Efficient viewshed computation on external memory DEM terrains

Mirella A. Magalhães
DPI - UFViçosa
mirella@dpi.ufv.br

Salles V. G. Magalhães
DPI - UFViçosa
smagalhaes@dpi.ufv.br

Marcus V. A. Andrade
DPI - UFViçosa
marcus@dpi.ufv.br

Abstract

Many GIS applications require efficient algorithms to manipulate huge volume of data about terrains stored in external memory. One of these applications is the viewshed computation which consists in obtaining the points that can be viewed by a given point. In this paper, we present an efficient algorithm to compute the viewshed on huge terrains stored in external memory.

1. Introduction

Terrain modeling is an important area in GIS applications and the recent technological advances in data acquisition (such as LiDAR) have produced a huge volume of information about the earth surface. For example, a $10km\times 10km$ terrain sampled at $1m$ resolution is represented by 10^8 points. To manipulate this huge volume of data (which does not fit in the internal memory) it is essential to use methods that minimize the external memory access [2, 6]. More generically, Aggarwal and Vitter [1] proposed a computational model to evaluate the complexity of algorithms that manipulate data stored in external memory. The complexity is defined considering the number of I/O operations executed.

An important problem involving terrains is the *viewshed* computation which consists in determining all points or region that can be viewed by a given point [4]. This problem has many applications such as: to determine the minimum number of cell phone towers to cover a region; to optimize the number of guards to safeguard a region, etc.

In this work, we consider a terrain represented by a regular grid (DEM) stored in the external memory and we present an I/O efficient algorithm to compute the *viewshed* of a point in the terrain. Our algorithm is an adaptation of Franklin's method [5] to allow an efficient manipulation of huge terrains (4GB or more). The large number of disk accesses is optimized using the library *STXXL* [3]. Comparing our algorithm with the algorithm proposed by Haverkort et al. [7], the tests showed that our algorithm is about 4 times faster than Haverkort's one and also, it is much simpler and easier to implement.

2. Computing viewshed on external memory

Given a terrain represented by a $n \times n$ elevation matrix T and given a point p on T, the algorithm proposed by Franklin [5] computes the viewshed of p considering a circle of radius r (the radius of interest) centered on p. To sweep this circle, the algorithm uses a square bounding box of side $2r$ and each cell in the square border is connected to p. For each ray, the algorithm determines if the terrain positions are visible or not from p. More precisely, initially all cells are set as not visible and given a ray l, the algorithm starts at p setting the ray height as $-\infty$ (i.e., a big negative number). So, this height is updated (increased) whenever a higher cell is accessed, that is, supposing the current ray height is h and the next cell height is h', if $h < h'$ then the cell is marked as visible and the ray height is updated to h'; on the other hand, if $h \geq h'$, the cell status and the ray height are preserved. The viewshed is stored in 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).

When the terrain is huge and the grid does not fit in the internal memory, one could think to do a simple adaptation to access the grid stored in the external memory. But, assuming the grid (matrix) is stored row by row, the cells processing order would require a "random" access sequence and the execution time would be too long.

To avoid the "random" access order we rearrange the cells based on the processing order. This is done using an external memory list L, managed by the *STXXL* library, and each list element is a pair containing a cell and an index indicating "when" that cell should be processed. The list is initialized with all cells and their respective indices, computed as described below. Next, the list is sorted by the indices and the cells are processed in this order.

To compute indices, let's suppose the rays are numbered $0, 1, \cdots k$ in the counterclockwise order, starting in the horizontal left to right ray - see figure 1 (a). So, the index ind of a cell c is given by $ind = r_i * n + d$, where r_i is the number of the ray passing through c, n is the number of cells in each ray [1] and d is the (horizontal or vertical) distance between the cell c and the point p. Notice that a cell can have more than one index (i.e, a cell will be processed more than one time); in this case, the cell is inserted in the list more than one time, one for each index - see figure 1 (b).

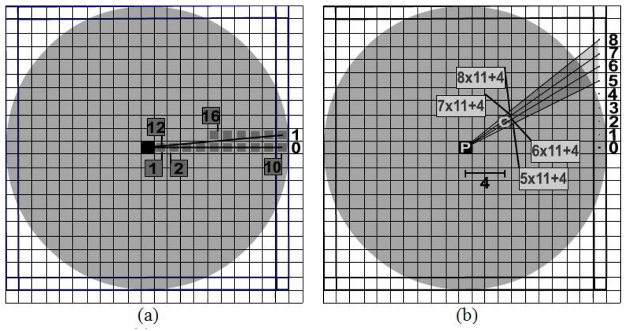

Figure 1. The processing order indices.

Thus, if the cells are processed in the sequence given by the list (sorted by the indices), it corresponds to a sequential access order and when a cell c is processed, all the "previous" cells that could block the visibility of c were already processed. Also, when a cell located on the square boundary is processed, it means that the processing of a ray has finished and the next cell in the list will be the observer's cell indicating that the processing of a new ray will start. So, the "random" access order is avoided.

3. Results, conclusion and future work

Our algorithm was implemented in C++, using *g++ 4.1.1*, and it was executed in a PC Pentium with 2.8 GHZ, 1 GB of RAM, 80 GB HD running Mandriva Linux. In this implementation, named EM_VS, we maintain a piece of the terrain in the internal memory and those cells that are not in the internal memory are stored in the list L.

The efficiency of our algorithm was compared with the Franklin's algorithm (WRF_VS) which was adapted to manipulate huge terrains stored externally. This adaptation also maintains a piece of the terrain in internal memory.

The table 1 shows the execution time (in seconds, including the index processing time) to compute the viewshed of a point (using different radii of interest - ROI) in terrains with 32427×32427 and 48040×48040 points (about 2GB and 4GB resp., since each elevation uses 2 bytes). These terrains were artificially generated by the concatenation of many instances of a 1201×1201 matrix representing the Lake Champlain (USA-Canada border). This dataset is interesting because it has flat regions (lake) and mountains.

Based on these results, we can conclude that our algorithm is about 4 times faster than the Franklin's algorithm and also, it allows the processing of much larger terrain (4GB or more while the original is limited to 2 GB). Furthermore, comparing our algorithm execution time with those reported by Haverkort et al. [7], we can conclude that our algorithm, besides of being much more simpler, it is also more than 4 times faster than that one.

As a next step, we are working on the NP-hard optimization problem of siting observer in huge terrains where our aim is to develop an approximation algorithm to place the minimum number of observers necessary to "see" (almost) the whole terrain.

	EM_VS		WRF_VS
ROI	2GB	4GB	2 GB
100	32	18	121
500	27	87	122
1000	31	87	128
5000	73	155	316
10000	219	587	833
15000	446	848	1855

Table 1. Execution time.

Acknowledgment

This work was partially supported by CNPq and FAPEMIG.

References

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

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

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

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

[5] W. R. Franklin. 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*, page 109120, 2002.

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

[7] H. Haverkort, L. Toma, and Y. Zhuang. Computing visibility on terrains on external memory. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*, 2007.

1 Considering the square box, this number is constant for all rays.