a complex visibility scenario. In order to define the elevation of an observer point, the user can define an offset value describing the vertical distance of the observer or target features above the surface elevation. For example, a ground antenna may have a constant offset value of two feet. Since a remote-sensing airplane is flying at a constant altitude, while the ground below is constantly changing elevation as its position

changes, using a constant offset value for the flight path would inaccurately represent the airplane's vertical location. There are two possible ways to circumvent this challenge. The flight path line can be converted to a set of points, and the offset value calculated for each point by subtracting the elevation from the altitude. Or, the flight path can be assigned a constant "spot" value equal to the altitude. The two different solutions produce identical results and have negligible difference in model run time, so the offset method was used in the model tested in this research.

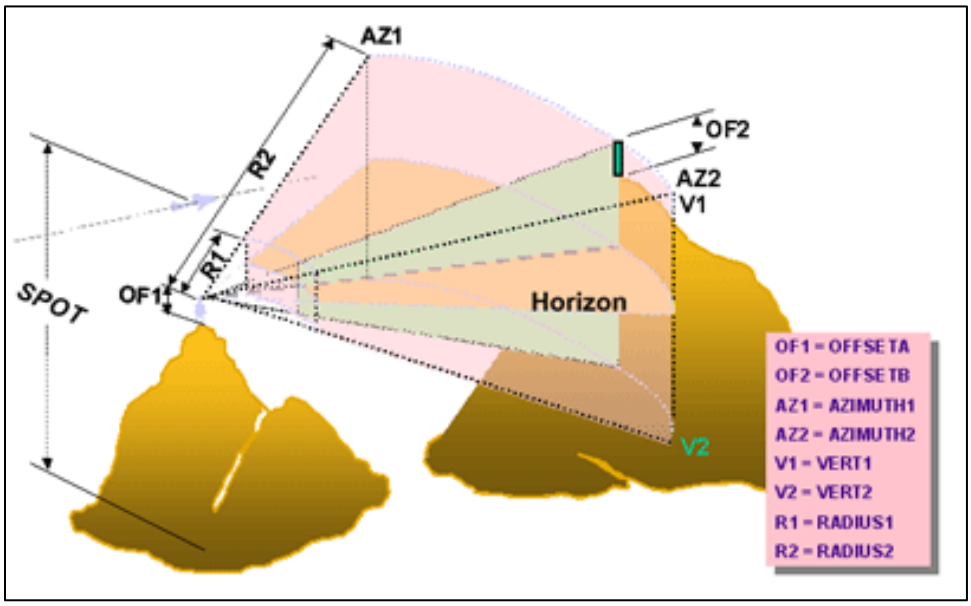

Figure 3.1: Diagram of the possible ArcMap viewshed tool input parameters in ArcMap 10.1 (Esri 2012).

In order to accurately represent the airplane's height above ground, the airplane's flight path must be used as the observer feature, rather than the proposed antenna sites on the ground. This may seem to be a poor choice: since

there are fewer antenna locations than airplane locations, using the ground station sites as the observer points would reduce the processing time. However, this decision is dictated by the Viewshed tool input parameters. Offset or spot values can only be assigned to observer features, not target features. Since the airplane's offset is highly variable and antenna offset is constant, the flight path must be used as the observer feature.

For a single observer point, viewshed output is binary: each cell of the output raster is classified as either "visible" (1) or "not visible" (0). The airplane flight path is converted to multiple observer points, representing its different locations in the air as it flies. Therefore, the viewshed result is a cumulative rather than a binary measure. In other words, if ten observer points (i.e., airplane locations) are evaluated, each cell of the output raster has values between zero and ten depending on how many of the observer points are able to see that cell. This result is then inverted to calculate the percentage of the flight path that is visible from the ground station. If n observer points can see a given target point on the ground, then that target point can see n of the total observer points. The last step of the model extracts the values from the cumulative viewshed raster at each proposed antenna location, and divides by the total number of flight path points. The complete ModelBuilder workflow can be seen in Figure 3.2.

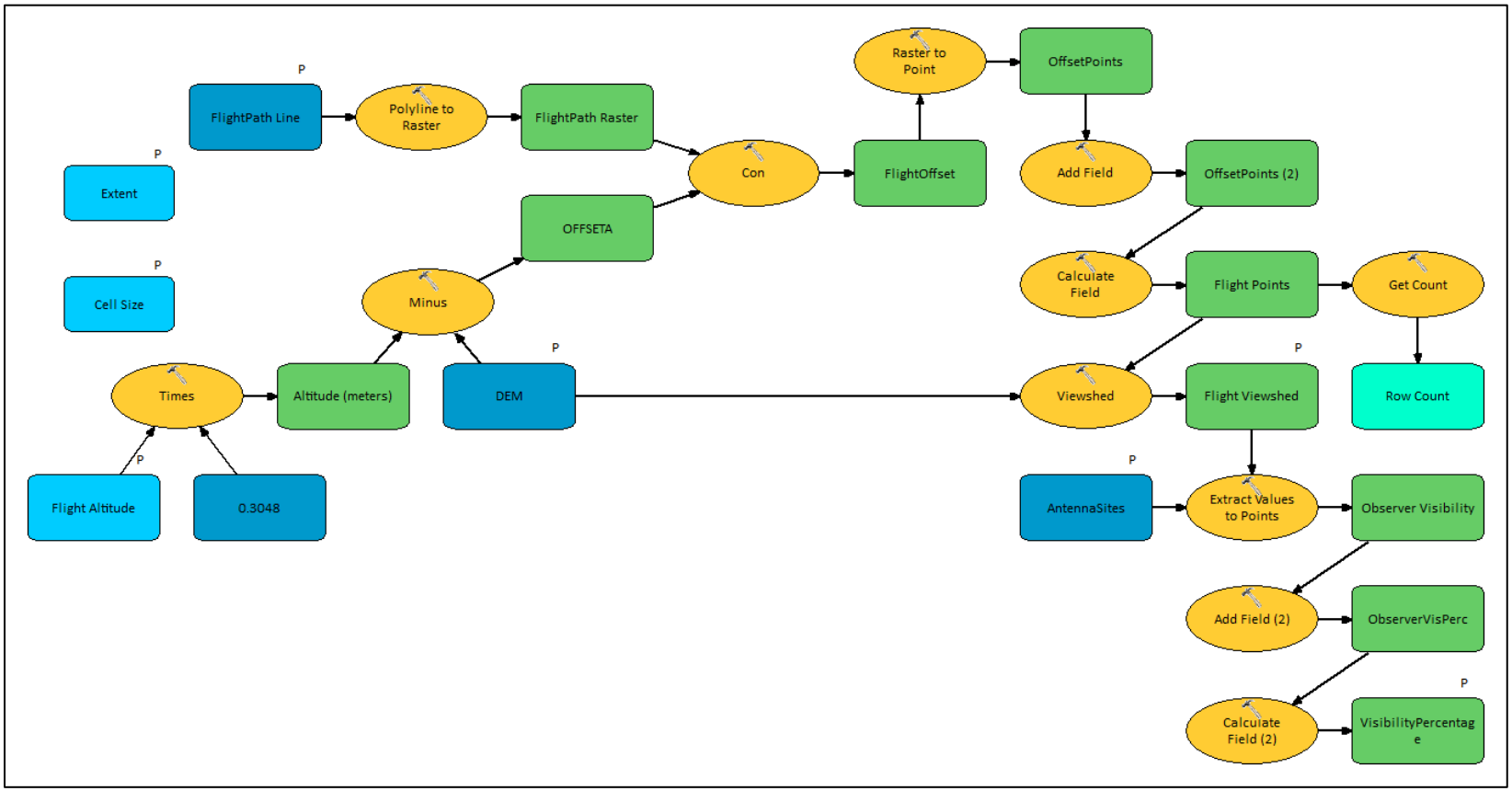

Figure 3.2: ModelBuilder diagram of the Viewshed-based visibility model.

3.1.2 SIGHT LINES MODEL (ARCMAP)

ArcMap has another tool which is designed to measure visibility along a line of sight. The workflow uses a combination of two tools. First, the Construct Sight Lines tool creates feature geometry between the observer and target features, generating sight lines between two points. Line or polygon target features are treated as a collection of points, and multiple sight lines are created. Second, the Line of Sight tool performs a visibility analysis along the constructed sight lines. The results are similar to the Viewshed tool, measuring surface visibility at all points along the sight line. The tool will indicate the location of the first obstruction along the visual path if desired.

Since the Line of Sight tool calculates visibility over a much more limited area than the Viewshed tool, it was expected to be a more effective and efficient alternative. Unfortunately, the Line of Sight tool does not actually use the three-dimensional information in the constructed sight lines in the visibility analysis process. Rather than measure visibility of points *along* these sight lines, the tool measures visibility of points *directly below* the lines, on the elevation surface itself rather than in the air. It is still a ground-to-ground process rather than ground-to-air.

Unlike Viewshed, there are no "offset" or "spot" attributes which can be included in raise the observer or target points off the ground. An attempted

workaround of adding a large offset to the elevation surface itself can "solve" the lack of offset by raising the apparent position of the target point. There are multiple problems with this modified approach, however. If multiple flight paths are present, the artificial elevation ridges are likely to falsely obstruct a number of the sight lines. Also, ArcMap raster algebra will always produce a result raster that is equal in size to the smaller of two input rasters. If the rectangular extent of the flight path is smaller than the original DSM, the modified DSM will shrink. If the observer point is outside the extent of the elevation surface, the model will not be able to execute at all.

These problems could not be reasonably overcome as part of this research project. Therefore, the Sight Lines model was not included in the testing and comparison results to be described later. A complete discussion of the model's construction and performance is included in Appendix B.

3.1.3 PYTHON ALGORITHM

The Viewshed model can be used to describe flight path visibility, but it is computationally intensive since it analyzes the full surface raster. This is far more data than are required for the antennae siting problem described, which only needs to know point-to-point visibility along sight lines. A simpler approach was a custom visibility algorithm, checking for obstructions only along the lines of

sight between each antenna and airplane point. Points which are not along the sight line are not analyzed.

The process of calculating visibility of an airborne object can be simply modeled using basic trigonometry. First, a straight line is drawn from the observer (antenna) and target (airplane) to represent the 3D line of sight between the two points. It is divided into segments, with section size depending on the available data resolution. At each segment along the sight line, the surface elevation is compared to the sight line elevation to check for obstructions.

The sight line elevation is calculated using the trigonometric concept of similar triangles (see Figure 3.3). If the elevation at a point exceeds the calculated allowed elevation z_{point}, then the sight line is blocked. The Python algorithm converts the image-based surface elevation raster into a numerical array, with each raster cell corresponding to one array element. Calculations are performed using the relative local array address rather than a geographic coordinate system. The flight path is divided up into points, one per each element it passes, and these flight points are used as targets to construct sight lines with the observer point. As the calculations described above are performed, the algorithm keeps track of the total number of obstructed sight lines, and thereby the total flight path visibility for each observer point.

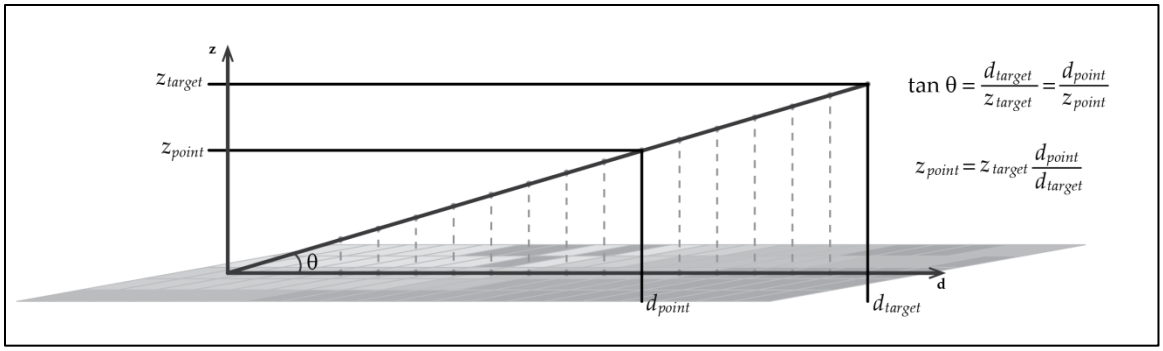

Figure 3.3: The Python algorithm uses trigonometric relations to calculate the maximum allowable elevation at a given point, and then extracts the actual elevation value from the surface raster to compare.

The algorithm was written in Python 2.7, using functions from the NumPy, SciPy, and ArcPy libraries. The complete source code is included in Appendix A. The program could be made fully open-source by using GDAL functions in the pre- and post-processing portions, eliminating any proprietary "black box" operations. As written, however, the algorithm still relies on some proprietary operations (namely ArcPy functions) for conversion to and from raster format and some additional minor functions to display results graphically. This choice was based on the usability needs of the RESPT project's target audience. Since most end-users in the emergency management community were likely to have their data in ArcMap, it was expedient to take advantage of that processing environment. None of the visibility mathematics use ArcPy operations, and the algorithm can therefore still be considered a demonstration of an open-source alternative to commercial visibility models.

3.2 STUDY AREAS

Richland County and the city of Columbia, South Carolina were used as the general study area. This choice was based on availability of data, and also the presence of a large urban area with trees and large buildings to potentially obstruct lines of sight. Locations of potential ground antenna sites and flight paths depended on the demands of various testing scenarios, and are discussed in more detail in Section 3.3. The elevation surface was predicted to have the most significant influence on accuracy and efficiency, so multiple resolutions were used. Different subsets of elevation data were used for various algorithm comparison tests, the specifics of which are discussed in Section 3.3.

Two possible sources for elevation data were considered for the model comparison testing: the National Elevation Dataset (NED) and a LiDAR dataset for Richland County, South Carolina. NED data were downloaded at 30 meter and 10 meter resolution. However, the NED data are not well suited to the GSSM question, since it does not incorporate vegetation or structure elevations which could block sight lines. In addition, the derived data for the NED coverage varied widely in source age, ranging from as recently as December 2013 for next-door Lexington County to as old as 1923-1959 for southeastern sections of Richland County (United States Geological Survey 2013).

A DSM with recent elevation data was a more appropriate solution for a visibility analysis. The City of Columbia GIS Department provided LiDAR data and building footprints for the city. The LiDAR was flown between January and March 2008 (Clifton 2013) over all of Richland County, although the available data covers less area than the NED DEMs (see Figure 3.4). However, since a DSM incorporates vegetation and structure information, it is a more appropriate raster to use for visibility analysis.

The LiDAR point cloud was converted into a surface elevation model (DSM) at 1 meter, 3 meter, 10 meter, and 30 meter resolutions using the ArcMap LAS Datatset to Raster tool. First returns were used to capture as many obstructions as possible. The maximum value in a particular cell was chosen as the elevation value instead of an average or interpolated value, and a linear void fill method (triangulating across cells with no LiDAR points) eliminated data gaps. In the resulting DSMs, surface features such as buildings and vegetation were clearly visible in addition to the base terrain elevation (see Figure 3.5)

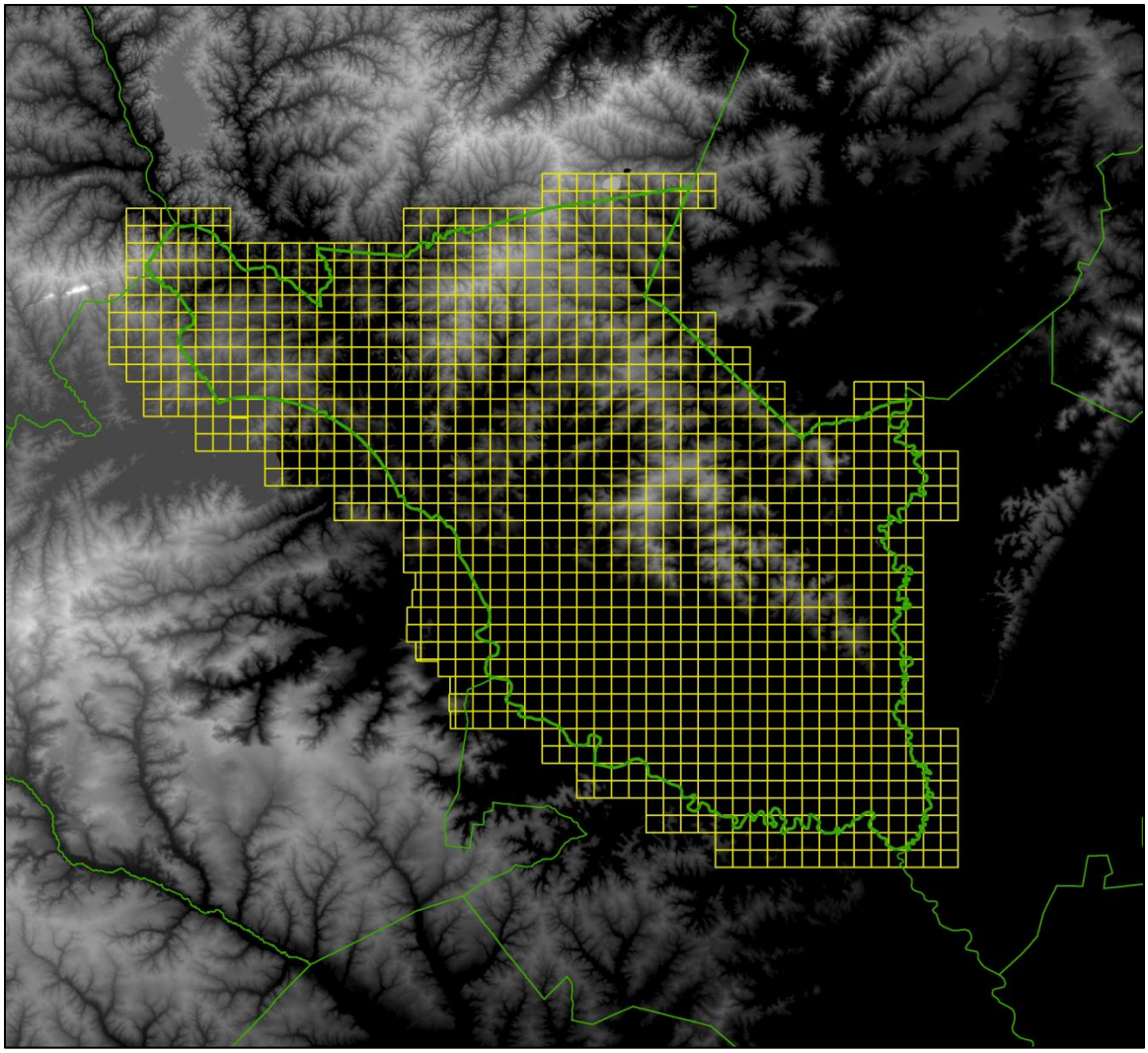

Figure 3.4: Available LiDAR data coverage (yellow grid) compared to the Richland County boundary (green outline) and NED DEM coverage.

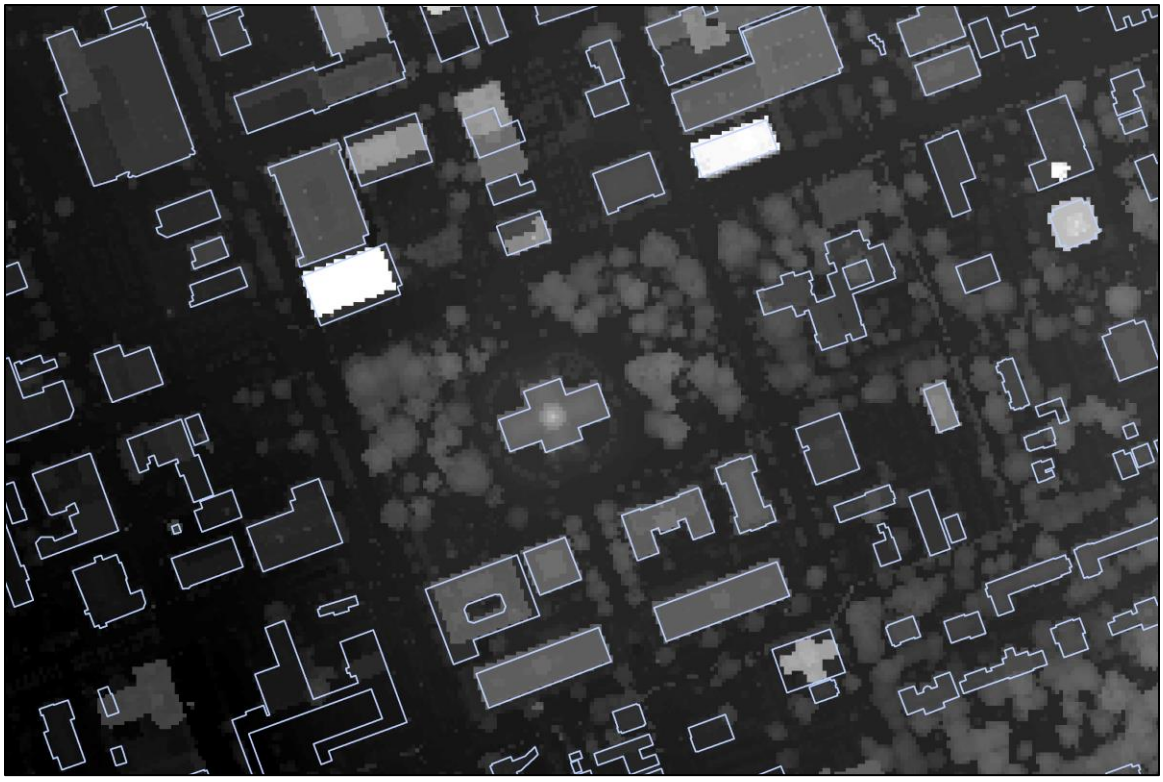

Figure 3.5: The 3 meter resolution DSM of downtown Columbia, SC, derived from 2008 LiDAR data. Building footprints are outlined in light blue. The South Carolina State House grounds are clearly identifiable as the domed building in the center surrounded by numerous trees and other government buildings.

3.3 MODEL COMPARISON

The Viewshed model and Python algorithm were evaluated using three performance metrics: speed, sensitivity to inputs, and accuracy. The results of the performance tests were used to describe basic characteristics of the three different methods, and to compare their relative performance and judge which would be the most suitable for a time-sensitive application in disaster response. Each test was performed on a laptop computer with 4 GB of RAM and a dual-core 2.2 GHz processor. The Python algorithm was run in 32-bit mode.

3.3.1 EFFICIENCY

The most important characteristic of this research question was efficiency. If a visibility analysis takes longer than sixty minutes to run, then there is no significant time advantage over waiting for an imagery flight to land. Thus, if an analysis cannot be completed on a desktop computer in under an hour, the method will probably not be suitable for emergency response purposes. The time limit was established by estimating round-trip travel time from the South Carolina Emergency Management headquarters to the second-nearest metropolitan airport, Owens Field (approximately half an hour), and doubling it to allow for major road closures that might result from a natural disaster.

A variety of hypothetical scenarios were created. Five candidate antenna placements were chosen that could be considered as reasonably accessible during disaster response (see Figure 3.6). The South Carolina Emergency Management Division headquarters would have been an ideal antenna site since it would plausibly be a primary processor and distributor of received data; unfortunately, it is too far from Richland County and was outside the DSM coverage. Hospitals are likely to have power in the event of a disaster, and so Lexington Medical Center (1), Palmetto Health Richland Hospital (2), and Palmetto Health Baptist (3) buildings were chosen as potential sites. A state government building that is reasonably tall with a flat, accessible roof is the Hampton Building (4) on

the South Carolina State House grounds. The BB&T building (5) was also chosen as an observer location simply because it is the tallest building in the city.

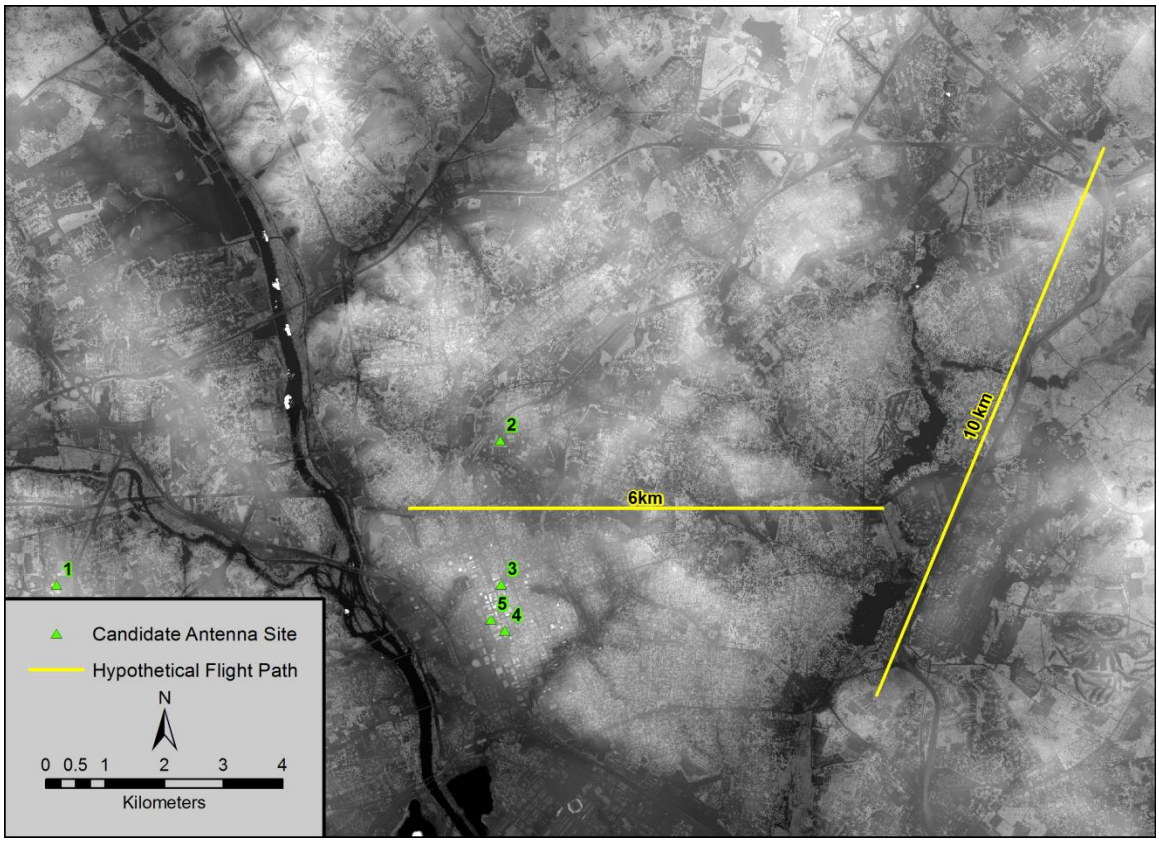

Figure 3.6: Antenna sites and flight paths in the greater Columbia area used for the efficiency tests.

Each antenna site was located on the roof of the building by using a combination of the 1-meter DSM and aerial photographs to find a flat area that was not occupied by fans, HVAC equipment, antennae, or other impediments. The rooftop also needed to be obviously accessible – the highest level of the Palmetto Health Richland Hospital building, for example, had no obvious access point, and the next highest rooftop (20 meters lower) in the hospital campus was chosen instead. Two arbitrary flight paths were drawn over a hypothetical

impacted area, and combinations of observer points, flight lines, and DSMs at different resolutions were used to run a series of efficiency tests.

3.3.2 SENSITIVITY TO INPUTS

In addition to testing efficiency of a hypothetical emergency response situation, a sensitivity analysis was performed on each model. The size of the analysis area, length of flight path, and quantity and position of candidate ground antenna sites were varied independently of one another. These criteria established whether the models would continue to perform adequately in a variety of situations, depending on the scope of the disaster response. The models were tested multiple times, and the execution time correlated to the input variables.

Analysis area size, measured in raster cells, is dependent on the elevation surface resolution and the geographical area. Testing for this input parameter was done on the 30-meter resolution NED DEM, with a single antenna site and a 1500-meter flight path (see Figure 3.7), and different analysis areas were created by modifying the processing extent in the geoprocessing environment settings. Total analysis area size ranged from 572,951 cells to 2,118,904 cells.

Flight path length measures the number of flight path points used as targets for the respective visibility analyses. The number of points depends on the number of flight paths, the length of each one, and the elevation surface

resolution. This parameter was tested using the 10-meter resolution LiDAR DSM, with a single antenna site (see Figure 3.7). To avoid influence from the observer distance, the total flight path length was increased by duplicating one feature in the same location – in other words, a 9,000 meter flight path consisted of six copies of the 1,500 meter path, and so on. The total flight path lengths tested ranged from 1,500 to 30,000 meters, or 150 to 3,000 flight path points.

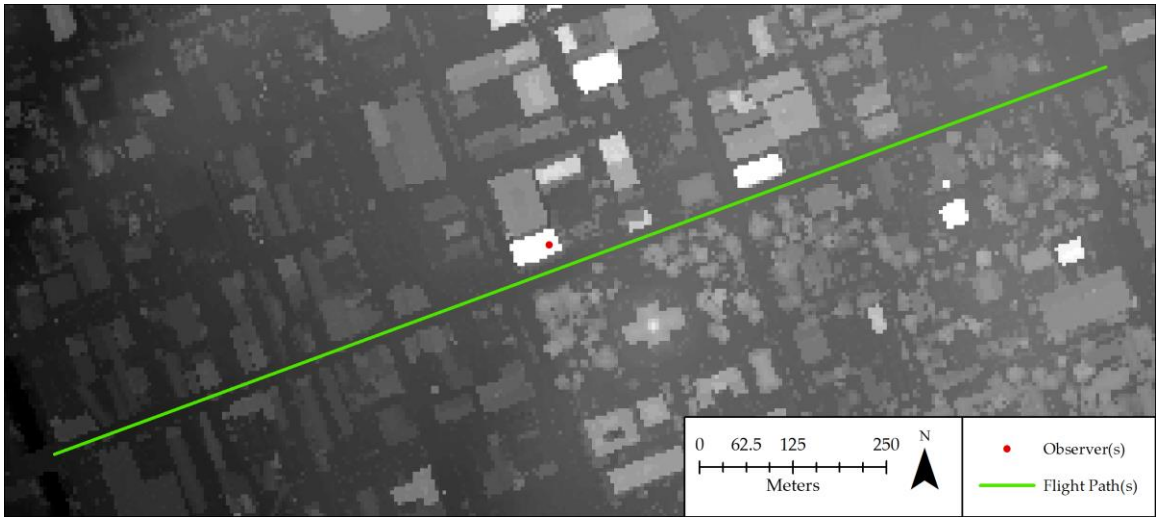

Figure 3.7: The analysis area size, flight path length, and number of observers were tested on a 3-meter DSM subset. In tests with multiple observers, all were co-located on the same point.

The *number of antenna sites* was treated as a distinct input parameter from the distance between antenna and flight path, since these are independent characteristics of the feature class used as an analysis input. To avoid influence from the observer location, up to 20 observers were co-located on the same point (see Figure 3.7). The *distance from antenna to flight path* affects the length of derived lines of sight, which may influence the execution time. Observer distance

from the flight path ranged from 500 to 5,000 meters, on a 1-meter resolution LiDAR DSM and looking at a 750 meter flight path (see Figure 3.8).

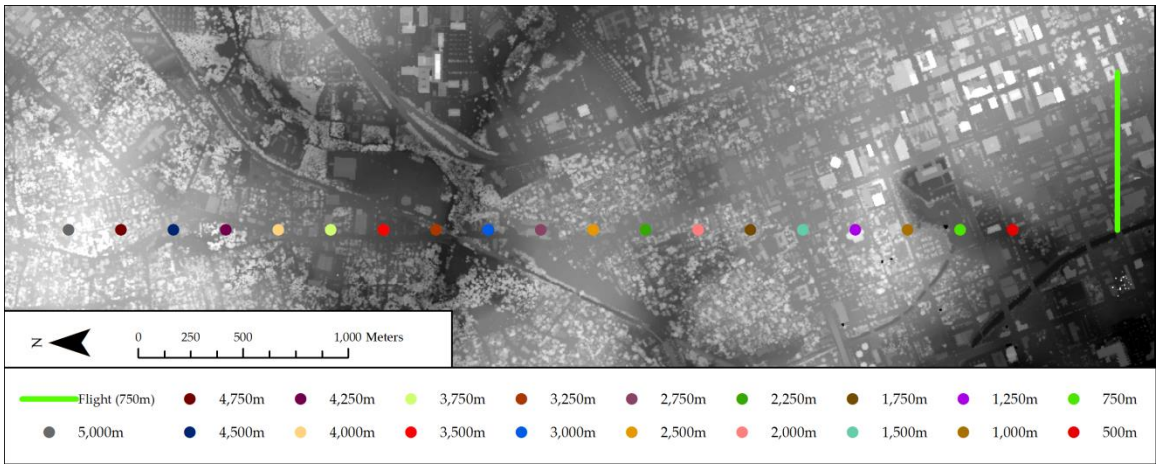

Figure 3.8: The observer distance test was run on a 1-meter DSM subset.

3.3.3 ACCURACY AND VALIDATION

Measuring the accuracy of visibility analyses is not a straightforward task due to the difficulty of establishing a "ground truth" against which to compare. It was not feasible to contract an imagery flight to track its visibility from a set of potential antenna sites. Two alternative approaches were therefore devised to test accuracy.

In the first test, a fictional elevation surface was created to establish a small, predictable study area. On a constant raster of zero elevation, all observer points would be able to see 100% of an overhead flight path. This artificial DSM provided a mathematically predictable area, and a way to establish a baseline accuracy performance for the different methods.

In the second test, ground-to-ground verification of visible and not-visible points was used to validate the ground-to-air visibility methods tested. Typical operation of a visibility algorithm would use a complex elevation surface describing real-world terrain validation. In order to validate performance in such situations, photographs were taken from an established reference point on top of the Bull Street parking garage on the USC campus (see Figure 3.9).

Identifiable points in the photograph that were visually clear were mapped as target points on a 1-meter resolution LiDAR-based DSM (see Figure 3.10), the most accurate elevation surface that was available. Points that could not be seen from the reference point were also included on the map to test not-visible measurement. (Any points obstructed by trees were not chosen as not-visible validation data, since the six-year time lapse between the 2008 LiDAR data and the photograph could allow significant vegetation growth.) The results of the various visibility analysis methods were compared to the visual confirmation as validation of accuracy.

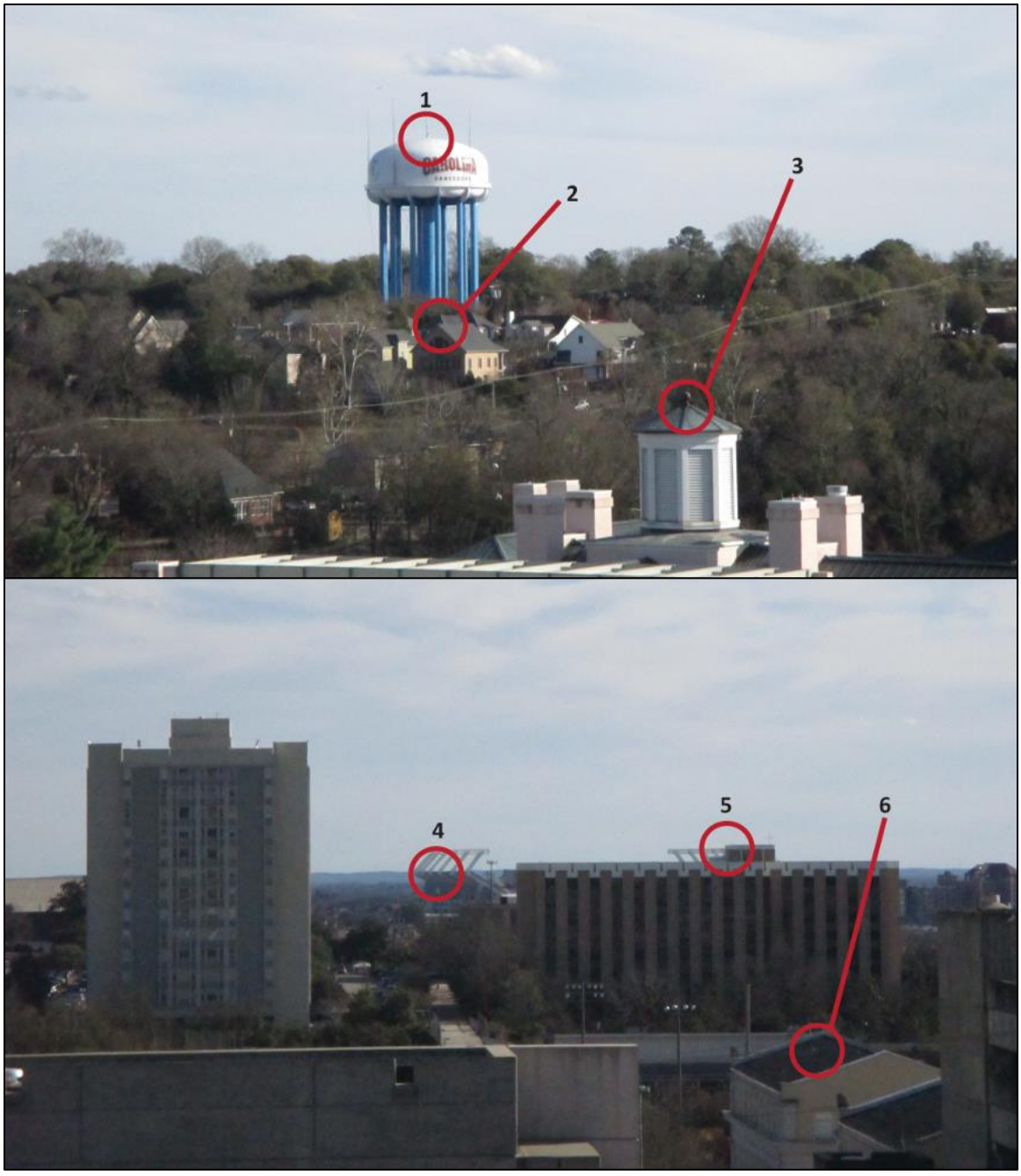

Figure 3.9: Photographs taken from top of Bull Street Garage, with validation points identified.

Figure 3.10: Overhead map with identified "visible" and "not visible" target points labeled.

CHAPTER 4

RESULTS

4.1 EFFICIENCY

The comparison for the Viewshed model and Python algorithm shows that the Python algorithm is faster for all study areas, especially at finer raster resolutions. The Viewshed model is two orders of magnitude slower at coarse resolution (10 meter), and three orders of magnitude slower on a fine resolution (3 meter) DSM. Once each method exceeded the sixty minute threshold, further efficiency testing was not necessary.

The Viewshed model (see Table 4.1) reached the time limit on the shortest flight path on the 10m × 10m DSM. Changes in the number of observer points did not affect the Viewshed model, and so that parameter was not changed during its efficiency tests.

The Python algorithm (see Table 4.2) completed the visibility analysis more quickly, and was therefore tested on a larger assortment of scenarios and at a higher resolution. Including different numbers of observer points affected the performance, and so this parameter was varied as well as the flight path length.

This emulates real-world usage. If the end user has done pre-screening on the candidate antenna points and knows that some of them are not accessible (for example, the power cannot be restored in the BB&T building), there is no need to measure its visibility. Leaving out such points can both improve the run time and generate more relevant results.

Table 4.1: Average run time for the Viewshed model on a 321 km^2 analysis area.

Raster Resolution	Flight Path Length (m)	Average Run Time (minutes)
30m × 30m	6,000	4.27
	10,000	7.39
	16,000	10.31
10m × 10m	6,000	83.12

Interestingly, the Python algorithm began to encounter an unexpected computational threshold in addition to the established processing time requirement. There is a limit on the amount of memory which can be allocated to hold a NumPy array, which the algorithm uses as an alternative to raster data for fast reference and calculations. When the analysis area size (measured in cells) exceeds the permitted dimensions for an array, the algorithm throws a memory warning and stops running. This capacity problem was first noticed on the full 3m × 3m DSM, which needed to be clipped to the analysis area to fit into array

memory. The 1m × 1m DSM could not be analyzed at all (although its run time would have exceeded the allowed limit anyway).

Table 4.2: Average run time for the Python algorithm in various scenarios.

Raster Resolution	Flight Path Length (m)	Observers Included	Average Run Time (minutes)
30m × 30m	6,000	5	0.73
	10,000	5	2.20
	16,000	5	2.89
10m × 10m	6,000	3	4.19
		4	5.09
		5	6.02
	10,000	3	12.51
		4	15.89
		5	19.30
	16,000	3	16.55
		4	20.78
		5	25.17
3m × 3m	6,000	3	45.09
		4	55.14
		5	65.47
	10,000	3	138.22
		4	175.85
		5	214.36
	16,000	3	187.44
		4	239.06
		5	279.61

One additional test of efficiency was performed during the artificial surface accuracy test. The Viewshed model finished the analysis of a 100 × 100 cell raster in approximately one minute. The Python algorithm was then tested with an equivalent number of observer points (10,000) to compare its accuracy performance. The Python algorithm took 2 hours to finish, two orders of magnitude longer than the Viewshed model. While interesting, this result is not particularly informative for the evaluation of performance during normal operation with far fewer observer sites.

4.2 SENSITIVITY TO INPUTS

Through the sensitivity analysis process, the influence of individual input parameters on the processing speed could be examined in more detail. There were significant differences in which parameters were influential on each method, and to what degree. When the same input parameters were used, execution times were highly repeatable execution and consistent. This predictable behavior is also apparent in the very high coefficients of correlation found for each of the relationships.

4.2.1 ANALYSIS AREA SIZE

Size of the analysis area has a very strong correlation ($R^2 = 0.9986$) to the execution time of the Viewshed model (see Figure 4.1). Depending on the total area of interest and the available DSM resolution, this relationship is clearly one

of the primary drivers of the long execution times seen in the efficiency tests. However, the number of cells in the surface raster had no influence on the execution time of the Python algorithm.

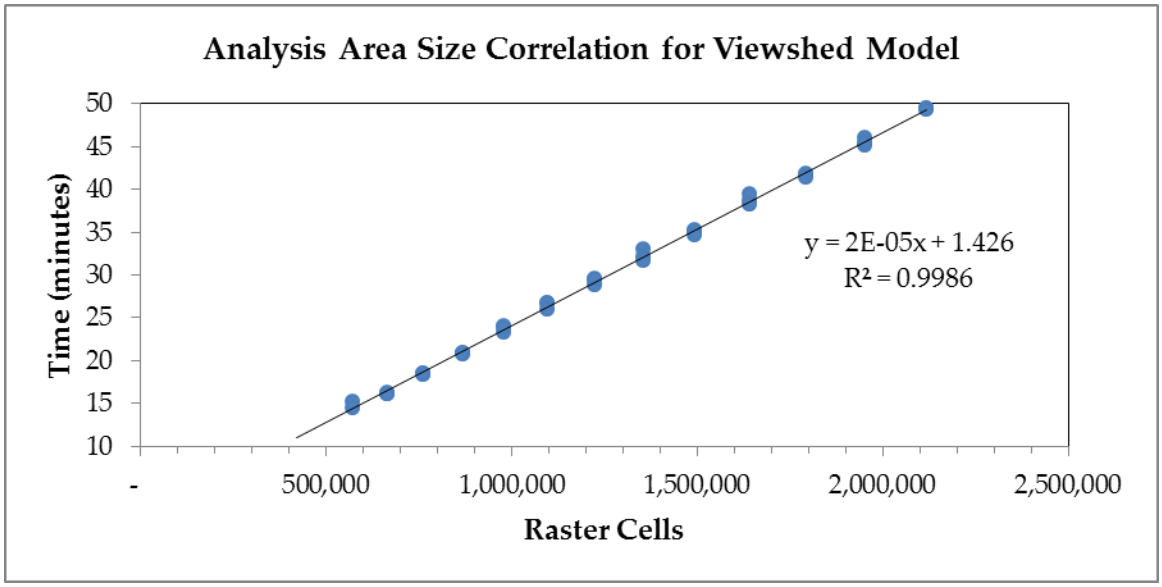

Figure 4.1: Correlation between analysis area size (measured in raster cells) and Viewshed model execution time.

4.2.2 NUMBER AND LOCATION OF ANTENNA SITES

The number and location of observer points had no noticeable effect on the Viewshed model. Since the observer points only become relevant to the model when extracting the visibility results to the points, the amount of time required for each point is negligible compared to the much longer execution time of the Viewshed tool within the model.

In contrast, the number and position of observer points heavily influenced the Python algorithm. Under actual operating conditions, the quantity and

position would not necessarily be able to be treated as separate variables. When treated as separate in a controlled study area, however, there were clearly different influences from the two different characteristics. Both the quantity of observer points (see Figure 4.2) and the distance from the observer to the flight path (see Figure 4.3) were strongly correlated ($R^2 = 0.9996$ and $R^2 = 0.9997$, respectively) to execution time for the Python algorithm.

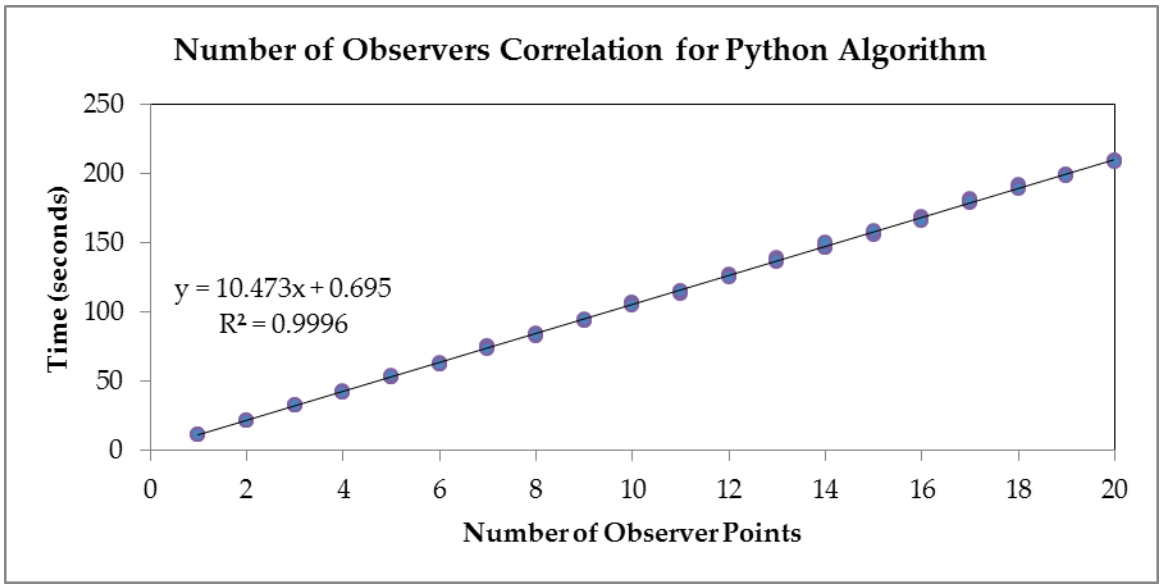

Figure 4.2: Correlation between number of antenna sites and Python algorithm execution time.

Increasing the distance between flight path and antenna position appeared to cause the execution time to increase more rapidly than adding additional observer points (approximately 160 seconds per kilometer, versus 10.5 seconds per observer point). However, the number of observers test was completed with points that were very close to the flight path, and this must be

taken into account when interpreting the results. The possible variations in observer location make a universal predictive equation difficult.

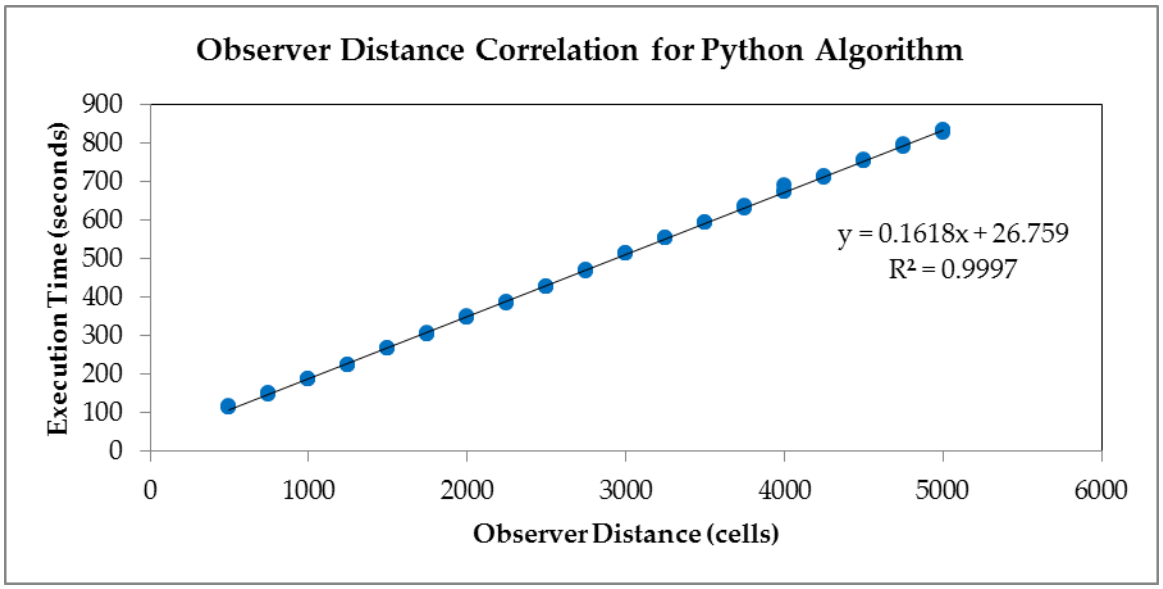

Figure 4.3: Correlation between antenna position and Python algorithm execution time.

4.2.3 TOTAL LENGTH OF FLIGHT PATHS

The length of the flight path and the resolution of the elevation surface together determine the number of flight path "target" points that will be analyzed. The flight path length had an effect on both methods. The total length was the significant factor, not the number of flight paths; in other words, a single 5,000 meter flight path would take the same amount of time to execute as ten 500 meter flight paths. The strongest influence was on the Viewshed model (see Figure 4.4), adding nearly a tenth of a second for every additional flight path point included (strong linear correlation of $R^2 = 0.9978$). Extending the flight path

added less time per point to the Python algorithm (see Figure 4.5) execution times (strong linear correlation of $R^2 = 0.9992$).

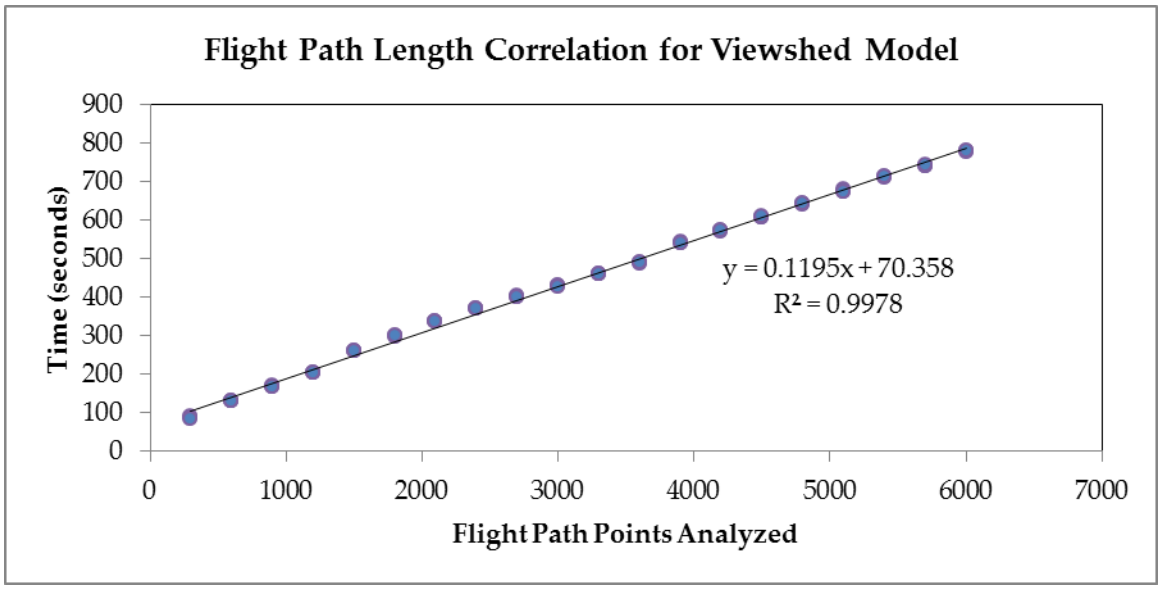

Figure 4.4: Correlation between flight path length and Viewshed model execution time.

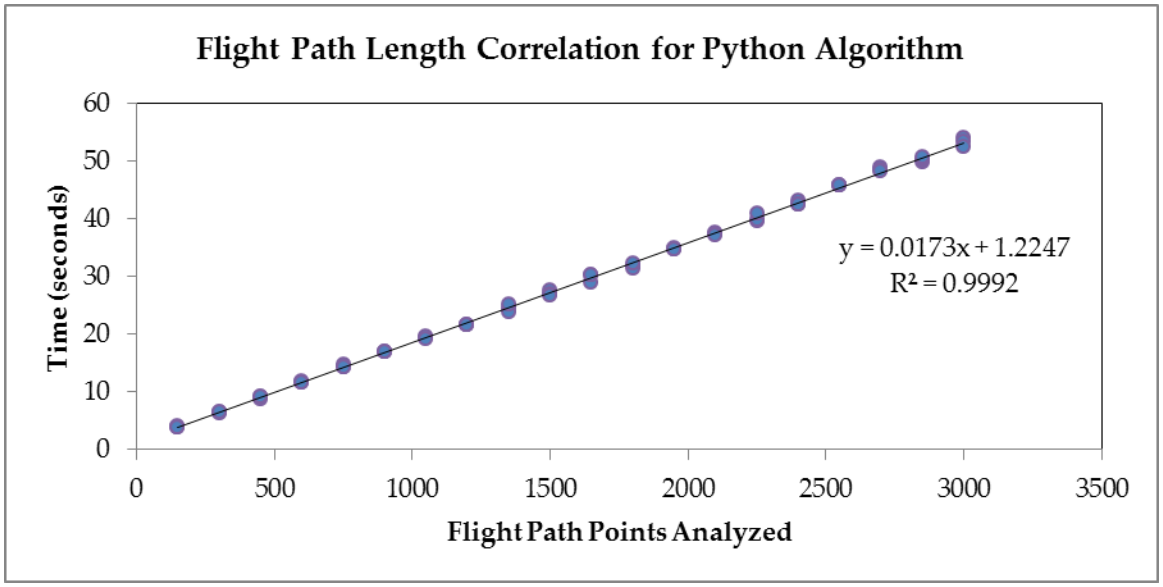

Figure 4.5: Correlation between flight path length and Python algorithm execution time.

The difference between the execution time of the Viewshed model and Python algorithm is best seen by comparing the two correlations on the same graph (see Figure 4.6). Not only does the Viewshed model take longer to execute at all points, but its execution time increases much more per point.

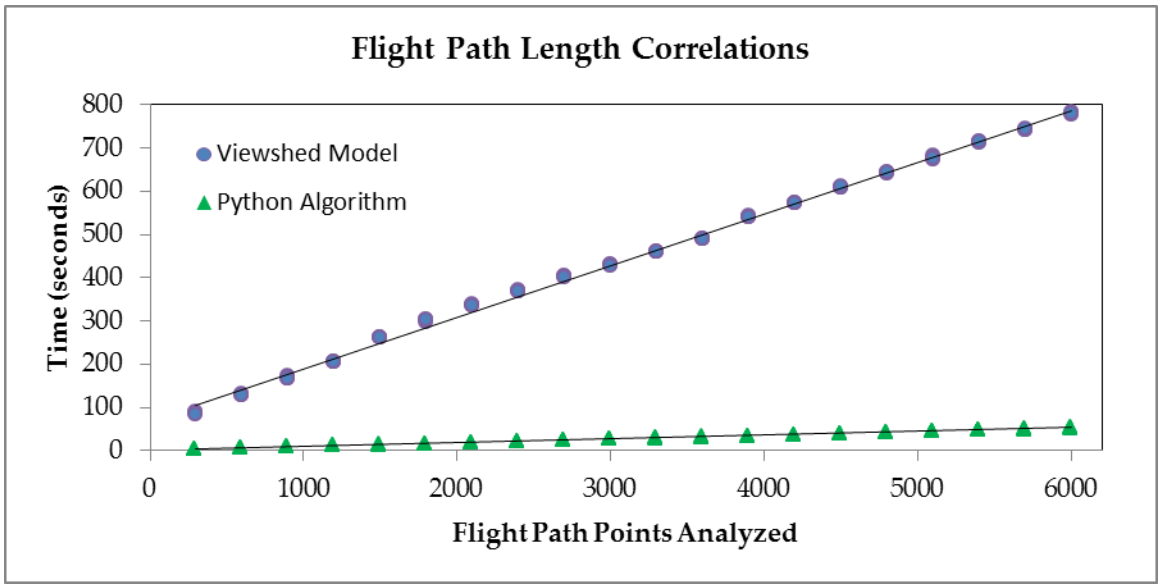

Figure 4.6: Comparison between Viewshed model and Python algorithm correlations to flight path length.

4.3 ACCURACY AND VALIDATION

One notable oddity in the Viewshed model results in a subset of cells which are coded as unable to see one flight path point. In the test on a straight north-south flight path, these cells are all directly underneath observer points. This may imply either that an observer point is considered unable to see the surface directly beneath its location, or the surface cell's visibility is not even measured directly beneath an observer point location. However, if a straight

east-west flight path is tested instead, cells with reduced visibility are not limited to only observer points, but also spread up to the north of the flight path (see Figure 4.7). In other words, the orientation of the flight path will have an influence on the relative accuracy of the Viewshed model results. This effect has no apparent explanation. It is presumably caused by the Visibility tool itself, and the proprietary nature of the tool precludes a more definitive answer.

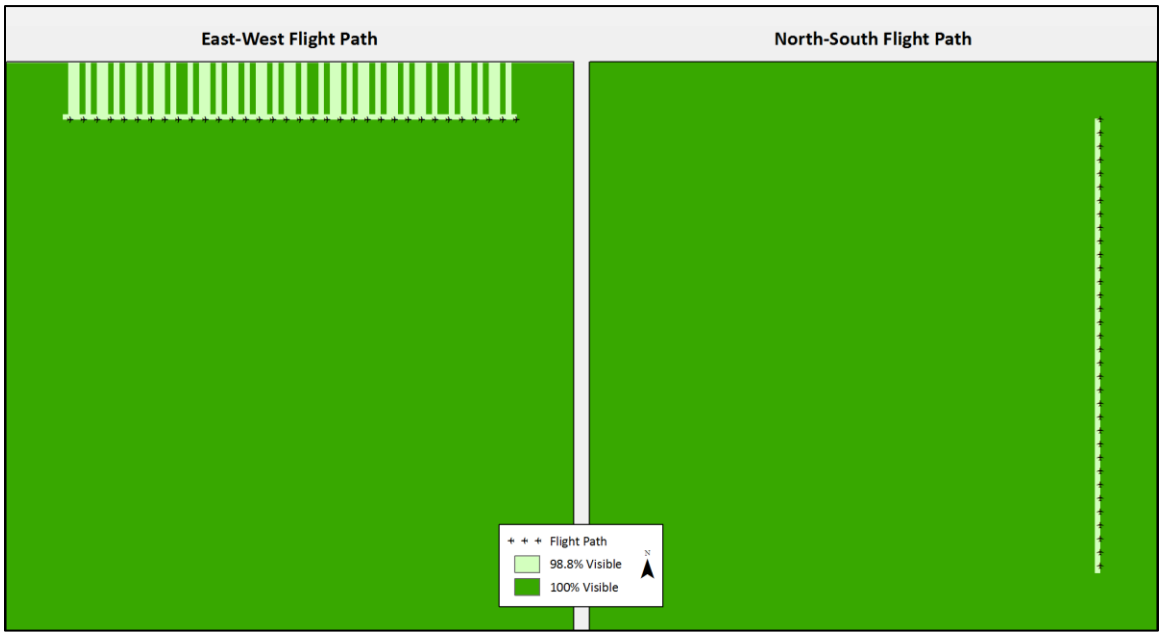

Figure 4.7: Results from Viewshed model test on a flat surface.

To look for similar behavior from the Python algorithm, it was tested by converting the 100 × 100 cell raster to 10,000 corresponding observer points. The Python algorithm correctly coded all cells as able to see 100% of the flight path, for both the east-west and north-south lines.

The visibility results in the validation tests were better (see Table 4.3). Both methods correctly identified all of the visibility and not-visible target points. Failing to account for the observer height (1.5 meters) caused approximately half of the visible target points to be marked as not visible, although all the not visible points were still correctly evaluated. This result emphasizes the need for accurate offsets and vertical position, and demonstrates the inaccuracy that can come from user error and erroneous inputs.

Table 4.3: Visibility of points around Bull Street Garage.

Point	Actually Visible	Viewhsed Model	Python Algorithm
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	Yes	Yes	Yes
7	No	No	No
8	No	No	No
9	No	No	No
10	No	No	No
11	No	No	No
12	No	No	No

CHAPTER 5

DISCUSSION

Overall, the results of this thesis were not surprising. Knowing the amount of extraneous analysis the Viewshed tool performs, it was expected to be much less efficient than an algorithm which only measures lines of sight. The process of testing the Viewshed model and Python algorithm as an alternative did provide a number of useful insights into attempts to measure ground-to-air visibility.

The Viewshed model was sensitive to raster resolution and total flight path length, with no significant impact from the number of observer points. In contrast, the Python algorithm was sensitive to changes in the number of observer points and total flight path length. Understanding these different input sensitivities is central to a comprehensive understanding of the relative efficiency of the visibility analysis methods. It also provides guidance for end users, helping them choose appropriate data to constrain execution time to a desired limit.

For most disaster response analyses, the Viewshed model would simply be too slow. At all but the coarsest resolution, it far exceeded the sixty minute

threshold. Even though a 30-meter DSM would allow it to perform quickly enough, such a coarse resolution drastically oversimplifies surface features and contributes a high level of uncertainty to the model's results. It is unlikely that most disasters will be constrained to small impact areas close to candidate antenna sites, so the appropriate model must be able to deal with large analysis areas rapidly. The Viewshed model cannot.

One important outcome of the accuracy testing was that algorithm results were consistent for multiple evaluations of the same input parameters. This does not indicate that there is not any error at all from the algorithms themselves, but does mean the error is non-random. Any error attributed due to the algorithm is therefore due to a bias, rather than stochastic influences.

The accuracy of the visibility analyses in a real-world validation (Section 4.3) shows that neither method was completely accurate. However, this is not a serious enough problem to rule out either method. Both models underestimated visibility, and had no false positives. This is not ideal, of course, since too many false negative results can eliminate candidate antenna sites which should be suitable. But, such behavior is preferable to an analysis that overestimates visibility, which could lead to deployment of resources to a site which cannot see enough of a flight path, wasting time and money.

The Viewshed model's inaccuracy on a fictional surface demonstrates clearly that the underlying visibility algorithm may cause unknown and unpredictable errors. It is possible that similar inaccuracy exists in the Python script, because any mathematical model relies on simplifications and assumptions about the physical world that can result in errors. The disadvantage the Viewshed model has in this case is that its algorithm is proprietary. The end user is forced to guess at the cause and scope of algorithm errors.

This leads to the discussion of an additional factor to consider when selecting a visibility analysis method: usability. The ArcGIS ModelBuilder environment can simplify the process of scripting tools, and it is part of a familiar software package for a majority of GIS users. Either a ModelBuilder or Python script can be integrated easily into an ArcMap workflow, but the proprietary Viewshed tool cannot be exported to open-source alternatives such as GRASS or QGIS, Using the Viewshed tool limits the end user's software options.

In contrast, the Python algorithm can be adapted to a fully open-source set of libraries without affecting performance, providing flexibility for users and basic code for future visibility algorithm development. The open-source approach also provides transparency for users and researchers interested in the algorithm's methods and limitations. Knowing how a tool performs its analysis

and models reality will provide insight into the reliability and accuracy of its results.

One question of visibility that was not adequately addressed in these models was directionality. This would not be an issue for mobile antenna deployment, which is set up on demand and can therefore be pointed toward the flight path. However, a permanent, more rigid antenna mount might be built, such as on the roof of an emergency management agency, in expectation of its eventual use as a ground station site in disaster response. In such a case, there may be a mechanical restriction on what direction the antenna can face and therefore what area would be visible. Neither the Viewshed model nor the Python algorithm is able to account for directional restrictions. Results would have to be manually checked for accuracy.

An interesting potential expansion of this research is the application of visibility analysis in the planning phases of the disaster response cycle. The work in this thesis was focused on the need to perform a visibility analysis under post-disaster time constraints. With some adaptation, however, the visibility principles and methods described in this work could be used to create data to describe what area of sky is visible from a proposed antenna site. This predictive analysis takes longer to run, but if performed during disaster planning instead of disaster response, there is little concern about algorithm execution time.

One possible disaster planning approach is to use the existing GSSM with flight paths over areas which are at high risk for damage in a disaster. Examples might include a riverbank or shoreline communities that are susceptible to flooding, the area surrounding a power plant, or indeed any disaster scenario which an emergency management team plans for. Running an analysis with the hypothetical data can narrow down the choice of candidate antenna sites for each potential hazard, creating a short list that can be quickly accessed by a disaster manager when an event occurs.

This planning stage use enables significant GIS processing and analysis to be done when time is not a concern, so site selection and resource deployment can happen much more quickly. This does have the advantage of being able to use an existing tool and workflow plan. The disadvantage is that it only works for expected or predictable scenarios, of course; the burden is on the user to come up with sufficient hypothetical disaster impacts to build a useful list of coverage.

An alternative, and realistically more robust, approach is to build a new tool, based on the theories and methods described in this research. Since the Python algorithm was not designed to be predictive, it does not evaluate visibility at points other than the antenna sites or flight path. The Viewshed tool is designed to analyze a large area all at once, and could be well suited to identifying potential antenna sites. (This would be particularly true if a

Viewshed algorithm allowed a constant SPOT value to be assigned to target points, which would significantly simplify the model and reduce computational complexity. Such a change depends on ESRI's software development team, however.) Each method would require some adaptation to be able to deliver useful and informative results, however.

An ideal planning workflow would be to identify polygonal outlines of flight locations which can be seen from proposed antenna sites. This would establish "zones" of a city or county which could be seen from each vantage point. These observation zones would be stored, and could be accessed quickly in the event of a disaster, overlaid over regions known to be damaged in order to generate optimum flight paths. This enables selection of the best ground station site with minimal required processing time and GIS skill level. Such polygons would be of great value to a project like RESPT, which provides a platform for emergency planners and responders to collaborate. More importantly, it would not require speculative scenario planning, but could be used for disasters of any type or geographical scope.

Finally, it is worth noting that the Python algorithm tested in this research is not the most efficient solution possible for ground-to-air visibility analysis. The script could be further streamlined and optimized in a number of ways. Pre-screening the elevation data to ignore points which could not possibly block

lines of sight (e.g., any locations with an elevation less than the antenna site) would be a first step, using conditional logic up front to reduce the amount of more computationally calculations required later in the algorithm. The mathematical evaluation could be performed in a different order (an approach which is discussed in more detail in Appendix C). Additionally, application of probability and spatial autocorrelation assumptions could further speed up the algorithm by reducing the amount of cells that need to be checked. Such changes could reasonably be introduced while simultaneously modifying the algorithm to be predictive rather than responsive.

CHAPTER 6

CONCLUSION

The results of this exploration of visibility analysis methods are informative in a variety of ways. The model comparison and validation testing established that the Python algorithm is a suitable method for the GSSM tool, one of the primary research questions behind this thesis. In addition, detailed research into the sensitivity of two basic visibility models provides useful data for both end users and future researchers.

It was noted nearly twenty-five years ago that viewshed algorithms were not well documented and could produce results that are inconsistent with other algorithms (Felleman and Griffin 1990). The data from this thesis indicates that the issue has not changed over time: the Viewshed model had some inexplicable inaccuracies (see Section 4.3). The ArcMap help files, while providing a thorough description of how to operate the tool, do not describe the underlying algorithm function. There is clearly an ongoing need for users of any visibility tools to be aware of, and document, the possible shortcomings of their analyses.

In both the Viewshed model and Python algorithm, the strong correlations between input parameters and execution time are related to the number of times the basic calculation is performed and how long that calculation takes to complete. The basic Viewshed model calculates analyzes visibility of an entire raster, while the basic Python algorithm calculates an inverse tangent. Repeated iterations of the former process take longer to complete. This concept can be extended to other visibility analysis methods: if the basic calculation is simpler, the execution time is faster.

Given the significant influence of DSM accuracy on visibility accuracy, it is worth emphasizing the importance of choosing an appropriate elevation surface. The DSM used for this research was adequate for the purposes of testing typical efficiency and accuracy performance of visibility analyses. However, it may not be the best DSM to answer the question of where a ground antenna site should be placed in the event of a disaster in Columbia, SC. The City of Columbia did not include any metadata with its LiDAR, and even the date that the data was captured was only known within a range of months. The data are also six years old, a period of time which could allow significant changes in both vegetation and human structures in the area. (An interesting exercise that was outside the scope of this thesis would be land use change analysis from 2008 to 2014 to more precisely locate areas of probable development and greatest DSM

uncertainty.) Finally, no processing was done to remove artifacts or errors from the raw point cloud before it was converted to an elevation raster. The resulting DSM is not expected to be as accurate as possible, but it is representative surface for the research project and is similar to what may be available for many disaster response situations.

One significant restriction on the visibility analyses tested was the limit to only one elevation surface. There are two fundamentally conflicting concerns when choosing an appropriate DSM resolution. Fine resolution is needed close to the antenna site in order to identify relatively small obstructions that are liable to block lines of sight, but which would be generalized at coarse resolutions. However, such fine resolution causes significant reductions in efficiency, as a much greater number of calculations must be performed. The ability to use two elevation surfaces of different resolution – the fine resolution close to antenna site, the coarse resolution further away – could improve efficiency without sacrificing accuracy.

REFERENCES

Amidon, Elliot L., and Gary H. Elsner. 1968. "Delineating Landscape View Areas: A Computer Approach". PSW-RN-180. Berkeley, CA: U. S. Department of Agriculture, Forest Service, Pacific Southwest Forest and Range Experiment Station.

Bagli, Stefano, Davide Geneletti, and Francesco Orsi. 2011. "Routeing of Power Lines through Least-Cost Path Analysis and Multicriteria Evaluation to Minimise Environmental Impacts." *Environmental Impact Assessment Review* 31 (3): 234–39.

Beesley, Bridget Joan. 2002. "Sky Viewshed Modeling for GPS Use in the Urban Environment". M. S., University of South Carolina.

Benedikt, Michael L. 1979. "To Take Hold of Space: Isovists and Isovist Fields." *Environment and Planning B: Planning and Design* 6 (1): 47–65.

Bruzese, Victoria M. 1989. "Terrain Analysis and Geographic Information Systems". Master of Science, Corvallis, OR: Oregon State University.

Clifton, Christopher D. 2013. "Personal Communication", March 6.

Congalton, Russell G. 1997. "Exploring and Evaluating the Consequences of Vector-to-Raster and Raster-to-Vector Conversion." *Photogrammetric Engineering and Remote Sensing* 63 (4): 425–34.

Dozier, Jeff, and James Frew. 1990. "Rapid Calculation of Terrain Parameters for Radiation Modeling from Digital Elevation Data." *IEEE Transactions on Geoscience and Remote Sensing* 28 (5): 963–69.

Esri. 2012. "ArcGIS Help 10.1 - Using Viewshed and Observer Points for Visibility Analysis." *ArcGIS Resources*. November 8.

Felleman, John P. 1982. "Visibility Mapping in New York's Coastal Zone: A Case Study of Alternative Methods." *Coastal Zone Management Journal* 9 (3/4): 249–70.

Felleman, John P., and Carol Griffin. 1990. "The Role of Error in GIS-Based Viewshed Determination: A Problem Analysis". Technical Report EIPP-90-2. Institute for Environmental Policy and Planning, SUNY College of Environmental Science and Forestry.

Fisher, Peter F. 1991. "First Experiments in Viewshed Uncertainty: The Accuracy of the Viewshed Area." *Photogrammetric Engineering & Remote Sensing* 57 (10): 1321–27.

———. 1996. "Extending the Applicability of Viewsheds in Landscape Planning." *Photogrammetric Engineering & Remote Sensing* 62 (11): 1297–1302.

Fisher, Peter F., and Nicholas J. Tate. 2006. "Causes and Consequences of Error in Digital Elevation Models." *Progress in Physical Geography* 30 (4): 467–89.

Fu, Pinde, and Paul M. Rich. 2000. "The Solar Anlayst User Manual". Helios Environmental Modeling Institute.

Hengl, Tomislav. 2006. "Finding the Right Pixel Size." *Computers & Geosciences* 32 (9): 1283–98.

Hengl, Tomislav, and Ian S. Evans. 2009. "Mathematical and Digital Models of the Land Surface." In *Geomorphometry: Concepts, Software, Applications*, edited by Tomislav Hengl and Hannes I. Reuter, 31–63. Amsterdam: Elsevier.

Kim, Kamyoung, Alan T. Murray, and Ningchuan Xiao. 2008. "A Multiobjective Evolutionary Algorithm for Surveillance Sensor Placement." *Environment and Planning B: Planning and Design* 35: 935–48.

Lee, Jay, and Dan Stucky. 1998. "On Applying Viewshed Analysis for Determining Least-Cost Paths on Digital Elevation Models." *International Journal of Geographical Information Science* 12 (8).

Lu, Min, Jinfang Zhang, Pin Lv, and Zhihua Fan. 2008. "Least Visible Path Analysis in Raster Terrain." *International Journal of Geographical Information Science* 22 (6): 645–56.

Lynch, Kevin. 1976. *Managing the Sense of a Region*. Cambridge: MIT Press.

Maichak, Eric J., and Krysten L. Schuler. 2004. "Applicability of Viewshed Analysis to Wildlife Population Estimation." *American Midland Naturalist* 152 (2): 277–85.

Maloy, Mark A., and Denis J. Dean. 2001. "An Accuracy Assessment of Various GIS-Based Viewshed Delineation Techniques." *Photogrammetric Engineering and Remote Sensing* 67 (11): 1293–98.

Mees, Romain M. 1976. "Computer Evaluation of Existing and Proposed Fire Lookouts". General PSW-19. Pacific Southwest Forest and Range Experiment Station, Berkeley, CA: US Forest Service.

———. 1978. "Seen Areas and the Distribution of Fires about a Lookout". General Technical Report PSW-26. Pacific Southwest Forest and Range Experiment Station, Berkeley, CA: US Forest Service.

Morello, Eugenio, and Carlo Ratti. 2009. "A Digital Image of the City: 3D Isovists in Lynch's Urban Analysis." *Environment and Planning B: Planning and Design* 36 (5): 837–53.

Murray, Alan T., Kamyoung Kim, James W. Davis, Raghu Machiraju, and Richard Parent. 2007. "Coverage Optimization to Support Security Monitoring." *Computers, Environment and Urban Systems* 31: 133–47.

O'Sullivan, Patrick Edmund. 1983. *The Geography of Warfare*. New York: St. Martin's Press.

Ruggles, Clive L. N., David J. Medyckyj-Scott, and Alun Gruffydd. 1993. "Multiple Viewshed Analysis Using GIS and Its Archaeological Application: A Case Study in Northern Mull." In *Computing the Past*, edited by Andresen, Madsen, and Scollar, 125–32. Aarhus: Aarhus University Press.

Show, Stuart B., Edward I. Kotok, George M. Gowen, John R. Curry, and Arthur A. Brown. 1937. *Planning, Constructing, and Operating Forest-Fire Lookout Systems in California*. Circular No. 449. Washington, D.C.: U.S. Dept. of Agriculture.

Travis, Michael R., Gary H. Elsner, Wayne D. Iverson, and Christine G. Johnson. 1975. "VIEWIT: Computation of Seen Areas, Slope, and Aspect for Land-Use Planning". General Technical Report PSW-11. Pacific Southwest Forest and Range Experiment Station, Berkeley, CA: US Forest Service.

United States Geological Survey. 2013. “National Elevation Dataset”. Washington, D.C.: US Geological Survey. http://ned.usgs.gov/downloads.asp.

VanHorn, Jason E., and Nathan A. Mosurinjohn. 2010. “Urban 3D GIS Modeling of Terrorism Sniper Hazards.” *Social Science Computer Review* 28 (4): 482–96.

Wheatley, David. 1995. “Cumulative Viewshed Analysis: A GIS-Based Method for Investigating Intervisbility, and Its Archaeological Application.” In *Archaeology and Geographical Information Systems: A European Perspective*, edited by Gary R. Lock and Zoran Stančič, 171–85. London: Taylor & Francis.

Wilson, John P. 2012. “Digital Terrain Modeling.” *Geomorphology* 137 (1): 107–21.

APPENDIX A – GSSM PYTHON ALGORITHM SOURCE CODE

```
# Ground Station Siting Model [GSSM]
# Version 1.0.9 [24 July 2013] beta
# Created for the RESPT project

import numpy as np
import scipy
from scipy import spatial
import arcpy
from arcpy import *
import time
import math

start_time = time.time()

########################################
# INPUT VARIABLES FROM MODEL PARAMETERS
########################################
# Location of ESRI files
ObserverPoint = arcpy.GetParameterAsText(0)
arcpy.AddMessage("Observer Point: " + ObserverPoint)
arcpy.AddField_management(ObserverPoint, "PctVisible",
"FLOAT")

FlightPath = arcpy.GetParameterAsText(1)
arcpy.AddMessage("Flight Path: " + FlightPath)

rasterDEM = arcpy.GetParameterAsText(2)
arcpy.AddMessage("DEM Raster: " + rasterDEM)

OutputWorkspace = arcpy.GetParameterAsText(5)
arcpy.AddMessage("Output Location for Flight Path
Blocked/Visible Points: " + OutputWorkspace)
arcpy.env.workspace = OutputWorkspace

# Additional Parameters
FlightAltitude = arcpy.GetParameterAsText(3)
zFlight = float(FlightAltitude) * 0.3048 # elevation of
airplane, in meters
```

```
arcpy.AddMessage("Flight Altitude (in meters): " +
str(zFlight))

mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd)[0]

# find cell size of raster, assuming it's square
cellSizeProperty =
GetRasterProperties_management(rasterDEM, "CELLSIZEX")
cellSize = int(cellSizeProperty.getOutput(0))
arcpy.AddMessage("Raster Cell Size: " + str(cellSize))

Verbose = "false"
##arcpy.AddMessage("Verbose? " + Verbose)

outputBlockedPoints = arcpy.GetParameterAsText(4)
if str(outputBlockedPoints) == "true":
 arcpy.AddMessage(" Script will generate feature class and
layer of blocked points.")
if str(outputBlockedPoints) == "false":
 arcpy.AddMessage(" Script will only generate percentage
visibility from observer points.")

#########################################################
# DEM RASTER PROPERTY ANALYSIS, CONVERSION TO NUMPY ARRAY
#########################################################
# array of data converted from ESRI raster to NumPy array
x, y = np.mgrid[0:25:cellSize, 0:25:cellSize]
z = arcpy.RasterToNumPyArray(rasterDEM)

offsetXresult = GetRasterProperties_management(rasterDEM,
"LEFT")
offsetX = float(offsetXresult.getOutput(0))
offsetYresult = GetRasterProperties_management(rasterDEM,
"BOTTOM")
offsetY = float(offsetYresult.getOutput(0))
# calculate offset to shift geographic coordinates to array
locations
arrayRowMax = z.shape[0] - 1

if str(Verbose) == "true":
 arcpy.AddMessage("X Min: " + str(offsetX))
 arcpy.AddMessage("Y Min: " + str(offsetY))
 arcpy.AddMessage("Raster Height: " + str(arrayRowMax))

arcpy.AddMessage("========== script setup complete")
```

```
################
# OBSERVER POINT
################
# Determine location of observer point
obsvCursor = arcpy.UpdateCursor(ObserverPoint)
desc = arcpy.Describe(ObserverPoint)
shapefieldname = desc.ShapeFieldName

# XY location of observer on the ground
obsvIndex = 0
for obsv in obsvCursor:
 obsvFeature = obsv.getValue(shapefieldname)
 obsvPt = obsvFeature.getPart()
 # account for offset of geographic coordinate raster
 obsvX = math.floor((obsvPt.X - float(offsetX))/cellSize)
 obsvY = math.floor((obsvPt.Y - float(offsetY))/cellSize)
 if str(Verbose) == "true":
 arcpy.AddMessage("Value at Observer Location")
 arcpy.AddMessage(" >> geographic: x " + str(obsvPt.X) + ",
y " + str(obsvPt.Y))
 obsvCol = obsvX
 obsvRow = arrayRowMax - obsvY
 if str(Verbose) == "true":
 arcpy.AddMessage(" >> array: row " + str(obsvRow) + ",
column " + str(obsvCol))

 # elevation of observer [z]
 zObserver = z[obsvRow, obsvCol]
 if str(Verbose) == "true":
 arcpy.AddMessage(" >> elevation: " + str(zObserver) + "
meters")

 ############################
 # ESTABLISH FLIGHT PATH LINE
 ############################
 blockedSightLines = 0
 totalSightLines = 0
 obstructionList = []
 # Identify the geometry field
 desc = arcpy.Describe(FlightPath)
 shapefieldname = desc.ShapeFieldName
 # Create search cursor
 flightCursor = arcpy.SearchCursor(FlightPath)
 # Enter for loop for each feature/row
 for flightPt in flightCursor:
 # Create the geometry object
 feat = flightPt.getValue(shapefieldname)
```

```
 ## if str(Verbose) == "true":
 ## # the current line ID
 ## arcpy.AddMessage("Feature %i: " %
flightPt.getValue(desc.OIDFieldName))
 #Set start point
 startpt = feat.firstPoint
 #Set Start coordinates
 startX = math.floor((startpt.X - float(offsetX))/cellSize)
 startCol = startX
 startY = math.floor((startpt.Y - float(offsetY))/cellSize)
 startRow = arrayRowMax - startY
 #Set end point
 endpt = feat.lastPoint
 #Set End coordinates
 endX = math.floor((endpt.X - float(offsetX))/cellSize)
 endCol = endX
 endY = math.floor((endpt.Y - float(offsetY))/cellSize)
 endRow = arrayRowMax - endY

 zFP1 = z[startRow, startCol]
 zFP2 = z[endRow, endCol]
 if str(Verbose) == "true":
 arcpy.AddMessage("Value at Flight Start")
 arcpy.AddMessage(" >> geographic: x " + str(startpt.X) +
", y " + str(startpt.Y))
 arcpy.AddMessage(" >> array: row " + str(startRow) + ",
column " + str(startCol))
 arcpy.AddMessage(" >> " + str(zFP1))
 arcpy.AddMessage("Value at Flight End")
 arcpy.AddMessage(" >> geographic: x " + str(endpt.X) + ",
y " + str(endpt.Y))
 arcpy.AddMessage(" >> array: row " + str(endRow) + ",
column " + str(endCol))
 arcpy.AddMessage(" >> " + str(zFP2))

 # flight path begins at
 fpX0, fpY0 = startRow, startCol
 # flight path ends at
 fpX1, fpY1 = endRow, endCol

 arrayFlightPath = [[fpX0, fpY0], [fpX1, fpY1]]
 flightLength =
scipy.spatial.distance.pdist(arrayFlightPath, 'euclidean')
 numFPts = int(flightLength) / int(cellSize)

 # Make a line with "num" points distributed along it
```

```
 fpX, fpY = np.linspace(fpX0, fpX1, numFPts),
np.linspace(fpY0, fpY1, numFPts)
 ##fpLine = [np.linspace(fpX0, fpX1, numFPts),
np.linspace(fpY0, fpY1, numFPts)]

 arcpy.AddMessage("Flight path from " + str(fpX0) + ", " +
str(fpY0) + " to " + str(fpX1) + ", " + str(fpY1) + " at
altitude " + str(zFlight))
 arcpy.AddMessage(" >> length: " + str(flightLength) + "
meters")
 arcpy.AddMessage(" >> points: " + str(numFPts))

 ############################################################
 # CONNECT EACH FLIGHT POINT TO THE OBSERVER POINT, ANALYZE
 ############################################################
 dzMax = zFlight - zObserver # difference between flight
altitude and observer
 for indexFP, point in np.ndenumerate(fpX):
 if str(Verbose) == "true":
 arcpy.AddMessage("flight point row/col: " +
str(fpX[indexFP]) + ", " + str(fpY[indexFP]))
 ##arcpy.AddMessage("observer point row/col: " +
str(obsvRow) + ", " + str(obsvCol))
 flightX = fpX[indexFP]
 flightY = fpY[indexFP]
 arraySightLine = [[obsvRow, obsvCol], [flightX, flightY]]
 sightLength = scipy.spatial.distance.pdist(arraySightLine,
'euclidean')
 numSPts = int(sightLength)
 if str(Verbose) == "true":
 arcpy.AddMessage(" >>> length: " + str(sightLength) + ",
number of points: " + str(numSPts))

 slX, slY = np.linspace(obsvRow, flightX, numSPts),
np.linspace(obsvCol, flightY, numSPts)
 # drop 1st values, since observer point doesn't need to be
evaluated
 slX = slX[1:]
 slY = slY[1:]
 ## arcpy.AddMessage(" >>> X Values along Sight Line: " +
slX)

 dMax = sightLength
 ratio = dzMax / dMax # tangent of observer sight line
vertical angle
 if str(Verbose) == "true":
```

```
 arcpy.AddMessage(" >>> dMax: " + str(dMax) + ", dzMax: " +
str(dzMax) + ", Ratio: " + str(ratio))

 zActual = z[slX.astype(np.int), slY.astype(np.int)]

 blockedPoints = 0
 for indexSL, pointSL in np.ndenumerate(slX):
 sightX = slX[indexSL]
 sightY = slY[indexSL]
 arrayTempLine = [[obsvRow, obsvCol], [sightX, sightY]]
 dPoint = scipy.spatial.distance.pdist(arrayTempLine,
'euclidean')
 zAllowed = zObserver + (dPoint * ratio)
 zPoint = zActual[indexSL]
 if str(Verbose) == "true":
 arcpy.AddMessage("slX " + "%.2f" % slX[indexSL] + ", slY "
+ "%.2f" % slY[indexSL] + " // dist " + "%.2f" % dPoint +
", zAllowed " + "%.2f" % zAllowed + ", zPoint " +
str(zPoint))
 if zPoint > zAllowed:
 if str(Verbose) == "true":
 arcpy.AddMessage(" >> OBSTRUCTION <<")
 blockedPoints += 1
 if str(outputBlockedPoints) == "true":
 BlockRow = flightX
 BlockCol = flightY
 BlockPtX = (BlockCol * cellSize) + float(offsetX)
 BlockPtY = ((arrayRowMax - BlockRow) * cellSize) +
float(offsetY)
 CurrentPoint = [BlockPtX, BlockPtY]
 obstructionList.append(CurrentPoint)
 if blockedPoints > 0:
 if str(Verbose) == "true":
 arcpy.AddMessage(" >> 1 Sight Line with " +
str(blockedPoints) + " Blocked Points")
 blockedSightLines += 1
 totalSightLines += 1
 if str(Verbose) == "true":
 arcpy.AddMessage(" ========= =========")

 # time result
 arcpy.AddMessage(" >> analyzed in " + str(time.time() -
start_time) + " seconds")
 obsvIndex = obsvIndex + 1

 ##########################
 # SHARE RESULTS WITH USER
```

```
 ##########################
 percentVisible = 1. - (float(blockedSightLines) /
float(totalSightLines))
 arcpy.AddMessage("========== observer point " +
str(obsvIndex) + " analysis finished")
 arcpy.AddMessage(" >> Sight Lines Analyzed: " +
str(totalSightLines))
 arcpy.AddMessage(" >> Sight Lines Blocked: " +
str(blockedSightLines))
 arcpy.AddMessage("FINAL RESULT: flight path is %" +
str(100 * percentVisible) + " visible.")

 obsv.PctVisible = percentVisible
 obsvCursor.updateRow(obsv)
 arcpy.AddMessage(" >> Observer Point Attribute Table
Updated")

 if str(Verbose) == "true":
 arcpy.AddMessage("Points causing obstruction: ")
 obstructionPt = arcpy.Point()
 obstructionGeom = []
 for Point in obstructionList:
 if str(Verbose) == "true":
 arcpy.AddMessage(" >> x: " + "%.2f" % Point[0] + ", y: " +
"%.2f" % Point[1])
 obstructionPt.X = Point[0]
 obstructionPt.Y = Point[1]
 obstructionGeom.append(arcpy.PointGeometry(obstructionPt))

 if str(outputBlockedPoints) == "true" and percentVisible <
1:
 ##################################################
 # PUSH LIST OF BLOCKING POINTS INTO A UNIQUE LAYER
 ##################################################
 CurrentDateTime =
datetime.datetime.now().strftime("%Y%m%d%H%M")
 FlightPathPoints = "FlightPath" + CurrentDateTime +
str(obsvIndex)

 arcpy.CopyFeatures_management(obstructionGeom,
FlightPathPoints)

 arcpy.env.overwriteOutput = True
 lyrName = "BlockedFlightPoints_Obsv" + str(obsvIndex)
 arcpy.MakeFeatureLayer_management(FlightPathPoints,
lyrName)
 arcpy.AddMessage(" >> FlightPathPoints layer created")
```

```
 lyrFile = arcpy.mapping.Layer(lyrName)
 arcpy.mapping.AddLayer(df, lyrFile)
 arcpy.RefreshActiveView()
 arcpy.AddMessage(" >> FlightPathPoints added to map")

 arcpy.AddMessage("========== ========= =========
=========")

del obsv
del obsvCursor
del flightPt
del flightCursor

obsvLayer = arcpy.mapping.ListLayers(mxd, ObserverPoint)[0]
#Indexing list for 1st layer
if obsvLayer.supports("LABELCLASSES"):
 for lblClass in obsvLayer.labelClasses:
 lblClass.showClassLabels = True
 lblClass.className = "PctVisible"
obsvLayer.showLabels = True
arcpy.AddMessage(" >> Observer Point Visibility Attribute
Label Added")

# time result
arcpy.AddMessage("========== ========= =========
=========")
arcpy.AddMessage(" time elapsed: " + str(time.time() -
start_time) + " seconds")
```

APPENDIX B – SIGHT LINES METHODOLOGY AND RESULTS

A number of efficiency tests were completed on the Sight Lines model before it was determined to be incapable of modeling ground-to-air visibility. The model and its behavior are therefore included in this appendix. In large part, the results support the conclusion that researchers should treat visibility analysis results with skepticism and caution.

The Sight Lines tools require a more complex model than the Viewshed tool. The desired output – a single ratio value to describe target visibility – requires that the visibility of the target point along each individual sight line be summarized, and the data then appended back to the original observer dataset. The Sight Lines model has to iterate through once for each observer feature in order to keep the data organized. The model is consequently more complex and runs through many more operations than the Viewshed model (see Figure B.1), although the additional tools are largely organizational and therefore not computationally intensive.

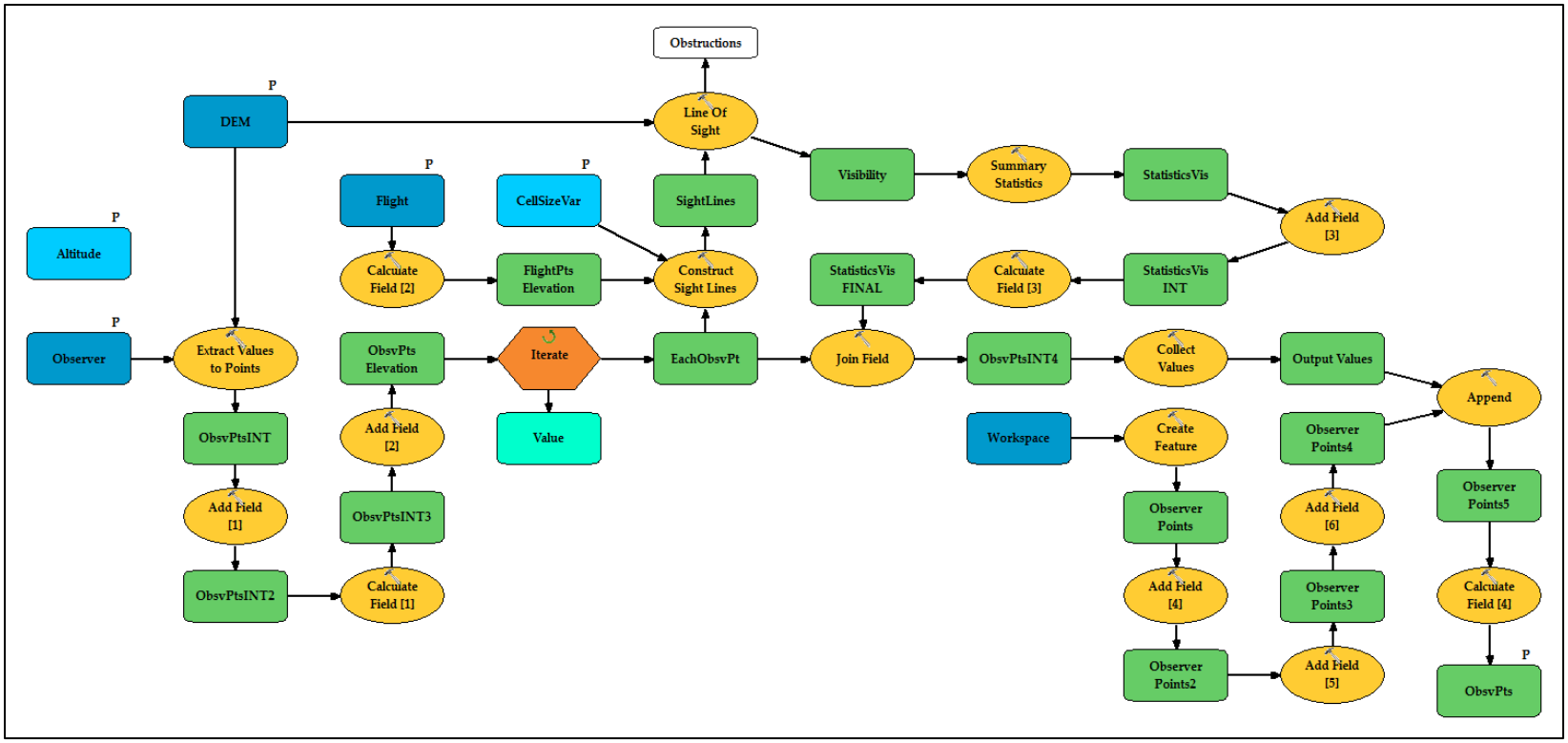

Figure B.1: ModelBuilder diagram of the model based on the Line of Sight tool.

The Sight Lines model was approximately as efficient as the Python algorithm, in general. There was one notable exception. When a large number of observer points were tested, the Python algorithm finished in approximately two hours. The Sight Lines model crashed after analyzing 263 points in 4.25 hours. At that rate, it would have taken nearly a week to finish its analysis. This indicates a much less efficient use of memory and computational resources than the Python algorithm. It was, however, capable of operating on a fine-resolution DSM (1m × 1m) without memory errors, as long as the number of observer points was small.

Sensitivity analysis results indicated that the Python algorithm and Sight Lines model were affected by the same input parameters, although in moderately different ways. The number of cells in the surface raster had no influence on the execution time of the Sight Lines model. The quantity of observer points was strongly correlated ($R^2 = 0.9993$) to the Sight Lines model performance (see Figure B.2). The influence was much stronger on the Sight Lines model than the Python algorithm, with the execution time increasing at nearly twice the rate of the Python algorithm.

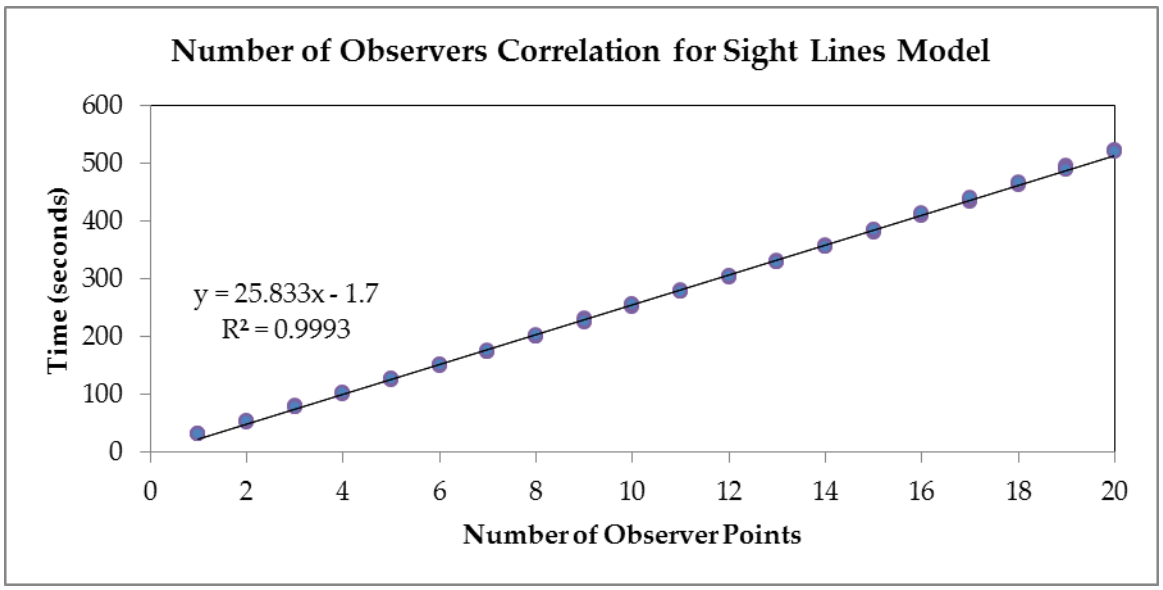

Figure B.2: Correlation between number of antenna sites and Sight Lines model execution time.

As with the Viewshed model and Python algorithm, the flight path length was strongly correlated ($R^2 = 0.9904$) to the execution time. Extending the flight path added less time per point to the Sight Lines model (see Figure B.3).

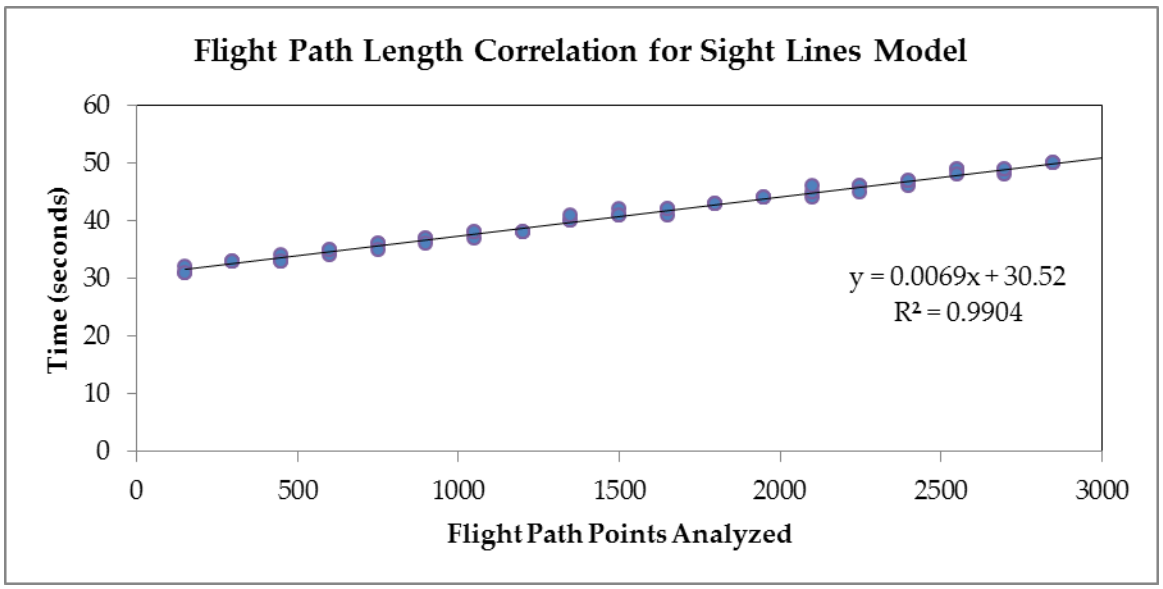

Figure B.3: Correlation between flight path length and Sight Lines model execution time.

The distance from the observer to the flight path was also strongly correlated to execution time in the Sight Lines model (see Figure B.4). This relationship was particularly interesting for the Sight Lines model. For observer distances of 2,000 cells or less (linear correlation of $R^2 = 0.9891$), the execution time did not increase as rapidly as for distances greater than 2,000 cells (linear correlation of $R^2 = 0.9995$). There were two very distinct correlations for the two subsets of data.

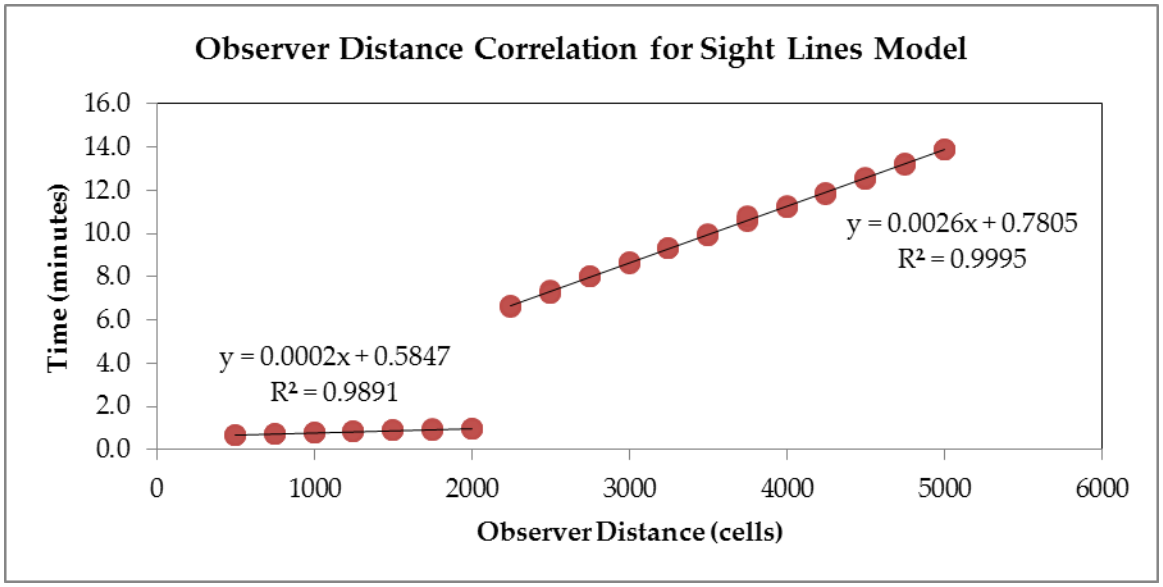

Figure B.4: Two different correlations between antenna position and Sight Lines model execution time.

The cause of this unusual shift in behavior is unclear, but is presumably due to a difference in the Sight Lines tool calculation method between "close" and "far" observers. Since the underlying algorithm is proprietary, the causes cannot be more clearly explained. Additionally, the relationship between

distance and time was the same for the Python algorithm and the "far" distances subset of the Sight Lines model, implying that the two methods use similar (or identical) basic calculations. The potential for unknown factors in algorithm design to influence results is clearly demonstrated in this shift in behavior. It can be assumed that there is a different calculation being performed for the two different regimes. If this influences the execution time, it also has the potential to influence the accuracy of results. Without understanding the underlying modeling process, it is impossible for researchers to adequately explore the causes and estimate the risk of increased error.

APPENDIX C – OPTIMIZING THE PYTHON ALGORITHM

C.1 BACKGROUND AND PERFORMANCE

A Ground-to-Air Visibility (GTAV) Python algorithm has also been written that seems to be both more efficient and more accurate than the GSSM Python algorithm described and tested in the main body of this thesis. These results are preliminary, but show exciting potential for further refinement of ground-to-air visibility modeling using sight lines.

Instead of calculating the tangent ratio at each point along the sight line, this algorithm takes advantage of arrays to calculate all the values in one step. The sight line analysis is then simpler, since it only needs to collect the existing ratios and compare the maximum ratio per each sight line to the tangent ratio of the target. If any the maximum tangent ratio of a point along the sight line exceeds the tangent ratio of the target, then the target is not visible. The general mathematical principles are the same as the original GSSM algorithm, but are calculated and manipulated in a more efficient fashion. Preliminary exploration indicates this algorithm performs faster than the GSSM, as seen in Table C.1. The sensitivity to inputs of the GTAV algorithm is similar to the GSSM algorithm, with increased execution time from increased observer points, length of flight

paths, and distance between observers and flight paths. However, the execution time does not increase as rapidly from each of the variations, and the GTAV algorithm is still well under the one hour threshold even in the most complex scenario.

Table C.1: Comparison of GSSM and GTAV performance.

Raster Resolution	Flight Path Length (m)	Observers Included	GSSM Run Time (minutes)	GTAV Run Time (minutes)
30m × 30m	6,000	5	0.73	1.08
	10,000	5	2.20	1.15
	16,000	5	2.89	1.17
10m × 10m	6,000	3	4.19	2.46
		5	6.02	3.96
	10,000	3	12.53	2.96
		5	19.30	4.92
	16,000	3	16.55	3.28
		5	25.17	5.21
3m × 3m	6,000	3	45.09	4.19
		5	65.47	6.04
	10,000	3	138.22	9.81
		5	214.36	15.27
	16,000	3	187.44	12.49
		5	279.61	19.49

The accuracy of the GTAV algorithm was not tested in this research, but it is approximately equivalent. It uses a GIS function (and therefore geographic coordinates) to calculate distance from the observer point. Since the GSSM

algorithm translates the elevation raster into a local array coordinate system before calculating distance between points, this introduced a potential for distortion, particularly as the observer and target point separation passes the point where Euclidean distance calculations begin to poorly represent actual distance. It is possible that the GTAV algorithm output is a better representation of real-world visibility as a result, although much more extensive testing is needed to support that conclusion.

The GTAV algorithm still encounters the same problems with memory usage noted in the GSSM testing, and can only be used with raster DSM data up to a certain size. A further reconsideration of the data storage and access methods is warranted. Much like the Viewshed model, which analyzed a full visibility surface to capture only a few points of data, these Python algorithms are still analyzing a full raster surface to explore a much more spatially limited set of data.

C.2 SOURCE CODE

```
# Ground-to-Air Visibility Algorithm

import numpy as np
import scipy
from scipy import spatial
import arcpy
from arcpy import *
from arcpy.sa import *
import time
import math
```

```
start_time = time.time()

# Location of data
ObserverPoint = arcpy.GetParameterAsText(0)
arcpy.AddField_management(ObserverPoint, 'PctVisible',
'FLOAT')

FlightPath = arcpy.GetParameterAsText(1)
rasterDEM = arcpy.GetParameterAsText(2)

# Additional Parameters
FlightAltitude = arcpy.GetParameterAsText(3)
zFlight = float(FlightAltitude) * 0.3048 # elevation of
airplane, in meters

mxd = arcpy.mapping.MapDocument('CURRENT')
df = arcpy.mapping.ListDataFrames(mxd)[0]

# find cell size of raster, assuming it's square
cellSizeProperty =
GetRasterProperties_management(rasterDEM, 'CELLSIZEX')
cellSize = float(cellSizeProperty.getOutput(0))

# Determine location of observer point
obsvCursor = arcpy.UpdateCursor(ObserverPoint)
desc = arcpy.Describe(ObserverPoint)
shapefieldname = desc.ShapeFieldName

# XY location of observer on the ground
obsvIndex = 0
for obsv in obsvCursor:
    obsvFeature = obsv.getValue(shapefieldname)
    obsvPt = obsvFeature.getPart()

    # ArcPy euclidean distance tool
    rasterDistance = EucDistance(obsvFeature, "", cellSize)

    # array of data converted from raster to NumPy array
    x, y = np.mgrid[0:25:cellSize, 0:25:cellSize]

    arrayZ = arcpy.RasterToNumPyArray(rasterDEM)

    offsetXresult =
GetRasterProperties_management(rasterDEM, 'LEFT')
    offsetX = float(offsetXresult.getOutput(0))
    offsetYresult =
GetRasterProperties_management(rasterDEM, 'BOTTOM')
```

```
    offsetY = float(offsetYresult.getOutput(0))
    # calculate offset to shift geographic coordinates to
array locations
    arrayRowMax = arrayZ.shape[0] - 1

    # account for offset of geographic coordinate raster
    obsvX = math.floor((obsvPt.X -
float(offsetX))/cellSize)
    obsvY = math.floor((obsvPt.Y -
float(offsetY))/cellSize)
    obsvCol = obsvX
    obsvRow = arrayRowMax - obsvY

    # elevation of observer [z]
    zObserver = arrayZ[obsvRow, obsvCol]

    # Raster calculator, angle = arctan (z / d)
    arrayDistance =
arcpy.RasterToNumPyArray(rasterDistance)
    arrayCellDistance = arrayDistance / cellSize
    arrayCorrectedZ = arrayZ - zObserver
    arrayTangent = arrayCorrectedZ / arrayCellDistance

    blockedSightLines = 0
    totalSightLines = 0
    obstructionList = []
    # Identify the geometry field
    desc = arcpy.Describe(FlightPath)
    shapefieldname = desc.ShapeFieldName
    # Create search cursor
    flightCursor = arcpy.SearchCursor(FlightPath)
    # Enter for loop for each feature/row
    for flightPt in flightCursor:
        # Create the geometry object
        feat = flightPt.getValue(shapefieldname)
        #Set start point
        startpt = feat.firstPoint
        #Set Start coordinates
        startX = math.floor((startpt.X -
float(offsetX))/cellSize)
        startCol = startX
        startY = math.floor((startpt.Y -
float(offsetY))/cellSize)
        startRow = arrayRowMax - startY
        #Set end point
        endpt = feat.lastPoint
        #Set End coordinates
```

```
        endX = math.floor((endpt.X -
float(offsetX))/cellSize)
        endCol = endX
        endY = math.floor((endpt.Y -
float(offsetY))/cellSize)
        endRow = arrayRowMax - endY

        zFP1 = arrayZ[startRow, startCol]
        zFP2 = arrayZ[endRow, endCol]

        # flight path begins at
        fpX0, fpY0 = startRow, startCol
        # flight path ends at
        fpX1, fpY1 = endRow, endCol

        arrayFlightPath = [[fpX0, fpY0], [fpX1, fpY1]]
        flightLength =
scipy.spatial.distance.pdist(arrayFlightPath, 'euclidean')
        numFPts = int(flightLength)# / int(cellSize)

        # Make a line with 'num' points distributed along
it
        fpX, fpY = np.linspace(fpX0, fpX1, numFPts),
np.linspace(fpY0, fpY1, numFPts)

        dzMax = zFlight - zObserver # difference between
flight altitude and observer
        for indexFP, point in np.ndenumerate(fpX):
            flightX = fpX[indexFP]
            flightY = fpY[indexFP]
            arraySightLine = [[obsvRow, obsvCol], [flightX,
flightY]]
            sightLength =
scipy.spatial.distance.pdist(arraySightLine, 'euclidean')
            numSPts = int(sightLength)

            slX, slY = np.linspace(obsvRow, flightX,
numSPts), np.linspace(obsvCol, flightY, numSPts)
            # drop first values so observer point will not
be evaluated
            slX = slX[1:]
            slY = slY[1:]

            dMax = sightLength
            tangentFlightPoint = dzMax / dMax # tangent of
observer sight line vertical angle
```

```
            tangentActual =
arrayTangent[slX.astype(np.int), slY.astype(np.int)]

            listAngles = []
            blockedPoints = 0
            for indexSL, pointSL in np.ndenumerate(slX):
                sightX = slX[indexSL]
                sightY = slY[indexSL]
                arrayTempLine = [[obsvRow, obsvCol],
[sightX, sightY]]
                tangentPoint = tangentActual[indexSL]
                listAngles.append(tangentPoint)
            if max(listAngles) > tangentFlightPoint:
                blockedSightLines += 1
            totalSightLines += 1

        # time result
        arcpy.AddMessage(' >> analyzed in ' +
str(time.time() - start_time) + ' seconds')
    obsvIndex = obsvIndex + 1

    percentVisible = 1. - (float(blockedSightLines) /
float(totalSightLines))
    arcpy.AddMessage('========== observer point ' +
str(obsvIndex) + ' analysis finished')
    arcpy.AddMessage('FINAL RESULT: flight path is %' +
str(100 * percentVisible) + ' visible.')

    obsv.PctVisible = percentVisible
    obsvCursor.updateRow(obsv)

del obsv, obsvCursor, flightPt, flightCursor

# time result
arcpy.AddMessage(' time elapsed: {}
seconds'.format(str(time.time() - start_time)))
```