More efficient terrain viewshed computation on massive datasets using external memory

Chaulio R. Ferreira
Universidade Fed. de Viçosa
Viçosa, MG, Brazil
chaulio.ferreira@ufv.br

Salles V. G. Magalhães
Universidade Fed. de Viçosa
Viçosa, MG, Brazil
salles@ufv.br

Marcus V. A. Andrade
Universidade Fed. de Viçosa
Viçosa, MG, Brazil
marcus@ufv.br

W. Randolph Franklin
Rensselaer Polytechnic Inst.
Troy, NY, USA
wrf@ecse.rpi.edu

André M. Pompermayer
Universidade Fed. de Viçosa
Viçosa, MG, Brazil
andre.pompermayer@ufv.br

ABSTRACT

We present a better algorithm and implementation for external memory viewshed computation. It is about four times faster than the most recent and most efficient published methods. Ours is also much simpler. Since processing large datasets can take hours, this improvement is significant. To reduce the total number of I/O operations, our method is based on subdividing the terrain into blocks which are stored in a special data structure managed as a cache memory.

The viewshed is that region of the terrain that is visible by a fixed observer, who may be on or above the terrain. Its applications range from visual nuisance abatement to radio transmitter siting and surveillance.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations

General Terms

Algorithms, Experimentation, Performance

Keywords

Terrain modeling, GIS, External memory processing

1. INTRODUCTION

Terrain modeling has been widely used in Geographical Information Science (GIS) including applications in hydrology, visibility and routing. In visibility applications it is usual to compute which points can be viewed from a given point (the *observer*) and the region composed of such points, known as *viewshed* [6]. Some applications include minimizing the number of cellular phone towers required to cover a region [3], optimizing the number and position of guards to cover a region [7], etc.

There are various algorithms for viewshed computation but most of them were designed assuming that the terrain data fits in internal memory. However, the huge volume of high resolution terrestrial data available has become a challenge for GIS since the internal memory algorithms do not run well for such volume of data on most computers. Thus, it is important to optimize the massive data processing algorithms simultaneously for computation and data movement between the external and internal memory since processing data in external memory takes much more time. That is, the algorithms for external memory should be designed (and analyzed) considering a computational model where the algorithm complexity is evaluated based on I/O operations. A model often used was proposed by Aggarwal and Vitter [1] where an I/O operation is defined as the transfer of one disk block of size B between the external and internal memory and the performance is measured considering the number of such I/O operations. The internal computation time is assumed to be comparatively insignificant. An algorithm's complexity is related to the number of I/O operations performed by fundamental operations such as scanning or sorting N contiguous elements. Those are $scan(N) = \theta(N/B)$ and $sort(N) = \theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ where M is the internal memory size.

This work presents an efficient algorithm, named *TiledVS*, to compute the viewshed of a point on terrains stored in external memory. The large number of disk accesses is optimized using a new library to manage the data swap between the external and internal memories. This new algorithm was compared against the most recent and most efficient published methods: *EMViewshed* [2] and *io_radial2*, *io_radial3* and *io_centrifugal* [4]. Our new method is much simpler and, also, the tests showed that it is more than four times faster than all of them. Since processing large datasets can take hours, this improvement is significant.

2. DEFINITIONS AND RELATED WORKS

A *terrain* is a tridimensional surface τ where any vertical line intersects τ in, at most, one point. In this paper we will consider terrains represented by *Raster Digital Eleva-*

tion Models (DEMs) [5] since they use simpler data structures, i.e., matrices storing the elevations of regularly spaced positions of the terrain.

An *observer* is a point in the space from where the other terrain points (the *targets*) will be visualized. Both the observer and the targets can be at a given height above the terrain, respectively indicated by h_o and h_t. Usually, it is assumed that the observer has a range of vision ρ, the *radius of interest*, which means that the observer can see points at a given distance ρ. Thus, a target T is visible from O if and only if the distance of T from O is, at most, ρ and the straight line, the *line of sight*, from O to T is always strictly above the terrain. See Figure 1.

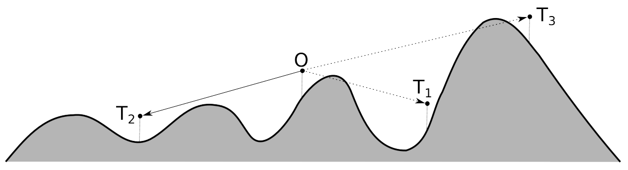


Figure 1: Target's visibility: T_1 and T_3 are not visible but T_2 is.

The *viewshed* of O corresponds to all points that can be seen by O. Since we are working with raster DEMs, we represent a viewshed by a square $(2\rho+1)\times(2\rho+1)$ matrix of bits where 1 indicates that the corresponding point is visible and 0 is not. By definition, the observer is in the center of this matrix.

Earlier works have presented different methods for viewshed computation. Among the methods for DEM terrains, we can point out the one proposed by Van Kreveld [9], and the one by Franklin et al., named *RFVS* [6]. These two methods are very efficient and are particularly important in this context because they were used as the base for some very recent and efficient methods for the viewshed computation in external memory: Fishman et al. [4] adapted Van Kreveld's method, and Andrade et al. [2] adapted the *RFVS* method. This work also presents an IO-efficient adaptation of the *RFVS* method. Therefore, below we will give a short description of the *RFVS* method.

In that method, the terrain cells' visibility is computed along rays connecting the observer to all cells in the boundary of a square of side $2\rho+1$ centered at the observer where ρ is the radius of interest. That is, the algorithm creates a ray connecting the observer to a cell on the boundary of this square, and this ray is counterclockwise rotated around the observer following the cells in that boundary and the visibility of the cells in each ray is determined following the cells on the segment. Thus, suppose the segment is composed by cells $c_0, c_1, \cdots, c_k$ where c_0 is the observer's cell and c_k is a cell in the square boundary. Let α_i be the slope of the line connecting the observer to c_i and let μ be the highest slope among all lines already processed, that is, when processing cell c_i, $\mu = \max\{\alpha_1, \alpha_2, \cdots, \alpha_{i-1}\}$. Thus, the target on c_i is visible if and only if the slope of the line from O to the target above c_i is greater than μ. If yes, the corresponding cell in the viewshed matrix is set to 1; otherwise, to 0. Also, if $\alpha_i > \mu$ then μ is updated to α_i. We say that a cell c_i blocks the visibility of the target above c_j if cell c_i belongs to the segment $\overline{c_0c_j}$ and α_i is greater or equal to the slope of the line connecting the observer to the target above c_j.

3. *TiledVS* METHOD

As mentioned in section 2, the *RFVS* sweeps the terrain cells rotating a ray connecting the observer cell to a cell in the boundary of a bounding box and the cells' visibility is processed along this ray. Thus, the matrix access pattern presents a spatial locality of reference, that is, in a short time interval, the accessed cells are close in the matrix. However, this access pattern is not efficient in external memory since the cells which are close in the (bidimensional) matrix may not be stored close because, usually, a matrix is stored using a linear row-major order.

To reduce the number of non-sequential accesses, we present a new method, called *TiledVS*, where the basic idea is to adapt the *RFVS* algorithm to manage the access to the matrices stored in external memory using the library *TiledMatrix* [8]. In brief, this library subdivides the matrix in small rectangular blocks (*tiles*) which are sequentially stored in the external memory. When a given cell needs to be accessed, the whole block containing that cell is loaded into the internal memory. The library keeps some of these blocks in the internal memory using a data structure, named *MemBlocks*, which is managed as a "cache memory" and the replacement policy adopted is based on *least recently used - LRU*. That is, when a block is accessed it is labeled with a *timestamp* and if it is necessary to load a new block into the cache (and there is no room for this block), the block with smaller *timestamp* is replaced with the new block. When a block is evicted, it is checked if that block was updated (it is particularly important for the viewshed matrix); if any cell was updated then the block is written back to the disk.

Now, we will show that it is possible to define the *MemBlocks* size such that the adopted matrix partitioning associated with the *LRU* policy can be effective for the *RFVS* algorithm, that is, we will prove that this process will load a block in the cache, keep it there while it is accessed and it will be evicted only when it will be no longer needed.

In the following, we will suppose that the matrix partitioning creates square blocks with $\omega \times \omega$ cells and these blocks are grouped in vertical bands with ω columns of cells. See figure 2. And, given a ray r defined by the *RFVS* algorithm, without loss of generality, in the demonstrations below, we will consider rays whose slope is, at most, 45°. For rays with greater slope just replace rows with columns.

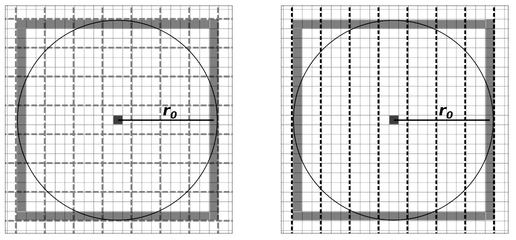


Figure 2: Matrix partitioning: (a) square blocks with 3×3 cells; (b) vertical bands with 3 columns.

LEMMA 3.1. *Any ray intersects, at most, $\frac{\rho}{\omega}+2$ bands where ρ is the radius of interest (in number of terrain cells).*

Proof For the viewshed computation, the *RFVS* algorithm defines a square bounding box of side $2\rho+1$ with the observer on its center and creates rays connecting the observer to the center of the cells in the square border. Since any ray whose slope is, at most, 45° intersects $\rho+1$ columns in this square, this ray intersects $\lceil \frac{\rho+1}{\omega} \rceil + 1$ vertical bands. The additional $+1$ is because the observer may not be in the central column of a band (notice that, if the observer in the Figure 2(b) is moved one cell to the right, ray r_0 will cross the last band boundary and will intersect an aditional band). Since $\lceil \frac{\rho+1}{\omega} \rceil = \lfloor \frac{\rho}{\omega} \rfloor + 1$ then $\lceil \frac{\rho+1}{\omega} \rceil + 1 \leq \frac{\rho}{\omega} + 2$.

LEMMA 3.2. *Let r_k and r_{k+1} be two consecutive rays in the* RFVS *algorithm sweeping. Then these two rays intersect, at most, $2\left(\frac{\rho}{\omega}+2\right)$ blocks.*

Proof Since the *RFVS* algorithm uses the Bresenham rasterization method, there is exactly one cell for each column intersected by a ray. Let l_r and c_r be respectively the number of rows and columns intersected by a ray r. As the ray slope is, at most, 45° then $l_r \leq c_r$.

Given two consecutive rays r_k and r_{k+1}, the vertical distance between them is, at most, one cell side - see Figure 3(a). As, for each vertical band, they intersect ω columns, they can intersect, at most, $\omega+1$ rows in that band. Thus, in each band, they can intersect, at most, two blocks (since the block height is ω rows). Therefore, from Lemma 3.1, rays r_k and r_{k+1} can intersect, at most, $2\left(\frac{\rho}{\omega}+2\right)$ blocks.

LEMMA 3.3. *Let r_0 be the first ray in the sweeping sequence. Given a block B not intersected by r_0, let r_k and r_{k+1} be two consecutive rays. If r_k intersects B and r_{k+1} doesn't, then no other ray after r_k will intersect block B.*

Proof It is straightforward from the fact that the algorithm uses a radial sweeping sequence and the blocks are convex. And it doesn't work for the blocks intersected by ray r_0 because, considering the radial sweeping, these blocks can be intersected again by the last rays. See Figure 3(b).

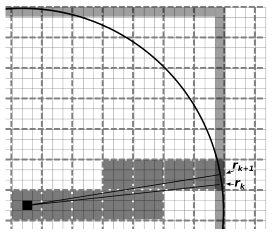
THEOREM 3.4. *Given a block B not intersected by r_0, if the* MemBlocks *size (in number of blocks) is, at least, $2\left(\frac{\rho}{\omega}+2\right)$ then the* LRU *policy will evict block B from* MemBlocks *only if it is no longer needed.*

Proof Suppose that *MemBlocks* has $2\left(\frac{\rho}{\omega}+2\right)$ slots to store the blocks. Let r_k and r_{k+1} be two consecutive rays such that r_k intersects block B. At some point during the processing of ray r_k, block B will start to be processed and it is stored in the *MemBlocks* (if r_k is the first ray intersecting block B then B will be loaded in *MemBlocks*). Now, if ray r_{k+1} also intersects block B, this block needs to be processed again. But, the *MemBlocks* size is enough to avoid block B eviction because, let $B'_1, B'_2, \cdots, B'_j$ be the sequence of blocks that need to be processed among the twice processing of B, that is, it is the sequence of blocks to be processed after B in the ray r_k and before B in ray r_{k+1}. From lemma 3.2, $j \leq 2\left(\frac{\rho}{\omega}+2\right)$ and since B is not included in the sequence then $j < 2\left(\frac{\rho}{\omega}+2\right)$. Thus, if *MemBlocks* size is $2\left(\frac{\rho}{\omega}+2\right)$ then it has slots to store all blocks that need to be processed and B will not be evicted. In other words, the *LRU* policy will not evict block B because the distinct blocks that need to be accessed can be stored in *MemBlocks*.

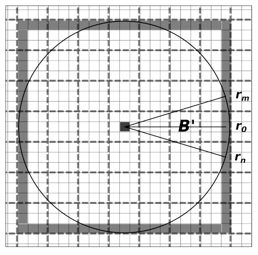
On the other hand, if ray r_{k+1} doesn't intersect block B then, from lemma 3.3, no other ray after r_k will intersect B and thus, it can be evicted since it is no longer needed.

There is a special situation for the blocks intersected by r_0 because, after being evicted, they can be loaded again when processing the last rays. But, notice that these blocks can be loaded, at most, twice. See Figure 3(b) where block B' is loaded in the processing of r_0, is evicted after the processing of r_m and it is loaded again when processing r_n.

It is possible to demonstrate that the *TiledVS* algorithm does $\theta(scan(N))$ I/O operations and takes $\theta(N)$ time to process a terrain with N cells considering that the memory can store $2\left(\frac{\rho}{\omega}+2\right)$ blocks. This complexity works even if the radius of interest ρ is large to cover the whole terrain.



(a)



(b)

Figure 3: (a) Blocks intersected by two consecutive rays; (b) Block B' is loaded because of ray r_0, is evicted after ray r_m and loaded again for ray r_n.

4. EXPERIMENTAL RESULTS

The *TiledVS* method was implemented in C++ and compiled with g++ 4.3.4. It was compared against the most efficient algorithms recently described in literature: *io-radial2, io-radial3* and *io-centrifugal*, proposed by Fishman et. al. [4], and *EMViewshed*, proposed by Andrade et al. [2].

Lacking access to Fishman's programs, we compared our algorithm to his published results. We executed our algorithm using the same datasets and also a same platform as that one used by those authors, i.e. a computer with an Intel Core 2 Duo E7500 2.93GHz processor, 4GiB of RAM memory, and a 5400RPM SATA HD (Samsung HD103SI) which was rebooted with 512MiB RAM. The operational system used was Linux, Ubuntu 10.04 32bits distribution.

Our results are presented in Table 1 and Figure 4 where we reproduce the times presented in [4]. Notice that our algorithm is faster than the others in all situations and, on huge terrains, it is about 4 times faster. Also, the table includes the processing time of our algorithm on very huge terrains generated by interpolation of Region02.

We also compared our new algorithm *TiledVS* against our previous one *EMViewshed* [2]. We used different datasets generated from two distinct USA regions sampled at different resolutions using 2 bytes per elevation value. The results are presented in Table 2. Note that our new algorithm *TiledVS* is about 7 times faster than our previous one.

Table 3 presents the *TiledVS* running time (in seconds) for different terrain sizes using only 128MiB and 512MiB of RAM. As can be noticed, our algorithm is scalable to data that is much bigger than the machine internal memory.

5. CONCLUSION

We presented a new algorithm for viewshed computation on huge grid terrains stored in external memory. Our new method uses a special data structure to manage the data

Table 1: Running time (in seconds) and CPU utilization (in parentheses) for *io-radial2*, *io-radial3*, *io-centrifugal* and *TiledVS* with 512MiB RAM.

Dataset	Size				*io-radial2*	*io-radial3*	*io-centrifugal*	*TiledVS*
	cols	×	rows	GiB				
Cumberlands	8 704	×	7 673	0.25	72 (84%)	104 (91%)	35 (49%)	17 (97%)
USA DEM 6	13 500	×	18 200	0.92	2,804 (13%)	458 (84%)	115 (57%)	85 (77%)
USA DEM 2	11 000	×	25 500	1.04	1,883 (34%)	735 (87%)	121 (54%)	98 (77%)
Washington	31 866	×	33 454	3.97	13,780 (22%)	3,008 (89%)	676 (41%)	386 (73%)
SRTM1-region03	50 401	×	43 201	8.11	37,982 (16%)	6,644 (81%)	2,845 (17%)	994 (69%)
SRTM1-region04	82 801	×	36 001	11.10		8,834 (79%)	5,341 (11%)	1,347 (69%)
SRTM1-region04	68 401	×	111 601	28.44		26,193 (61%)	12,186 (18%)	5,034 (71%)
Region02 interp.	100 000	×	100 000	37.26				5,079 (77%)
Region02 interp.	150 000	×	150 000	83.82				12,642 (72%)

transference between the internal and external memory reducing the number of I/O operations. For terrains with N cells, its I/O completixy is $\Theta(scan(N))$.

Table 2: Running time (seconds) for *EMViewshed* (EMVS) and *TiledVS* with 1024MiB RAM.

Size		EMVS	*TiledVS*
cols × rows	GiB		
30 000 × 30 000	1.68	727	256
40 000 × 40 000	2.98	3168	515
50 000 × 50 000	4.65	5701	812
60 000 × 60 000	6.71	8961	1265

Table 3: *TiledVS* running time (seconds) using a RAM memory with 128MiB and 512MiB.

Terrain Size				RAM Size	
cols	×	rows	GiB	128MiB	512MiB
37 000	×	37 000	5	634	604
52 000	×	52 000	10	1277	1168
73 500	×	73 500	20	3324	2708
104 000	×	104 000	40	7511	5612

The algorithm was compared against the most recent and efficient algorithms in the literature and, as the tests showed, it was faster than all others. In general, it was about 4 times faster and this improvement is significant because processing huge terrains can take hours. Also, it is much simpler.

Addionally, the algorithm was able to process huge terrains using few RAM memory. For example, the viewshed of a whole terrain of size 40 GiB, using 128 MiB RAM was computed in 7511 seconds.

The algorithm souce code (in C++) is avalable and distributed under Creative Common GNU GPL license at `http://www.dpi.ufv.br/~marcus/TiledVS.htm`

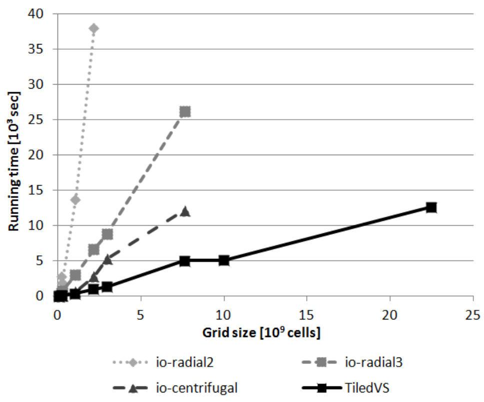


Figure 4: The running time of the four methods.

Acknowledgments

This research was partially supported by FAPEMIG, CAPES, CNPq, GAPSO and NSF grants CMMI-0835762, IIS-1117277.

6. 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] M. V. A. Andrade, S. V. G. Magalhães, M. A. Magalhães, W. R. Franklin, and B. M. Cutler. Efficient viewshed computation on terrain in external memory. *GeoInformatica*, pages 381 – 397, 2011.

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

[4] J. Fishman, H. J. Haverkort, and L. Toma. Improved visibility computation on massive grid terrains. In O. Wolfson, D. Agrawal, and C.-T. Lu, editors, *GIS*, pages 121–130. ACM, 2009.

[5] L. D. Floriani, E. Puppo, and P. Magillo. Applications of computational geometry to geographic information systems. In J. U. J. R. Sack, editor, *Handbook of Comp. Geom.*, pages 303–311. Elsevier Science, 1999.

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

[7] W. R. Franklin and C. Vogt. Tradeoffs when multiple observer siting on large terrain cells. In *12th Int. Symp. on Spatial Data Handling*, 2006.

[8] J. A. Silveira, S. V. G. Magalhães, M. V. A. Andrade, and V. S. Conceição. A library for external memory processing of huge matrix in external memory. In *XIII Brazilian Symposium on Geoinformatics.* (to appear).

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