Preprint of a published article:

Carabaño, J., Sarjakoski, T. and J. Westerholm, 2015. Efficient Implementation of a Fast Viewshed Algorithm on SIMD Architectures. In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 199–202, ISBN: 978-1-4799-8491-6, DOI: 10.1109/PDP.2015.62

Efficient implementation of a fast viewshed algorithm on SIMD architectures

Jesús Carabaño Bravo
Åbo Akademi University, Dept. of IT
Joukahainengatan 3-5, 20520 Åbo,
Finland
Jesus.CarabanoBravo@abo.fi

Tapani Sarjakoski
Finnish Geodetic Institute, Dept. of
Geoinformatics and Cartography
P.O.B 15, 02431 Masala, Finland
Tapani.Sarjakoski@fgi.fi

Jan Westerholm
Åbo Akademi University, Dept. of IT
Joukahainengatan 3-5, 20520 Åbo
Finland
Jan.Westerholm@abo.fi

***Abstract*— Viewshed refers to the land area that is visible to an observer placed in a point of a terrain. Due to the advances in remote sensing technologies the volume of data is today beyond the capability of traditional GIS tools and therefore new and fast algorithms become essential. In this paper we present an efficient implementation of the XDRAW algorithm [5] to quickly compute viewsheds on very large digital elevation models. We redesign the algorithm to make it IO-efficient and compatible with modern SIMD architectures. Our implementation is able to compute viewsheds on digital elevation models at the rate of 10^9 points per second on an Intel quad-core CPU with AVX2 technology, which makes the algorithm suitable for real-time applications.**

Keywords*— *viewshed; gis; simd; parallel; real-time

I. INTRODUCTION

A viewshed, also known as visibility map, is the subarea of a terrain that is visible to an observer placed on a particular point of the terrain at a specific altitude. The digital representation of a terrain is called Digital Elevation Model (DEM). Computing the visibility map of a DEM is a common task for any Geographical Information System (GIS) and a key process in many spatial analysis problems. Some applications of visibility maps include finding the minimal set of observers that fully covers a region [4, 10], evaluating the impact of a new construction on the view of a landscape [12] or computing optimal hidden paths [15].

The extraordinary advances in remote sensing technologies during the last years have lead to an imperative need for fast computational methods. Today, technologies like LIDAR produce ever-increasing high resolution DEMs that enable research impossible only a decade ago. This, however, also brings new challenges given that the volume of data has already surpassed the limits of traditional GIS tools. For instance, 1-meter resolution data sets for regions the size of European countries or USA states are in the order of terabytes [3], which clearly exceed the capability of former algorithms.

Most algorithmic methods for computing viewsheds were already proposed several years ago [6, 16] and most present researches are about optimizing them. We see two principal lines of research in this scope. The first studies the so-called I/O-efficient algorithms, focusing on the design of external memory algorithms and the use of cache-aware and cache-oblivious techniques [1, 8, 17]. The second line of research targets on porting, redesigning and creating new viewshed algorithms that run efficiently on parallel computers. Shared-memory multi-core computers and modern accelerators are the top technologies in this respect [13, 14, 19].

In this work we cover both lines of research by redesigning the XDRAW algorithm [5] to make it IO-efficient and compatible with modern SIMD architectures (e.g. SSE4.2, AVX2, GPUs and AVX-512 from Xeon Phi). In section II we describe XDRAW and compare it to other approaches. In section III we detail the three key improvements to the original algorithm: blocking, Morton order and code regularization with multiple code paths. Finally we report results from our implementation in section IV and our conclusions in section V.

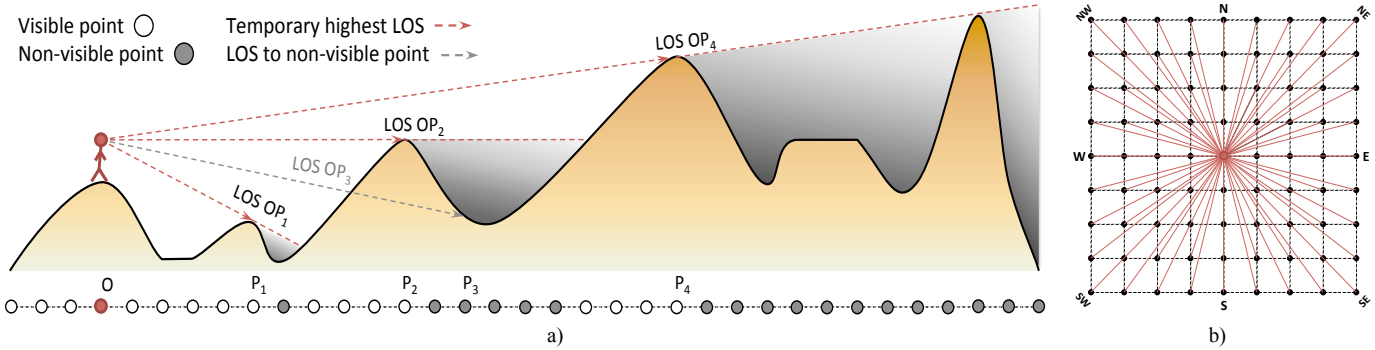
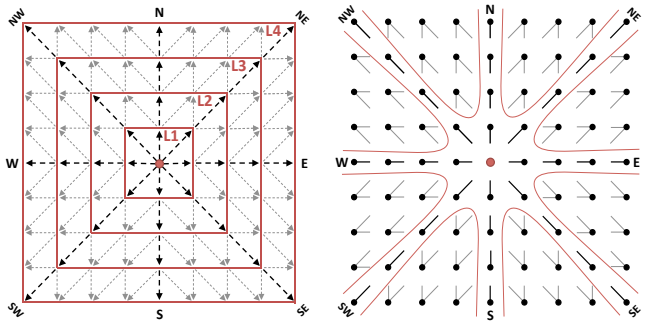


Figure 1. (a) Viewshed problem in one dimension. Each discrete one-dimensional point at the bottom stores an elevation. (b) Top view of a two-dimensional case. The observer in the center throws lines-of-sight (LOS) to all other points in the Digital elevation Model (DEM).

II. XDRAW ALGORITHM

XDRAW is commonly attributed to [5] but similar approaches were already known, used and reported [16]. It has been further studied for instance in [6]. XDRAW builds the output visibility map following an iterative expansion from the observer toward the cardinal and ordinal directions (Fig. 2a). The simplest way to understand this behavior is to look at a one-dimensional example. In Fig. 1a the visibility of each new point directly depends on whether its elevation is above or below the highest line-of-sight (LOS) discovered up to the position of the point. More formally, any point P_x is visible if the LOS OP_x from the observer O is higher than the highest accumulated LOS. A LOS is higher than another when its slope is greater than the other slope.

Notice that the described process only works if all points lie in the same vertical plane. For the two-dimensional case (Fig. 1b) all LOSs except the ones in the eight compass directions need to be walked individually. XDRAW, however, avoids this restriction by transforming the two-dimensional problem to a one-dimensional case. To do so the highest LOS affecting a point is approximated from the two LOSs accumulated until its closest preceding neighbors (as indicated by the thin short lines in Fig. 2b). In this way the processing of each element depends solely on previous computations and this leads to a computational complexity of *O(1)* per data element.



a) b)

Figure 2. (a) Flow Map and (b) Dependency Map in XDRAW.

A. High Level Description

XDRAW can be summarized by the following steps:

1. Choose the position and height of the observer O.
2. Initialize the maximum slope MS_o accumulated at O to the lowest floating-point value (-3.40282e+38).
3. Move to the next layer L_n. Every layer L_n is composed by the DEM points with distance n to the observer in the vertical, horizontal or both axis (Fig. 2a).
4. For each point P_x in the new layer L_n calculate its MS_x by interpolation of the maximum slopes MS_a and MS_b accumulated until the nearest preceding neighbors of the previous layer. The data dependency map (Fig. 2b) shows the direction of the involved neighbors. The weights applied in the interpolation are the inverses of the distances from the neighbors to the LOS OP_x. Cardinal and ordinal directions correspond to the one-dimensional case and do not need interpolation.
5. Compute the slope S_x of the LOS OP_x for each point P_x with the equation of a straight line. Set P_x to visible if S_x is greater than MS_x. In this case MS_x becomes S_x, otherwise the previous value is kept in MS_x.
6. Go to 3 while more layers remain.

B. XDRAW Versus Other Approaches

There exist three groups of viewsheds methods that possess, respectively, complexities of *O(N)*, *O(logN)* and *O(1)* per data element. The first group is composed by those algorithms that walk all LOSs and interpolate between DEM height values to obtain the real elevation crossing the LOSs. The common example of *O(N)* algorithm is R3 [6] and some IO-efficient versions of this approach are presented in [7, 19].

The second group is characterized by algorithms that accept the gridded values as close enough approximations to the real elevations lying in the trajectory of the LOSs. With this change now multiple LOSs traverse same gridded points and several strategies can be utilized to avoid repeated access. As a result the complexity is reduced to *O(logN)* in exchange of some loss of accuracy. Examples are the SWEEP-LINE method [11] and some IO-efficient versions of it [3, 8].

The last group comprises those algorithms that, like XDRAW, reduce the two-dimensional problems to the one-dimensional case (Fig. 1). This effectively decreases the complexity to *O(1)* but at the expense of even more loss of accuracy than the second set of algorithms. Other examples of algorithms from this group are R2 [6], REFERENCE-PLANE [18], BACKTRACK [9] and CENTRIFUGAL-SWEEP [3].

Finally, note that no algorithm is unconditionally better than any others. They all present very different tradeoffs between speed and accuracy and only the final application and input data will determine the suitability of the methods. In this paper our main goal is performance and hence we do not distrust the accuracy of XDRAW. This is because fast algorithms are essential in real-time application for instance, where the execution of more accurate methods would be too slow and hence useless. On the other hand, applications with high requirements of accuracy should be evaluated thoroughly first before using a fast approximate approach like XDRAW.

III. IMPROVEMENTS AND REDESING

We made three major changes to XDRAW. First we applied a blocking scheme to avoid uncoalesced accesses to memory. Secondly we orchestrated the processing of blocks with the space-filling z-curve to reduce the thrashing of memory. Finally we regularized the code with multiple execution paths so that instructions are compatible with the SIMD model. In addition we process different sectors in different threads to exploit the available cores of modern multiple-core processor.

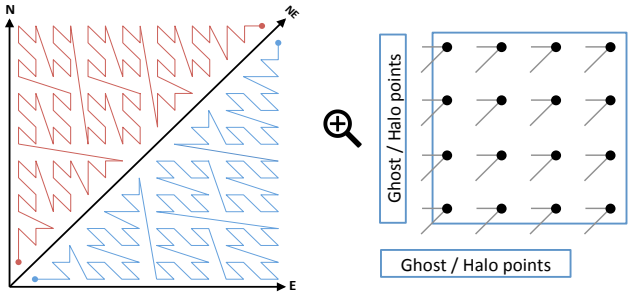
1) Blocking

Blocking, also called tiling, is a cache-aware technique that pursues a better utilization of cache memories. This technique solves the situation where XDRAW was issuing uncoalesced accesses to memory. The problem appears when the propagation of the algorithm advances perpendicularly to the

memory layout and memory elements have to be gathered in multiple memory transfers. In our improved implementation full blocks of data are loaded in a coalesced fashion whichever direction the propagation goes into and this ensures an efficient utilization of the memory resources. Besides, this technique reduces the memory requirements of the algorithm, which before required three times the size of the input data (input, output and slope buffers) and now just requires three times the block size per used thread. For the cases where the propagation is perpendicular to the memory layout it is still necessary to transpose the block. Nevertheless, this operation is significantly more efficient now that the block is loaded into cache.

2) Z-Order

Formally known as Morton encoding [2], this space-filling curve maps multidimensional data into one dimension while keeping its order and locality. Following this order when processing data can automatically lead to an efficient utilization of cache memories. This is commonly called cache-obliviousness because the effect is achieved automatically and independently of the cache configuration. This technique can also help to reduce the trashing of disk pages from main memory and improve the disk IO performance since modern operating systems always use the main memory as a large software cache for the data stored in disk.



a) b)

Figure 3. (a) Traversing order of the blocks in both NE sectors when using z-order filling curves. (b) Zoom inside a block from the NE-E sector, its data dependencies and ghost cells required from previously processed blocks.

In our implementation we apply the z-curve to orchestrate the traversal order in which blocks are processed. During the propagation of XDRAW each new layer reutilizes results computed in the previous layer. These data elements face the risk of being thrashed when following depth-first or bread-first traversal orders in very large DEMs. After applying this improvement new blocks are likely to be issued soon after their previous neighbors have been computed and this reduces the thrashing of memory. The ordering starts from the observer and grows independently for each sectors as indicated in Fig. 3a.

3) Code Paths

In XDRAW all points in a layer experience a similar condition, but depending on their relative position to the observer some points might require accessing more neighbors, performing interpolation or computing some arithmetic operations in different order. For instance, while a point in the north direction will only access the neighbor in its south, a point in the northeast-east sector will access two preceding points (Fig. 2b). These dissimilarities end up producing a heavily branched code with heterogeneous instructions and dynamic memory accesses, which go against the SIMD model.

Avoiding the aforementioned obstacle requires the code to clearly expose the parallelism. In our implementation we set up different code paths for each of the possible cases. This helps to avoid branches within the inner loops and creates more compacted and static codes. Consequently the compiler can both issue SIMD instructions and produce more efficient code. After this change blocks like the one showed in Fig. 4b will be only processed by their own code path, which is specially optimized for their specific case.

IV. BENCHMARKING

For this work we employed an *Intel(R) Core(TM) i7-4770K @ 3.50GHz* processor. It is a quad-core CPU with multithreading and AVX2 technology. The machine also included 32GB of main memory and a pair of *Samsung 840 PRO* SSD flash-disks in RAID0 mode. The operating system was *Ubuntu 14.04 LTS*. Both algorithms, the reference and our efficient version, were

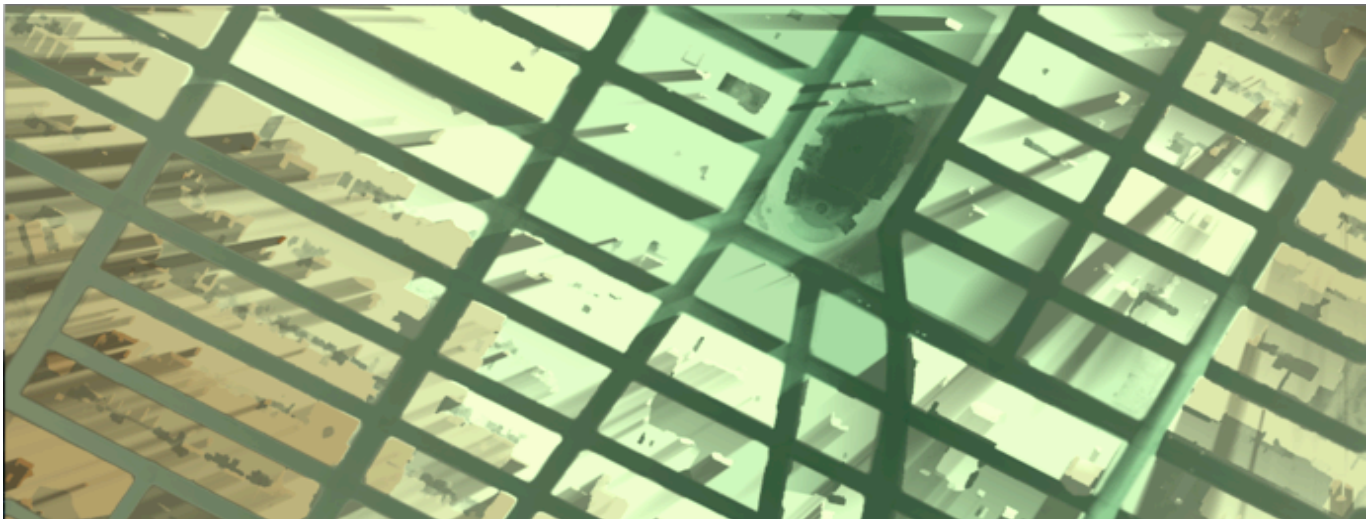


Figure 4. Viewshed on data set of a city with 1-feet resolution. The observer was set in a roof of the top-right corner of the DEM.

compiled using *GCC 4.9.1* with the following options: *Ofast, m64, march=native, mtune=native*. All the vectorized code (AVX2 instructions) was automatically generated by GCC.

The data sets used in this work were both synthetic and real data (Fig. 4). Using synthetic data is adequate because we are not measuring the accuracy of the method but its performance. XDRAW is an algorithm whose running time solely depends on the size of the input terrain hence any data configuration serves our purpose. The upper limit of the data size was imposed by the largest sample that the original XDRAW could run in our testing machine due to its memory requirements. On the other hand our efficient version is able to run any data size that fits on disk thanks to the blocking technique.

Results from our experimentation are presented in Table 1. Our efficient XDRAW was up to 50 times faster than the reference method for the largest tested data set. The given speedups result from the combination of the multiple improvements discussed in section 3. Table 2 lists the individual contribution of said enhancements. The speedup achieved through SIMD execution is reported within the *Code Paths* column since this technique is the main enabler for SIMD execution. Blocking becomes an important change, not only because it eliminates the uncoalesced accesses to memory but also because of the reduction on the memory requirements.

TABLE I. Rereference vs. efficient XDRAW.

Input Size (elevation points)	512 x 512	1024 x 1024	2048 x 2048	4096 x 4096	8192 x 8192	16384 x 16384	32768 x 32768
Reference Xdraw (seconds)	0,006	0,024	0,138	0,667	2,965	12,073	49,113
Efficient Xdraw (seconds)	0,004	0,006	0,013	0,03	0,088	0,312	0,98
Speedup (naïve / efficient)	1,5	4	10,62	22,23	33,69	38,696	50,115

TABLE II. Individual speedups of the different improvements.

Technique	**Blocking**	**Z-Order**	**Code Paths**	**Threads** (4 cores)
Speedup	4,111	1,147	2,819	3,769

V. CONCLUSIONS

In this paper we present an efficient implementation of XDRAW, a fast algorithm for the computation of viewsheds [5]. We apply several changes to the algorithm in order to make it IO-efficient and compatible with SIMD architectures. We add a blocking scheme that eliminates the uncolaesced accesses to memory; we reorder the computation with the space-filling z-curve to reduce the memory thrashing effect and we set up different code paths that clearly expose the inherent fine-grain parallelism of the algorithm and enable the use of advanced SIMD instructions. In addition we distribute the computation of the sectors of the DEM to different threads.

Our improved XDRAW is 50 times faster than the reference algorithm and is able to process 10^9 elevation points per second on a quad-core desktop processor with AVX2 technology. This makes our method a useful tool when processing viewsheds on high-resolution geographical data. For example it allows solving optimization problems like finding minimum sets of observers in reasonable times. It is moreover suitable for situations where real-time responses are desirable, like in real-time visualization. On the other hand, XDRAW presents the drawback of being less accurate than other viewshed methods and it might be a compromise to use our approach in applications with critical requirements of accuracy.

ACKNOWLEDGMENT

We thank Academy of Finland for supporting this work and the Quick-GC project with grants 259995 and 259557.

REFERENCES

[1] A. Aggarwal and J. Vitter, "The input/output complexity of sorting and related problems," *Communications of the ACM*, vol. 31, no. 9, pp. 1116–1127, Aug. 1988.

[2] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," *Theor. Comput. Sci.*, vol. 181, no. 1, pp. 3–15, Jul. 1997.

[3] J. Fishman, H. Haverkort, and L. Toma, "Improved visibility computation on massive grid terrains," in *Proceedings of the 17th ACM SIGSPATIAL GIS'09*, 2009, p. 121.

[4] W. Franklin, "Siting observers on terrain," *Adv. Spat. Data Handl.*, pp. 109–120, 2002.

[5] W. Franklin and C. Ray, "Higher isn't necessarily better: Visibility algorithms and experiments," *Adv. GIS Res. sixth International Symposium of Spatial Data Handling*, vol. 2, pp. 1–22, 1994.

[6] W. Franklin, C. Ray, and S. Mehta, "Geometric algorithms for siting of air defense missile batteries," *Research Project for Battle 2756*, 1994.

[7] H. Haverkort, L. Toma, and B. P. Wei, "On IO-efficient viewshed algorithms and their accuracy," in *Proceedings of the 21st ACM SIGSPATIAL GIS'13*, 2013, pp. 24–33.

[8] H. Haverkort, L. Toma, and Y. Zhuang, "Computing visibility on terrains in external memory," *J. Exp. Algorithmics*, vol. 13, no. 1, p. 1.5, Feb. 2009.

[9] D. Izraelevitz, "A Fast Algorithm for Approximate Viewshed Computation," *Photogramm. Eng. Remote Sens.*, vol. 69, no. 7, pp. 767–774, Jul. 2003.

[10] Y.-H. Kim, S. Rana, and S. Wise, "Exploring multiple viewshed analysis using terrain features and optimisation techniques," *Comput. Geosci.*, vol. 30, no. 9–10, pp. 1019–1032, Nov. 2004.

[11] M. Van Kreveld, "Variations on Sweep Algorithms: efficient computatin of extended viewsheds and class intervals," *Proc. 7th Int. Symp. Spat. Data Handl.*, pp. 1–14, 1996.

[12] M. Llobera, "Extending GIS-based visual analysis: the concept of visualscapes," *Int. J. Geogr. Inf. Sci.*, vol. 17, no. 1, pp. 25–48, Jan. 2003.

[13] A. Osterman, L. Benedičič, and P. Ritoša, "An IO-efficient parallel implementation of an R2 viewshed algorithm for large terrain maps on a CUDA GPU," *Int. J. Geogr. Inf. Sci.*, June 2014, pp. 1–24, May 2014.

[14] X. Shi, V. Kindratenko, and C. Yang, *Modern Accelerator Technologies for Geographic Information Science*. Boston, MA: Springer US, 2013.

[15] J. L. D. Stucky, "On applying viewshed analysis for determining least-cost paths on Digital Elevation Models," *Int. J. Geogr. Inf. Sci.*, vol. 12, no. 8, pp. 891–905, Dec. 1998.

[16] Y. Teng and L. Davis, "Visibility analysis on digital terrain models and its parallel implementation," University of Maryland, Center for Automation Research, Computer Vision Laboratory, 1992.

[17] L. Toma, "Viewsheds on terrains in external memory," In Newslett. *ACM SIGSPATIAL Special*, vol. 4, no. 2, pp. 13–17, Jul. 2012.

[18] J. Wang, G. Robinson, and K. White, "Generating viewsheds without using sightlines," *Photogramm. Eng. Remote Sensing*, vol. 66, no. 1, pp. 87–90, 2000.

[19] Y. Zhao, A. Padmanabhan, and S. Wang, "A parallel computing approach to viewshed analysis of large terrain data using graphics processing units," *Int. J. Geogr. Inf. Sci.*, vol. 27, no. 2, pp. 363–384, Feb. 2013.