IX Brazilian Symposium on GeoInformatics, Campos do Jordão, Brazil, November 25-28, 2007, INPE, p. 183-194.

An efficient algorithm to compute the viewshed on DEM terrains stored in the external memory

Mirella A. Magalhães[1], Salles V. G. Magalhães[1], Marcus V. A. Andrade[1], Jugurta Lisboa Filho[1]

[1]Departamento de Informática – Universidade Federal de Viçosa (UFV)
CEP 36.570.000 – Viçosa – MG – Brazil

{mirella, smagalhaes, marcus, jugurta}@dpi.ufv.br

Abstract. *Nowadays, there is a huge volume of data about terrains available and generally, these data do not fit in the internal memory. So, many GIS applications require efficient algorithms to manipulate the data externally. One of these applications is the viewshed computation that consists in obtain the visible points from a given point p. In this paper, we present an efficient algorithm to compute the viewshed on terrains stored in the external memory. The algorithm complexity is $O(scan(N))$ where N is the number of points in a DEM and $scan(N)$ is the minimum number of I/O operations required to read N contiguous items stored in the external memory. Also, as shown in the results, our algorithm outperforms the known algorithms described in the literature.*

1. Introduction

Terrain modeling is an important area in GIS applications and in general, a terrain can be represented by a *triangulated irregular network (TIN)* or a *Raster Digital Elevation Model (DEM)* [Li et al. 2005, Felgueiras 2001]. A TIN is a vector based representation of a surface made up of irregularly distributed nodes with three dimensional coordinates (x, y, and z) that are connected and arranged in a network of non overlapping triangles. Thus, the surface is approximated by triangle patches and the elevation (the z coordinate) of any point can be interpolated from the vertices of the planar triangle containing the (x, y) coordinates of the point. A DEM is a digital file or a matrix consisting of terrain elevations for ground positions at regularly spaced horizontal intervals.

There is no consensus about which of these representations is the best and there are many discussion about this theme [Kumler 1994, Floriani et al. 1999, Felgueiras 2001]. Anyway, we can say that DEM requires a simple data structure, it is easier to analyze and has high accuracy at high resolution, but it requires high memory space and it is time-consuming processing. On the other hand, TIN has a restricted accuracy, requires more complex algorithms, but it is less memory-consuming and more time-efficient processing. Given its simplicity, in this work, we consider a terrain represented by a DEM.

The recent technological advances in data collection (such as LiDAR) have produced a huge volume of data about Earth's surface [USGS 2007]. For example, a $100km \times 100km$ terrain sampled at $1m$ resolution results in 10^{10} points. And, regardless of the representation used, most of the computational systems can not store/process this huge volume of data internally and thus, they need to be processed in the external memory, generally disks. Since the required time to access and transfer data from and to the exter-

nal memory is much longer than time for internal processing, the algorithms performing external processing must minimize data access [Arge 1997, Goodrich et al. 1993].

More specifically, these algorithms should be designed and analyzed considering a computational model that evaluates the algorithm complexity based on data transfer operations instead of cpu processing operations. One of these models was proposed by Aggarwal and Vitter [Aggarwal and Vitter 1988] where the algorithm complexity is measured considering the number of I/O (input/output) operations executed.

An important GIS problem related to terrain modeling is the computation of all points that can be viewed by a given point (the observer); the region formed by the visible points is named *viewshed* [Floriani and Magillo 2003, Franklin and Ray 1994]. This problem has been widely studied in many applications such as to determine the minimum number of cellular phone towers to cover a region [Ben-Moshe et al. 2007a, Camp et al. 1995, Bespamyatnikh et al. 2001], to optimize the number and position of guards to cover a region [Franklin and Vogt 2006, Eidenbenz 2002], to analyze the influences on property prices in an urban environment [Lake et al. 1998], to optimize path planning on DEM [Lee and Stucky 1998], etc.

In this work, we present an I/O efficient algorithm to compute the viewshed of a point on terrains represented by DEM stored in the external memory. Our algorithm is an adaptation of Franklin and Ray's method [Franklin and Ray 1994, Franklin 2002] to allow an efficient manipulation of huge terrains (5GB or more). The large number of disk accesses is optimized using the library STXXL [Dementiev et al. 2005]. Comparing our algorithm with the original one (adapted to perform external processing) and with the algorithm proposed by Haverkort et al. [Haverkort et al. 2007], the tests showed that our algorithm is about 3.5 times faster than both algorithms and also, it is much simpler and easier to implement than the latter.

The paper is organized as follow: the section 2 gives a brief description about works on viewshed computation and also, on I/O-efficient algorithms for general problems and for viewshed computation too; in the section 3, the viewshed concepts are formally presented; in section 4, the I/O-efficient computational model is shortly described; in section 5, the algorithm is described in details and its complexity is presented in section 6; the tests results are given in section 7 and the conclusions in section 8.

2. Related Works

The visibility on terrains has been widely studied in many different areas. For example, Stewart [Stewart 1998] shows how the viewshed can be efficiently computed for every point of a DEM and his interest involves radio transmission towers positioning. Kreveld [van Kreveld 1996] proposes a sweep-line approach to compute viewshed in $O(nlogn)$ time on a $\sqrt{n} \times \sqrt{n}$ grid. In [Franklin 2002, Franklin and Ray 1994], Franklin and Ray describe experimental studies for fast implementations of visibility computation and present several programs that explore various trade-offs between speed and accuracy. Kim, Rana and Wise in [Young-Hoom et al. 2004] analyze two strategies to use viewshed for optimization problems. Ben-Moshe et al. [Ben-Moshe et al. 2004b, Ben-Moshe et al. 2004a, Ben-Moshe et al. 2007b] have worked on visibility for terrain simplification and for facilities positioning. For a survey on visibility algorithms, see [Floriani and Magillo 2003].

Some problems related to external memory processing are discussed by Aggarwal and Vitter [Aggarwal and Vitter 1988]. They proposed a computational model to evaluate the algorithm complexity considering the number of input/output operations executed. In [Goodrich et al. 1993], Goodrich et al. presented some variants for the sweep plane paradigm considering external processing and Arge et al. [Arge et al. 1995] described a solution for the external processing of line segments in the context of GIS. This technique was also used to solve problems in hydrology such as the computation of the water flow and watershed [Arge et al. 2003] on huge terrains.

Recently, Haverkort et al. [Haverkort et al. 2007] presented an adaption of the Kreveld's method to compute the viewshed on terrains stored in the external memory. The (I/O) complexity of this algorithm is $O(sort(n))$, where n is the number of points in the terrain. It is worth to say that our algorithm described in this paper is faster and easier to implement than that one.

3. Viewshed Problem

Most of GIS problems related to visibility involve the viewshed computation and in general, they are optimization problems such as the optimal positioning of facilities, the siting guards minimization, path planing, etc.

The visibility problems can be classified into two major categories: visibility queries and visibility structures computation. The visibility queries consist in checking if a given point is visible or not from an observer (another point) on the terrain. This query can be answered considering that a point q is visible from another point p if and only if the segment connecting the two points, named *the line of sight*, is strictly above the terrain (except on the ending points p and q). See figure 1

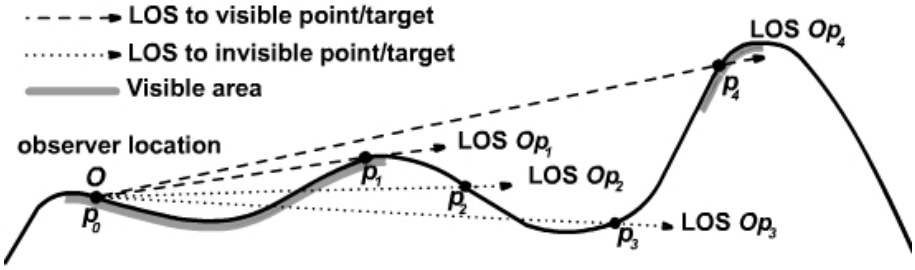

Figure 1. Points Visibility: p_1 and p_4 are visible from p_0; p_2 and p_3 are not visible from p_0.

The visibility structures computation consists in determining some terrain features such as the horizon, the viewshed, etc. The viewshed of a point p on a terrain T can be defined as:

$$viewshed(p) = \{q \in T |\ q \text{ is visible from } p\}$$

Usually, it is convenient to restrict the viewshed to a smaller region, for example, to consider only the points inside a circle centered at p with radius r, the *radius of interest*, that is,

$$viewshed(p, r) = \{q \in T |\ \text{distance}(p, q) \leq r \text{ and } q \text{ is visible from } p\}$$

Unless explicitly said otherwise, when the radius of interest has been defined, we will use $viewshed(p)$ to refer $viewshed(p, r)$.

Usually, the viewshed (in a DEM) is represented by a grid whose size is defined by the radius of interest and each cell stores 1 or 0 to indicate if that cell (point) is visible or not, respectively.

4. I/O efficient Algorithms

As mentioned before, when processing a huge amount of data, the data transfer between fast internal memory and slow external storage (such as disks) often becomes the computation bottleneck. Usually, many GIS software packages implement algorithms for terrain manipulation that were designed assuming internal processing whose aim is to minimize the internal computation time; consequently they often do not scale to large datasets.

In recent years, much research has been done on this topic including the definition of computational models for design and analysis of algorithms that manipulate data in external memory. A model largely accepted was proposed by Aggarwal and Vitter [Aggarwal and Vitter 1988]. Shortly, assuming that M is the internal memory size, B is the disk block size and N is the problem size, this model defines each I/O operation as the transfer of one (disk) block from the external to the internal memory or vice-versa. Then, the measure of performance is determined by the number of such I/O operations executed. The internal computation time is assumed to be free.

Also, the complexity of an algorithm is given based on the complexity of some fundamental problems such as scan or sort N contiguous elements stored in the external memory whose complexities related to I/O operations are:

$$\begin{aligned} scan(N) &= \Theta\left(\frac{N}{B}\right) \\ sort(N) &= \Theta\left(\frac{N}{B}\log_{\left(\frac{M}{B}\right)}\left(\frac{N}{B}\right)\right) \end{aligned}$$

It is important to notice that, usually $scan(N) < sort(N) << N$ and so, in many practical situations, it is significantly better to have an algorithm doing $sort(N)$ instead of N I/O operations. Therefore, many algorithms try to reorganize the data in the external memory to decrease the number of I/O operations executed.

5. External Memory Viewshed Computation (EMVS)

Our algorithm, named External Memory Viewshed (EMVS), is based on the method proposed by Franklin and Ray [Franklin and Ray 1994] that computes the viewshed of a point on a terrain represented as a internal memory matrix. A short description of this method is given below.

5.1. Franklin and Ray's Method

Given a terrain represented by a $n \times n$ elevation matrix T and given a point p on T, the algorithm computes the viewshed of p considering a circle of radius r (the radius of interest) centered at p. The algorithm performs a radial sweep of this circle using a ray, a line of sight (*LOS*), starting at p and thus, it walks along each *LOS* to determine if each

terrain position on the *LOS* is visible from p or not. A terrain position q is visible from p if the *LOS* does not intersect any position whose height is higher than q.

To simplify the circle sweeping, the algorithm uses a square bounding box centered at p with side $2r$ and the lines of sight are defined connecting p to each cell in the square border. Initially, all cells inside the bounding box are set as not visible and for each line of sight l, the algorithm starts at p setting the height of l as $-\infty$ (i.e., a big negative number). So, this height is updated (increased) whenever a higher cell is reached, that is, supposing the current l's height is h and the next cell height is h', if $h < h'$ then the cell is marked as visible and the l's height is updated to h'; on the other hand, if $h \geq h'$, the cell status and the l's height are preserved. The viewshed is stored as a $2r \times 2r$ bit matrix where the visible positions are indicated by 1 and the not visible by 0 (the positions inside the square but outside the circle are set as not visible).

At first glance, the algorithm could be easily adapted to access the terrain points stored in the external memory in the sequence determined by the radial sweep. But, since the terrain matrix is, as usual, stored row by row (in a file), the radial sweeping order would require a "random" access to the file and the execution time would be unacceptably long. Therefore, we adapted this algorithm to avoid the random access order.

5.2. The EMVS algorithm

The basic idea is to generate a list containing the terrain positions (points) sorted by the processing order, that is, the points will appear in the list in the sequence that they will be processed. Thus, instead of accessing the file in a random sequence to process the points as they are reached along the line of sight, the algorithm will access (and process) the points in a sequential order.

It is important to say that the list is also stored in the external memory, but it is managed by a special library STXXL (*Standard Template Library for Extra Large Data Sets*) [Dementiev et al. 2005] that implements containers and algorithms to process huge volumes of data. This library allows an efficient manipulation of data stored externally and, as stated by the authors, "it can save more than half the number of I/Os performed by many applications".

More specifically, the algorithm creates a list L of pairs (c, i) where c is a matrix cell (a terrain point) and i is an index that indicates "when" the cell c should be processed. That is, if a cell c has the index $i = k$ then c will be the kth cell to be processed.

To compute the indices, the lines of sight (originating at the observer p) are numbered in the counterclockwise order starting in the horizontal left to right line of sight that receives the number 0 - see figure 2. Thus, the cells are numbered increasingly along each line of sight; when a line of sight ends, the enumeration continues from the observer (again numbered) following the next line of sight. Of course, a same cell (point) can receive multiple indices since it can be intercepted by many lines of sight. It means that a same point can appear in multiple pairs in the list L, but each pair will have a different index. Also, if the observer is near to the terrain border, that is, if the distance between the observer and the terrain border is smaller than the radius of interest r, some "cells" in the line of sight can be outside the terrain. In this case, those "cells" still will be numbered but they will be ignored and will not be inserted in the list L. This is done to avoid additional tests during the indices computation.

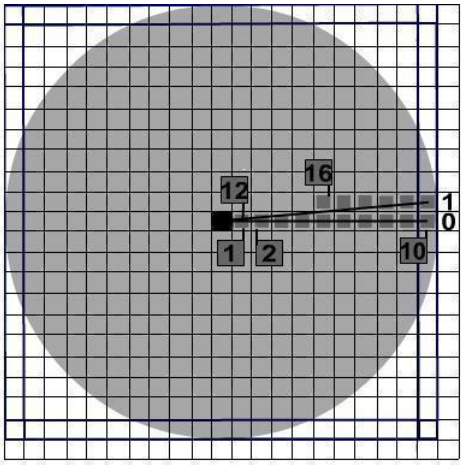

Figure 2. Line of sight numeration

It is important to notice that if the cells indices were computed following the lines of sight as described above, the cells still would be randomly accessed as in the original algorithm. So, to build the list L, the algorithm reads the terrain cells sequentially from the external file and for each cell c, it determines (the number of) all lines of sight that intercept the cell.

Since a cell is not "undimensional", we can determine the cells intercepted by a line of sight using a process similar to the line rasterization [Bresenham 1965]. That is, let s be the side of each (square) cell and suppose the cell is referenced by its center. Also, let a be a line of sight whose slope is α and suppose that $0 < \alpha \leq 45°$ [1]. So, given a cell $c = (c_x, c_y)$, see figure 3, the line of sight a "intersects" the cell c if and only if the intersection point between a and the vertical line c_x is between the points $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$; more precisely, given $(q_x, q_y) = a \cap c_x$, a intersects c if and only if $c_y - 0.5s \leq q_y < c_y + 0.5s$.

Then, as it is easy to see, all lines of sight intersecting the cell c are those between the two lines passing through the points $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$ - figure 3. Let k_1 and k_2 be the numbers of these two lines respectively. Thus, considering the line enumeration sequence, the number of all intersecting lines are $k : k_1 \leq k \leq k_2$.

Now, given a cell c, let r be the number of a line of sight intercepting c. Then, the index i of the cell c associated to r is given by the formula $i = r * n + d$, where n is the number of cells in each ray (this number is constant for all rays in the bounding square box) and d is the (horizontal or vertical) distance between the points c and p - see figure 4. Notice that the distance d is defined as the maximum between the number of rows and columns from p to c.

Next, the list L is sorted by the elements index and then, the cells are processed in the sequence given by the sorted list. Notice that, when a cell c is processed, all the "previous" cells that could block its visibility were already processed. So, the visibility of c can be computed, as described above, just checking the height of the cells along the line of sight. When a cell located on the square border is processed, it means that the

[1] For $45° < \alpha \leq 90°$, use a similar idea interchanging x and y.

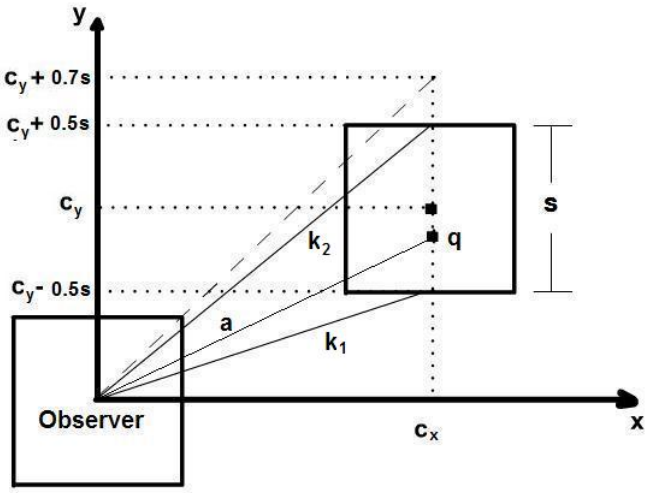

Figure 3. Lines of sight intersecting a cell

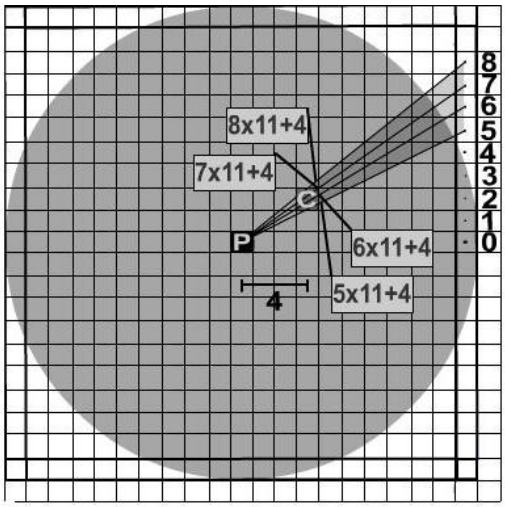

Figure 4. The index cell computation.

processing of a line of sight has finished and the next cell in the list will be the observer's cell indicating that the processing of a new line of sight will start.

For the sake of efficiency, the algorithm uses another list L' (also stored externally and managed by *STXXL*) to keep only the visible cells. More precisely, when the algorithm determines that a cell c is visible, this cell is inserted in the list L'. This list is, in general, much smaller than L since it does not keep the indices and also, usually many points are not visible.

Given the visible cells in L', the algorithm saves the viewshed as a matrix of 0s and 1s such that a visible position is indicated by 1 and a not visible position by 0. This matrix is also stored externally and to generate it (avoiding "random access") the list L' is sorted lexicographically by x and y and each element of the sorted list is stored in the corresponding matrix position.

Finally, it is worth to say that an efficiency improvement is achieved storing a piece of the terrain matrix in the internal memory and so, a lot of I/O operations are avoided. The idea is to store in the internal memory the cells around the observer since those cells are processed more times than the farthest cells. Then, the algorithm selects all cells inside a square centered at the observer position, stores those cells in the internal memory and they are not inserted in the list L. In this way, when a cell needs to be processed, the algorithm checks if it is in the internal memory. If yes, the cell is processed normally; otherwise, it is read from the list L.

6. Algorithm complexity

Let T be a terrain represented by a $n \times n$ elevation matrix. So, T has n^2 cells (points). Also, let p be the observer's point and let r be the radius of interest. As described in section 5.2, the algorithm analyzes the cells that are inside the $2r \times 2r$ square centered at p. Assuming that each cell's side is s then there are, at most[2], $\frac{2r}{s}$ cells in each square's side which implies there are $\frac{8r}{s}$ cells on the square's perimeter. Let $K = \frac{r}{s}$. Thus, the algorithm shots $8K$ lines of sight and since each line of sight has K cells, the list L has, in the worst case, $O(K^2)$ elements.

In the first step, the algorithm does $\frac{n^2}{B}$ I/O operations to read the cells and to build the list L. Next, the list with $O(K^2)$ elements is sorted and then it is swept to compute the cell's visibility. Thus, the total number of I/O operations is:

$$O\left(\frac{n^2}{B}\right) + O\left(\frac{K^2}{B}\log_{\left(\frac{M}{B}\right)}\left(\frac{K^2}{B}\right)\right) + O\left(\frac{K^2}{B}\right)$$

Since the radius of interest r is (much) smaller than n (the terrain matrix side) then K is smaller than n and so, the number of I/O operations is given by $O(\frac{n^2}{B}) = O(scan(n^2))$.

The algorithm also uses an additional external list L' to keep the visible cells and this list needs to be sorted. But, since the list size is (much) smaller than the size of L, the number of I/O operations executed in this step does not change the algorithm complexity. Thus, we can conclude that the algorithm complexity is $O(scan(n^2))$.

7. Results

The algorithm EMVS was implemented in C++, using *g++ 4.1.1*, and the tests were executed in a PC Pentium with 2.8 GHZ, 1 GB of RAM, 80 GB 7200 RPM serial ATA HD running Mandriva Linux.

The algorithm execution time was compared with the Franklin and Ray's algorithm (WRF_VS) which was adapted to manipulate huge terrains stored externally. The adapted algorithm also maintains part of the terrain in internal memory in a similar way that EMVS does. The table 1 and the charts in the figure 5 show the execution time (in seconds) to compute the viewshed considering different radii of interest (ROI) on terrains of different sizes. The 1.45GB (27596 $\times$ 27596 points) and 5.7GB (55192 $\times$ 55192 points) terrains were generated by the concatenation of 4 and 16 instances of a 363MB

[2]If the observer is close to the terrain border, the square might not be completely contained in the terrain.

(13798 $\times$ 13798) matrix representing the Hawaii Big Island. The 2GB (32427 $\times$ 32427 points) terrain was generated by the concatenation of many instances of a 1201×1201 matrix representing the Lake Champlain West (USA-Canada border). These datasets are interesting because they have large height variations since they include lake, ocean and mountains.

Each table entry was obtained by the average of three execution time using different observer positions randomly selected on each terrain.

	1.45GB		2GB		5.7GB
ROI	EMVS	WRF_VS	EMVS	WRF_VS	EMVS
100	21	77	26	121	78
500	26	81	30	122	85
1000	33	97	36	128	99
5000	137	438	73	316	248
10000	478	1313	219	833	643
15000	836	2977	446	1855	1663

Table 1. Execution time (in seconds)

.

Based on these results, it is possible to conclude that the EMVS algorithm is about 3.5 times faster than WRF_VS and also, the former can process much larger terrain (5.7GB or more) while the latter is limited to 2 GB. On the other hand, it is important to say that when the terrain is "small" (i.e. when it fits in the internal memory) the WRF_VS algorithm is a little faster than EMVS, mainly because the lists management and sorting add a time overhead that is not amortized when the terrain size is small.

Furthermore, comparing the EMVS execution time with those reported by Haverkort et al. [Haverkort et al. 2007], we can conclude that our algorithm, besides of being much simpler and easier to implement, is also more than 3.5 times faster than that one. Additionally, it is worth to say that, in their tests, they used a Power Macintosh G5 dual 2.5 GHz, 1GB RAM and 80 GB 7200 RPM that is considerable faster than the machine used in our tests. Thus, it is correct to suppose that our algorithm is still faster than that one.

8. Conclusions and future works

We presented an I/O-efficient algorithm to compute the viewshed of a point in huge terrains represented by a raster DEM stored in the external memory. As tests showed, our algorithm is more than 3.5 times faster than the other ones described in the literature and also, it can process very huge terrains (we used it in 5.7GB terrain). Furthermore, the algorithm is quite simple to understand and to implement. The algorithm implementation is available at http://www.dpi.ufv.br/ marcus/TerrainModeling/EMViewshed/EMVS.tgz as an open source code distributed under Creative Common GNU GPL license [Creative Commons 2007].

As a next step, we started to work on the NP-hard optimization problem to site observers in huge terrains stored in the external memory. Our aim is to develop an approximation algorithm to place the "almost" minimum number of observers necessary to

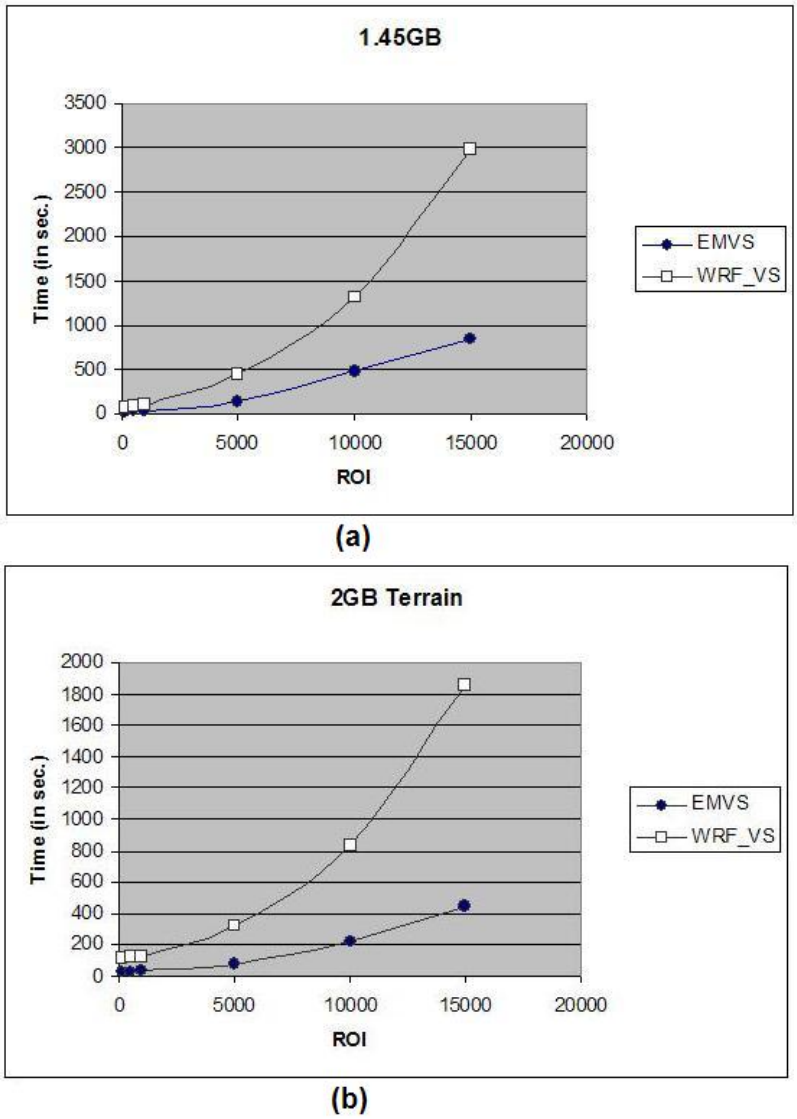

Figure 5. Execution time charts: (a) in 1.45GB terrain; (b) in 2GB terrain.

cover "almost" the whole terrain. More precisely, given a terrain T, we want to determine where to site the almost minimum number of observers to cover visually an user defined percentage of the terrain. This problem has a wide range of applications such as: telecommunications (cellular towers), military (guards), agriculture (irrigation), etc.

Acknowledgment

This work was partially supported by CNPq and FAPEMIG.

References

Aggarwal, A. and Vitter, J. S. (1988). The input/output complexity of sorting and related problems. *Communications of the ACM*, 9:1116–1127.

Arge, L. (1997). External-memory algorithms with applications in geographic information systems. In In M. van Kreveld, J. Nievergelt, T. R. e. P. W., editor, *Algorithmic Foundations of GIS*. Springer-Verlag.

Arge, L., Chase, J. S., Halpin, P., Toma, L., Vitter, J. S., Urban, D., and Wickremesinghe, R. (2003). Efficient flow computation on massive grid terrains. *GeoInformatica*, 7:283–313.

Arge, L., Vengroff, D. E., and Vitter, J. S. (1995). External memory algorithms for processing line segments in geographic information systems. In *In Proc. European Symposium on Algorithms, LNCS 979*, pages 295–310.

Ben-Moshe, B., Ben-Shimol, Y., and Y. Ben-Yehezkel, A. Dvir, M. S. (2007a). Automated antenna positioning algorithms for wireless fixed-access networks. *Journal of Heuristics*, 13(3):243–263.

Ben-Moshe, B., Carmi, P., and Katz, M. (2004a). Approximating the visible region of a point on a terrain. In *Proc. Algorithm Engineering and Experiments (ALENEX'04)*, pages 120–128.

Ben-Moshe, B., Katz, M., Mitchell, J., and Nir, Y. (2004b). Visibility preserving terrain simplification - an experimental study. *Comp. Geom., Theory and Applications*, 28:175–190.

Ben-Moshe, B., Katz, M. J., and Mitchell, J. S. B. (2007b). A constant-factor approximation algorithm for optimal 1.5d terrain guarding. *SIAM J. Comput*, 36(6):1631–1647.

Bespamyatnikh, S., Chen, Z., Wang, K., and Zhu, B. (2001). On the planar two-watchtower problem. In *In 7th International Computing and Combinatorics Conference*, pages 121–130.

Bresenham, J. (1965). An incremental algorithm for digital plotting. *IBM Systems Journal*.

Camp, R. J., Sinton, D. T., and Knight, R. L. (1995). Viewsheds: A complementary management approach to buffer zones. *Wildlife Society Bulletin*, 25:612–615.

Creative Commons (2007). http://creativecommons.org/license/cc-gpl (accessed august 2007).

Dementiev, R., Kettner, L., and Sanders, P. (2005). Stxxl : Standard template library for xxl data sets. Technical report, Fakultat fur Informatik, Universitat Karlsruhe. http://stxxl.sourceforge.net/ (acessed on July 2007).

Eidenbenz, S. (2002). Approximation algorithms for terrain guarding. *Inf. Process. Lett.*, 82(2):99–105.

Felgueiras, C. A. (2001). Modelagem numérica de terreno. In In G. Câmara, C. Davis, A. M. V. M., editor, *Introdução à Ciência da Geoinformação*, volume 1. INPE.

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

Floriani, L. D., Puppo, E., and Magillo, P. (1999). Applications of computational geometry to geographic information systems. In J. R. Sack, J. U., editor, *Handbook of Computational Geometry*, pages 303–311. Elsevier Science.

Franklin, W. R. (2002). Siting observers on terrain. In Springer-Verlag, editor, *In D. Richardson and P. van Oosterom editors, Advances in Spatial Data Handling: 10th International Symposium on Spatial Data Handling*, pages 109–120.

Franklin, W. R. and Ray, C. (1994). Higher isn't necessarily better - visibility algorithms and experiments. In *6th Symposium on Spatial Data Handling*, Edinburgh, Scotland.

Franklin, W. R. and Vogt, C. (2006). Tradeoffs when multiple observer siting on large terrain cells. In *12th International Symposium on Spatial Data Handling*.

Goodrich, M. T., Tsay, J. J., Vangroff, D. E., and Vitter, J. S. (1993). External-memory computational geometry. In *IEEE Symp. on Foundations of Computer Science*, pages 714–723.

Haverkort, H., Toma, L., and Zhuang, Y. (2007). Computing visibility on terrains in external memory. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments / Workshop on Analytic Algorithms and Combinatorics (ALENEX/ANALCO)*.

Kumler, M. P. (1994). An intensive comparison of triangulated irregular network (tins) and digital elevation models (dems). *Cartographica*, 31(2).

Lake, I. R., Lovett, A. A., Bateman, I. J., and Langford, I. H. (1998). Modeling environmental influences on property prices in an urban environment. *Computers, Environment and Urban Systems*, 22:121–136.

Lee, J. and Stucky, D. (1998). On applying viewshed analysis for determining least-cost paths on digital elevation models. *Journal of Geographical Information Science*, 12:891–905.

Li, Z., Zhu, Q., and Gold, C. (2005). *Digital Terrain Modeling - principles and methodology*. CRC Press.

Stewart, A. J. (1998). Fast horizon computation at all points of a terrain with visibility and shading applications. In *IEEE Trans. Visualization Computer Graphics*, pages 82 – 93.

USGS (2007). The USGS Center for LIDAR Information Coordination and Knowledge. http://lidar.cr.usgs.gov/ (accessed October 2007).

van Kreveld, M. (1996). Variations on sweep algorithms: efficient computation of extended viewsheds and class intervals. In *Symposium on Spatial Data Handling*, pages 15–27.

Young-Hoom, K., Rana, S., and Wise, S. (2004). Exploring multiple viewshed analysis using terrain features and optimization techniques. *Computers and Geosciences*, 30:1019–10323.