Border Patrol

Shingo Murata
Swarthmore College
Swarthmore, PA
19081
smurata1@cs.swarthmore.edu

Dan Amato
Swarthmore College
Swarthmore, PA
19081
damato1@cs.swarthmore.edu

Abstract

We implement a border patrol program that computes ideal locations for observation towers overlooking a border of interest. In this particular project, we study the border of Arizona facing Mexico. We use GRASS to manipulate elevation raster data. Our algorithm extracts the border from a raster file and locates candidate positions for observation towers along that border. The viewshed of each observation tower is computed with a direct line of sight algorithm. We employ a tower placement algorithm to select only the necessary towers from the set of all candidate towers. Our algorithm selected 92 towers within a 5km zone from the border to patrol the approximately 680km border.

1 Introduction

Border patrol is a common interest involved in national security. Militaries are frequently concerned with detecting threats along the extent of a particular border as completely and efficiently as possible. It is important that border security can be established cost-effectively as well. We model this problem as the task of placing the minimum number of towers necessary to view the entire border of interest within some range of that border.

Current GIS technology makes it possible to automate the process of planning optimal locations for border observation towers. In this paper we develop algorithms to automatically extract a border from a raster file, find candidate tower locations near the border, and select the optimal set of towers that is capable of observing the entire border.

Elevation data for many areas of interest are readily available on the Internet. We use the Geographic Resources Analysis Support System (GRASS) for data manipulation. GRASS is suited for this project because it has full functionality in visualizing elevation, data conversion, and I/O support for ascii files that are compatible with our C programs. With an abundance of digital elevation models in raster format and the software tools to manipulate them, we can successfully apply useful viewshed computation algorithms to the problem of finding the optimal set of observation towers.

The viewshed of a view point is the set of points in the terrain model that can be observed from that point. Viewshed computation is central to the automation of these algorithms because it is essential to know which border points a candidate tower is capable of observing. Our tower placement algorithm involves the iterative selection of candidate towers with the greatest contribution of yet unseen border points to the current optimal set of observation towers.

There are several parameters involved in tower placement. One of our goals is to minimize the number of towers we need to observe the entire border. This number is dependant both on the height of the towers and the maximum distance they are allowed to be placed from the border. As the height of the towers is increased, the number of necessary towers decreases, but the cost of each tower increases.

Finding a cost-minimizing solution would involve balancing these factors. Increasing the distance from the border in which towers can be placed may incorporate useful elevation maxima into the model that were previously out of range. The gains from this are limited by atmospheric conditions that limit visibility and, ultimately, by the curvature of the Earth.

We begin by reviewing the viewshed algorithm presented by David Izraelevitz in (Izraelevitz, 2003). We then describe in detail the three major steps in our project: border extraction, viewshed computation, and tower placement. Finally, we present the results in section 4.

2 Related Work

Several algorithms for computing the viewshed of a point are presented in (Izraelevitz, 2003). The first method presented is direct computation. This method essentially checks each possible obstruction on a line from the view point to the target point. If there are no obstructions along this line, then the target point is considered visible. This algorithm is straight forward, but is computationally inefficient, because it requires $O(n)$ computations for each grid point on an n x n field, resulting in an $O(n^3)$ algorithm.

The Xdraw algorithm employs the Line of Sight (LOS) function to compute the viewshed of a view point. This algorithm is faster than the direct method because it stores previous results that can be utilized at the next stage of computation along the same line.

The final algorithm improves the Xdraw algorithm by introducing a backtracking method to reduce the number of interpolations and increase the accuracy of the LOS calculations. If any point along the line between the view point and the target point coincides with a grid data point, that data point is used to initialize the LOS computation. Otherwise, the algorithm backtracks a specified distance and initializes the computation with an interpolated LOS value.

To determine border visibility, we do not need to compute the entire viewshed from a given view point. We only need to determine whether target points on the border are visible from a tower's view point. It is irrelevant to know whether the points between the observation tower and the border are visible. This negates the benefits of the Xdraw algorithm which are based on a fast incremental computation. We implement and use the direct algorithm because its inefficient computation is mitigated due to the limited number of points we are examining. In addition, its accuracy is superior to Xdraw.

Vincent presents an alternative viewshed algorithm based on ray casting in three dimensional polygonal models in (Vincent, 1999). The algorithm is based on a hierarchical partitioning of three dimensional space. This partition is represented by a k-d tree. The hierarchical partitioning of the space continues until the model of the terrain is enclosed in appropriate bounding boxes. This hierarchical structuring of the space allows Vincent to increase the search speed for view obstructions. Other optimizations include a backface culling mechanism to remove irrelevant surface polygons from the search space and a parallelization of the algorithm. Vincent also sets up an error vs. speed tradeoff by adaptively spacing the rays used to test for intersection. As the number of rays increases, accuracy improves at the cost of increased execution time.

3 Methods

Our automated border patrol algorithm has three main phases. First, the relevant border must be extracted from the data. Second, the set of candidate towers must be determined, and the viewshed of these towers must be computed. Finally, the tower placement algorithm must select the optimal set of observation towers from the entire set of candidate towers. These selections are based on the visibility characteristics of the candidate towers. These steps are elaborated below.

3.1 Border Extraction

To patrol the border, we first need the coordinates of each point on the border. It is not trivial because the digital elevation model contains no information regarding the border. Since we are looking at the Arizona border for this particular experiment, and Arizona has a straight border facing Mexico, we could have figured out the equation for the line. However, this method would severely restrict the type of border we can patrol. Therefore, we present an algorithm to extract a border in general.

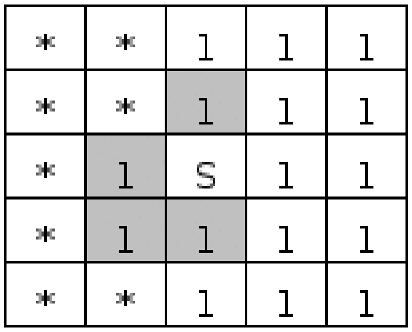


Figure 1: Start List Construction

Although we have no information about the border in the raster file, we do have a vector file that traces the border. We begin by converting the vector file into raster format using GRASS. Raster cells on the interior of the border are marked with a 1, while raster cells exterior to the border are marked with a NULL flag. We then output the raster to an ascii file, giving us a file of 1's and NULL's. Our algorithm defines a *border candidate* to be a point whose own value is 1, and also has at least one neighbor that is NULL.

Since we are working with a section of the border, we need to specify the start coordinate and the end coordinate. The algorithm checks the 8 neighbors of the start point. If any of them are border candidates, the points are added to the *start list*. It also marks all 8 neighbors as "visited". Figure 1 displays the state of the algorithm after the start list has been created. The grey cells are those that have been added to the start list.

For each point in the start list, our algorithm works as follows:

1. Dequeue a point from the start list. Call it p.

2. Look at the 8 neighbors of p, and find border candidates that have *not* been marked as "visited" yet. If there are any such points, add them to a work list. The work list is implemented as a linked-list queue, as is our temporary "border" described below.

3. Dequeue the first point from the work list and mark it as "visited." Add the point to the temporary border linked list. Call this point p and then repeat step 2. Continue until the work list becomes empty or we hit the "end" point.

4. If the work list has become empty without reaching the end, we scrap that temporary border. If we hit the end point, we have detected a potential border. In either case, we restart the process from step 1 with another start point, until we exhaust all the start point possibilities.

Once all start points have been expanded, we may have two valid borders: one for each of the two directions along the border that lead from the start point to the end point. We must determine which of the two potential borders is the real border of interest, and which is simply the remaining border of the region which excludes the border of interest. For now, we make this decision by simply choosing the shorter border.

We then copy the linked list into an array, as arrays are easier to work with for our purpose. At this point, each border cell all has height of 1 since they were extracted from the 1 or NULL raster. It is important that we go through the actual elevation raster and set the elevation of these border points to their true values.

At this point we have extracted the border between the start point and the end point. We also define the *zone* in which towers may be placed. This zone is a band of specified width along the extracted border and inside of the "home" territory. To find the points falling in this zone, we essentially perform a breadth first search of the 1 or NULL raster. We begin with the extracted border points, and add them into a queue. While the queue is not empty and the specified width has not been reached, we dequeue a point from the queue. This point is flagged as "visited," and all of its unvisited, home territory neighbors are added to the queue. All of the points in the zone are stored in an array for future use.

3.2 Viewshed Computation

The viewshed of a view point in a terrain model is the set of all points visible from that point. To patrol the border we do not need to compute the full viewshed of a tower location. We only need to compute the visibility of points on the border. If a set of towers can collectively view each border point, then the border is considered to be fully patrolled by that set.

As we are not interested in computing the full viewshed of potential tower points, but only the visibility of the border points, we do not need the optimized approximation algorithms presented in (Izraelevitz, 2003). We can afford to simply compute the visibility from the potential tower point to each target border point. The relevant points in the viewshed computation are the view point, p_v, the target point, p_t, and the point of potential view obstruction, p. The visibility of p_t from p_v can be determined with equation 1.

$$elv(p_t) > elv(p_v) + \frac{|p_t - p_v|}{|p - p_v|}(S(p) - elv(p_v)) \quad (1)$$

$elv(p_a)$ denotes the elevation of point p_a. $|p_a - p_b|$ represents the Euclidean distance between two points, p_a and p_b. Finally, $S(p_a)$ is the interpolated elevation of the terrain model at the (x, y) coordinates of p_a.

This inequality places a constraint on the height of the target point, p_t, based on the elevation data along a sight line between the view point and the target point. The inequality generates the minimum elevation at the target location that is visible from the view point given the elevation of the potential obstruction at point p. If the actual target elevation is less than or equal to this minimum, then the target point is not visible from the view point because the sight line is obstructed by the obstacle at point p.

To determine the visibility of the target point, equation 1 must be evaluated at each point, p, along the sight line between the view point and the target point. If no point along the sight line obstructs the view, then the target is visible. We approximate this test by evaluating equation 1 at regular intervals along the sight line between the view point and the target point. The theoretical sight line will fall between two grid data points in general. The elevation at point p is the interpolated elevation of the two grid data points on either side of the sight line.

The interpolated elevation at point p and the elevation of the target point are adjusted for the Earth's curvature prior to the computation of equation 1. The curvature adjustment of a point p_a's elevation, $\Delta_{elv(p_a)}$, is given by equation 2, where d represents the quantity $|p_a - p_v|$. The geometry of the approximation is illustrated in figure 2.

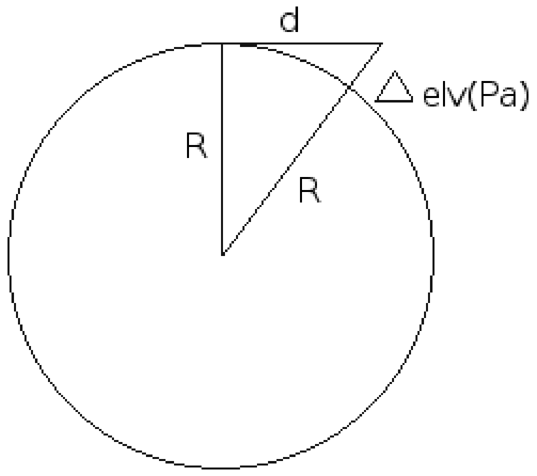


Figure 2: Geometry of the Earth Curvature Approximation

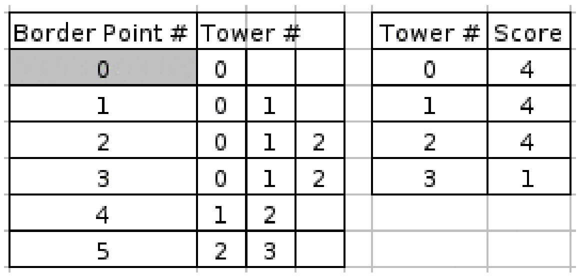
$$\Delta_{elv(p_a)} = \sqrt{{R_{Earth}}^2 + d^2} - R_{Earth} \quad (2)$$

We use this viewshed computation method to compute the visibility of each border point from a potential tower location.

3.3 Placement Algorithm

Our goal is to view the entire border with as few towers as possible. As noted in section 3.1, there is a zone of specified width back from the border in which towers may be placed. We base our tower placement on the principle that an observer can see more if s/he is at a higher elevation. We begin by locating all of the local maxima with in the zone. A local maximum is defined as a raster cell having no higher neighbors within the zone. These maxima serve as the candidate positions of our towers.

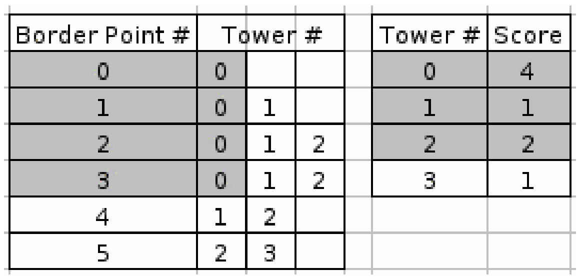
We first figure out which candidate towers are absolutely necessary. A candidate is considered necessary if there are points on the border that can only by seen by that candidate position. Therefore, instead of computing which points each tower would be able to see, we compute for each point on the border, which towers can see it (although they are computationally equivalent).



Border Point #	Tower #		
0	0		
1	0	1	
2	0	1	2
3	0	1	2
4	1	2	
5	2	3	

Tower #	Score
0	4
1	4
2	4
3	1

Figure 3: Example of border visibility state



Border Point #	Tower #		
0	0		
1	0	1	
2	0	1	2
3	0	1	2
4	1	2	
5	2	3	

Tower #	Score
0	4
1	1
2	2
3	1

Figure 4: Example of updated border visibility state

We maintain a table of integers, with each row corresponding to a point on the border. In each column, we keep track of a tower identifier that can see the point. For each point on the border, we test whether that point can be seen by each of the candidate positions. If the point is visible from a candidate tower location, we add the identifier for that tower to the point's row in the table.

Our table may look like figure 3, which shows that the first point on the border can be seen by Tower 0, the second by Tower 0 and 1, the third by Tower 0, 1 and 2, and so on. Every time a tower is added to the table, we increment the "score" of that candidate tower by 1. The score corresponds to the number of points the candidate tower could see if it were built.

After we have created the table, we traverse the table to find the border points visible from only one tower. We know that the single towers that can see these points are critical. In the table above, for example, Tower 0 is critical because it is the only tower that can see the first point of the border.

Every time we find a critical tower, we add it to the permanent tower list. Then we traverse the integer table to "close" all border points that can be seen by that tower. We traverse the list, find out which points that tower can see, and, flag that point as "secured." We also traverse the row for the secured point, and if any other towers can see it, we *decrement* the score of those towers. By decrementing the scores of these towers, we ensure that the score represents only the number of unsecured points viewable from the candidate tower. This prevents redundant towers which can view few new border points from being selected as good candidates in future iterations of the algorithm. The effect of adding Tower 0 to the permanent list in our example above is illustrated in figure 4. The score of Tower 1 is decremented by three because three of the border points it can view are made redundant with the addition of Tower 0 to the permanent list.

Every time we add a tower to the list, we check to see if the entire border has been "secured" yet. The chances are, the border cannot be secured by just those critical towers. So we move on to the second phase of the placement algorithm, in which we simply pick the candidate tower with the highest score out of the remaining towers (when we put a tower into the permanent tower list, we set its score to -1, so we never look at it again). When we add the tower, we "close" the points it can see as before. We repeat this process until the entire border is secured, or there are no more candidate towers to consider.

4 Results

The input raster, which covers the southern half of Arizona, has 2846 x 5705 data points at 100m resolution. When we extract the border, we find that it has 6768 data points. Out of the 6768 points, 1332 are local maxima, so we begin with 1332 candidate points for the towers.

We begin by restricting tower placement to points directly on the border. When we scan for points visible from only one tower, we find that only 6 towers are critical. With those 6 critical towers, we can view 844 / 6768 border points. The next tower we place, the tower with the highest score, can view 619 points by itself. After securing 6642 / 6769 points, our additional towers can only see 5 novel points. We also need 15 towers that can only see 1 unique point. In total, we need 120 towers.

When we expand the zone, we have many more points to work with. Within a 5km zone from the border, we have more than 370,000 points, from which we obtain 8974 maxima. While we have

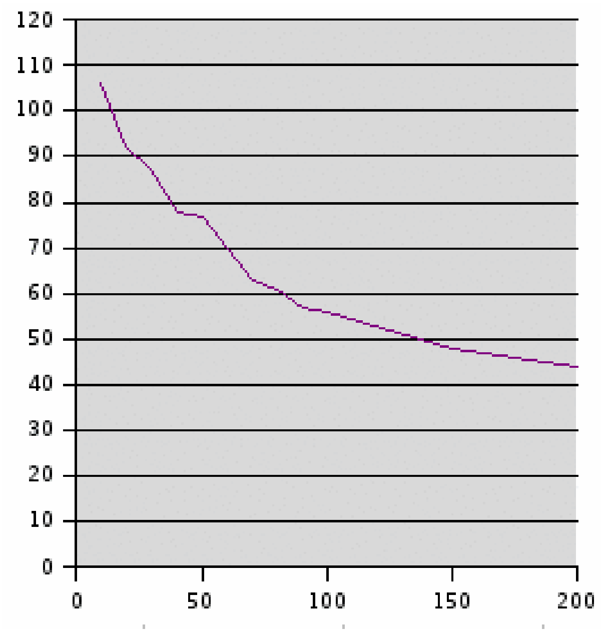


Figure 7: Number of Towers vs. Height of Towers (meters)

about 50 times as many total data points, we only have about 7 times as many maxima because we are now checking all 8 neighbors to test if a point is a maximum, as opposed to just checking the 2 adjacent points on the border. For 20m towers, by expanding the border zone to 5km wide we decreased the number of necessary towers to 92.

5 Discussion

Considering that the border is approximately 680km long, the number of towers we need is relatively low. With 20m towers in a 5km border zone, we need 92 towers. On average, each tower is responsible for roughly 70 data points, or 7km of the border. The number of towers seems to be higher than it could be due to the clusters that we can observe in figure 6. The clusters suggest that our algorithm may not be suited to certain geographical features found in these regions.

Still, we can infer some useful information from the data. We expected the number of towers required to decrease as we increased the tower height, but as we can see from figure 7, it seems to be asymptotic. In fact, when we tested with kilometer-high towers (which is absurdly high), we found that we would still need 23 towers. What it tells us is that after

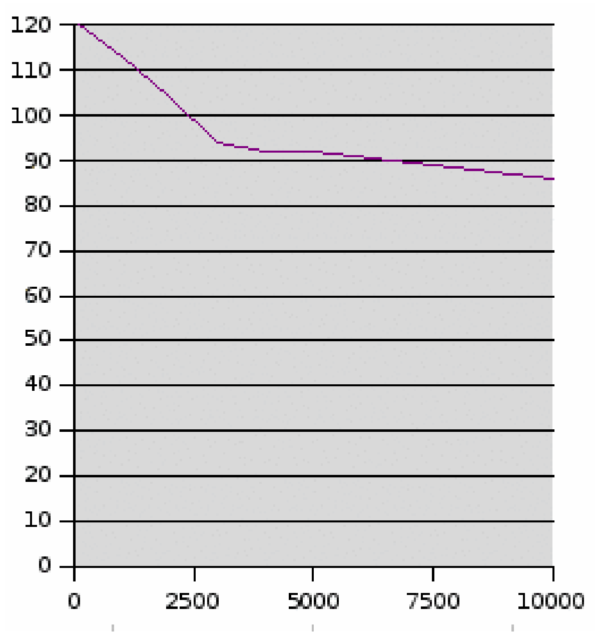
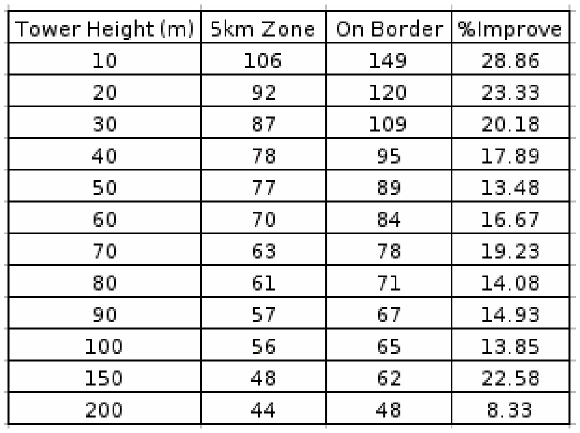


Figure 8: Number of 20 meter Towers vs. Width of Border Zone (meters)



Tower Height (m)	5km Zone	On Border	%Improve
10	106	149	28.86
20	92	120	23.33
30	87	109	20.18
40	78	95	17.89
50	77	89	13.48
60	70	84	16.67
70	63	78	19.23
80	61	71	14.08
90	57	67	14.93
100	56	65	13.85
150	48	62	22.58
200	44	48	8.33

Figure 9: Effect of Zone Size on Number of Necessary Towers

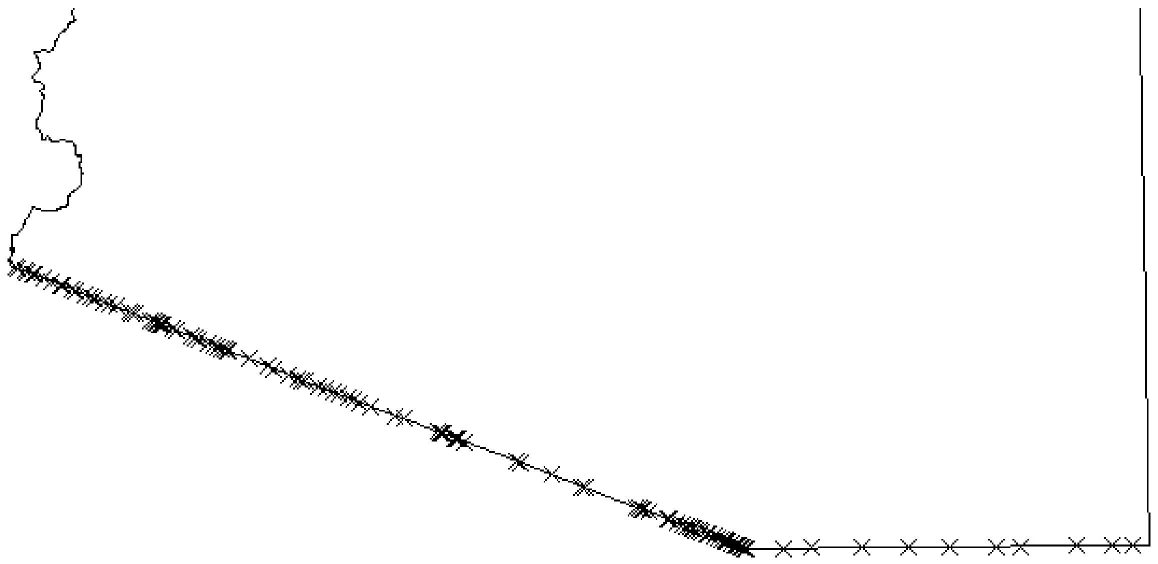


Figure 5: Tower Locations Directly On The Border

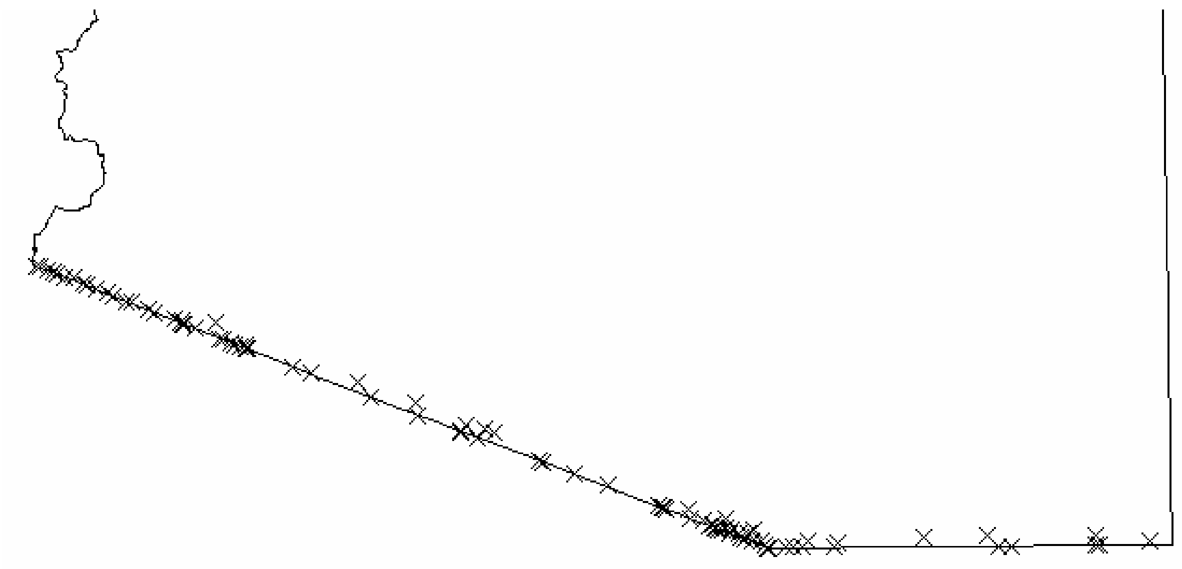


Figure 6: Tower Locations Within A 5km Zone

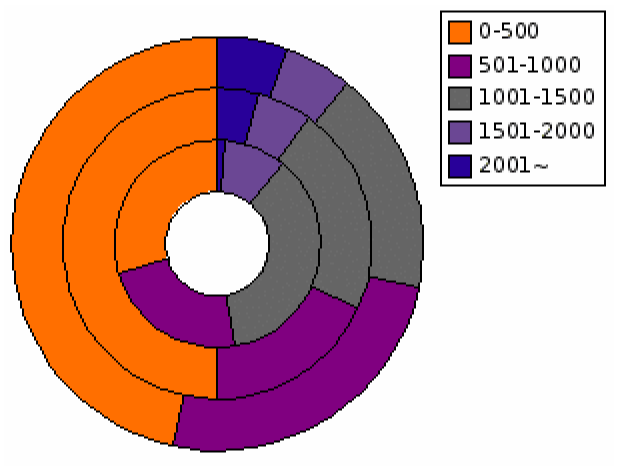


Figure 10: Pie Chart Overlay of Zone Point Elevation, Maxima Elevation, and Tower Elevation

a certain point, we do not gain as much from making the towers taller. By comparing our data with the cost of building towers of varying height, we can theoretically obtain the optimal tower height in terms of cost.

We can also see that expanding the zone from just on the border to some distance away from the border tends to improve performance. As seen in figure 9, we can reduce the number of towers (ranging in height from 10 meters to 100 meters) by between 13.85% and 28.86% by allowing a 5km zone. The number is bound to be inconsistent, because it depends on how many more important maxima we can gain by backing up, and the definition of an important maxima depends on the tower height. We also note that this behavior also seems to be asymptotic, even more so than the tower height. By moving away from the border, we are expanding the visible region, but we are also making ourselves susceptible to more obstacles on the way. Also, we may lose some visible points due to the Earth's curvature. For 20m towers, it looks like 3km away from the border is the best distance. Still, it seems to be very important that we actually use the zone to improve our performance rather than rely on increasing the tower height. For 20m towers, the 23% improvement due to the 5km zone is roughly equivalent to improvement that could be gained by increasing the tower height to 50m.

The distribution graph in figure 10 shows the elevation distribution of 3 categories: the outermost ring corresponds to the entire border zone, the middle ring is the local maxima, and the inner ring shows the actually placed towers. Local maxima are distributed fairly evenly across the zone, as inferred from the graph. In contrast, we can tell that the algorithm preferred to place the towers on higher points. Even though majority of the region is low elevation (0 - 1000m), almost half of the towers are placed at elevation 1000m or higher. We had few towers placed in elevation 2km or higher relative to the region, but it is mostly because all the high elevation points are clustered near the east end of the border, and if we place a few towers in the elevated region their visible field should be large, negating the benefit of more towers in that region.

A major problem with our algorithm is its computational complexity. Disregarding the complexity of file I/O, it first takes $O(n)$ time for locating the maxima, where n is the length of the border. Then scanning the points on the border costs $O(n^3)$ time (each border point $\times$ each candidate point $\times$ linear viewshed computation, though we expect the linear viewshed computation not to be as large as n). The addition of critical towers to the permanent tower list requires another $O(n^3)$ time, as we need to traverse the table to find each point visible from the critical tower. For each point found, we need to traverse its row in the table again to "close" the point and decrement the score of all other towers that can see the point. Again, the last step usually is much smaller than n, as we do not expect all towers to be able to see the same border point. When placing 20m towers on top of the border, our running time was around 90 seconds after extracting the border. As we increase the border zone, running time increases, as is the case when we increase the tower height. This happens due to the method of our viewshed computation. Because we are doing linear line of sight from a point to a point, as soon as we see an obstacle high enough to block the target point on the direct line, we can stop the computation. When more towers tend to be visible from the border, the number of computations required increases.

6 Conclusion

We have determined that the border of Arizona can feasibly be patrolled by observation towers of reasonable height and reasonable distance from the border. We expect that this algorithm could be successfully applied to any border. The execution time of the algorithm for a large data set was not prohibitive, and we conclude that the direct line of sight method for determining visibility is sufficient for the border patrol problem. Finally, we conclude that there are diminishing returns in terms of visibility as tower height and distance from the border are increased. Given appropriate cost parameters, the most efficient means of patrolling the border could be obtained with our method.

7 Acknowledgements

We would like to heartily thank seamless.usgs.gov for presenting us with the elevation data.

References

David Izraelevitz. 2003. A Fast Algorithm for Approximate Viewshed Computation. *American Society for Photogrammetry and Remote Sensing*, 69.7.

Andrew Vincent. 1999. Terrain Occlusion Using Binary Adaptive Ray Casting. *Silicon Graphics Inc.*