An Efficient External Memory Algorithm for Terrain Viewshed Computation

CHAULIO R. FERREIRA, MARCUS V. A. ANDRADE, and SALLES V. G. MAGALHÃES,
DPI Federal University of Viçosa, Brazil
W. RANDOLPH FRANKLIN, Rensselaer Polytechnic Institute

6

This article presents **TILEDVS**, a fast external algorithm and implementation for computing viewsheds. **TILEDVS** is intended for terrains that are too large for internal memory, even more than $100,000 \times 100,000$ points. It subdivides the terrain into tiles that are stored compressed on disk and then paged into memory with a custom cache data structure and least recently used algorithm. If there is sufficient available memory to store a whole row of tiles, which is easy, then this specialized data management is faster than relying on the operating system's virtual memory management. Applications of viewshed computation include siting radio transmitters, surveillance, and visual environmental impact measurement.

TILEDVS runs a rotating line of sight from the observer to points on the region boundary. For each boundary point, it computes the visibility of all terrain points close to the line of sight. The running time is linear in the number of points. No terrain tile is read more than twice. **TILEDVS** is very fast, for instance, processing a $104,000 \times 104,000$ terrain on a modest computer with only 512MB of RAM took only $1\frac{1}{2}$ hours. On large datasets, **TILEDVS** was several times faster than competing algorithms, such as the ones included in **GRASS**. The source code of **TILEDVS** is freely available for nonprofit researchers to study, use, and extend.

A preliminary version of this algorithm appeared in a four-page ACM SIGSPATIAL GIS 2012 conference paper, "More Efficient Terrain Viewshed Computation on Massive Datasets Using External Memory." This more detailed version adds the fast lossless compression stage that reduces the time by 30% to 40%, and many more experiments and comparisons.

Categories and Subject Descriptors: F.2.2 [Nonnumerical Algorithms and Problems]: Geometrical Problems and Computations

General Terms: Design, Algorithms

Additional Key Words and Phrases: External memory algorithms, terrain visibility, geographic information systems

ACM Reference Format:

Chaulio R. Ferreira, Marcus V. A. Andrade, Salles V. G. Magalhães, and W. Randolph Franklin. 2016. An efficient external memory algorithm for terrain viewshed computation. *ACM Trans. Spatial Algorithms Syst.* 2, 2, Article 6 (June 2016), 17 pages.
DOI: <http://dx.doi.org/10.1145/2903206>

1. INTRODUCTION

Visibility, or line of sight, computation is an important component of terrain modeling in both geographical information science (GIS) and military modeling and simulation.

This research was partially supported by FAPEMIG, CAPES (Ciencia sem Fronteiras grant 9085/13-0), CNPq, and the NSF under grant IIS-1117277.

Authors' addresses: C. R. Ferreira, M. V. A. Andrade, and S. V. G. Magalhaes, Departamento de Informatica, Universidade Federal de Vicos, 36570, Vicos, MG, Brasil; emails: {chaulio.ferreira, marcus, salles}@ufv.br; W. Randolph Franklin, Electrical, Computer & Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York, 12180-3590, USA; email: wrf@ecse.rpi.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2016 ACM 2374-0353/2016/06-ART6 \$15.00

DOI: <http://dx.doi.org/10.1145/2903206>

It determines the viewshed, or set of target points, that can be seen from a given observer point [de Floriani et al. 2000; Franklin and Ray 1994; Nagy 1994]. The observer and targets might be at some given height above the terrain. Applications range from visual nuisance abatement to radio transmitter siting and surveillance. Possible goal applications include minimizing the number of cellular phone towers required to cover a region [Ben-Shimol et al. 2007; Camp et al. 1997; Bespamyatnikh et al. 2001], optimizing the number and position of guards to cover a region [Franklin and Vogt 2006; Eidenbenz 2002; Magalhães et al. 2011], analyzing the influences on property prices in an urban environment [Lake et al. 1998], and optimizing path planning [Lee and Stucky 1998]. Other applications are presented in Champion and Lavery [2002], where the term *line of sight* is used.

Currently, active visibility research potentially benefiting from faster viewshed computations includes optimizing a moving observer's path within a polygonal domain to see the target as quickly as possible [Polishchuk et al. 2016], maximizing visibility while sweeping a terrain with a set of flying observers [Efrat et al. 2013], and optimizing observer (sensor) placement in a polygonal domain when the sensors have a short battery life and cannot be on all the time [Arkin et al. 2014].

Lebeck et al. [2014] present a new method for quickly computing occluded paths over a terrain using a sparse one-dimensional network over the terrain. They describe three different strategies for constructing the sparse network and present experimental results showing that their approach is significantly faster than others, and that the different strategies offer a trade-off between higher-quality paths and lower preprocessing times.

Hurtado et al. [2014] present a pioneering theoretical study about visibility in polyhedral terrains in the presence of multiple viewpoints. They analyze the complexity and describe algorithms to compute three visibility structures: the visibility map, which is a partition of the terrain into visible and invisible regions; the colored visibility map, which is a partition of the terrain into regions whose points have exactly the same visible viewpoints; and the Voronoi visibility map, which is a partition of the terrain into regions whose points have the same closest visible viewpoint. For example, in Theorem 11 of Hurtado et al. [2014], the authors prove that the colored visibility map of a polyhedral terrain may be computed in $O(m(n\alpha(n) + \min(k_c, n^2)) \log n + mk_c)$ time, where n is the number of vertices, m is the number of viewpoints, $\alpha(\cdot)$ is the inverse Ackermann's function (for practical purposes, a constant), and k_c is the size of the colored visibility map. The max size of $k_c = \Theta(m^2 n^2)$. In contrast, the algorithm presented in this article solves a much simpler problem but has been implemented and shown to run linearly and fast when processing very large raster datasets on small machines.

Alderson and Samavati [2015] describe some techniques for speeding up line of sight algorithms. They use a terrain simplification strategy that counteracts the loss of accuracy due to simplification with some iterative methods (using residual multiresolution vectors) to try to improve the accuracy. They present a hierarchical line of sight algorithm that combines Bresenham's algorithm, a quad tree data structure and simplification.

Returning to the current work, there is the question of the future relevance of an algorithm designed to use small amounts of memory when workstations with tens of gigabytes of main memory (RAM) are available. First, as of February 2016, many widely used current computers have small amounts of RAM. The Dell Chromebook 11 has only 2GB [Dell Inc. 2016], much of which is needed by the operating system and window manager. Typical smartphones have only 2GB of RAM [Tom's Guide 2016]. Some flight simulators are 32-bit programs and thus have a total of only 3GB of addressable RAM available for the whole program. Now, combine this with recent available high-resolution terrestrial data, such as the 30m resolution terrain data from NASA's Shuttle

Radar Topography Mission (SRTM) and submeter resolution LIDAR data. Processing these large datasets on those small platforms will require external algorithms.

The computer hardware design community has also been studying the time/space trade-offs of compressing memory and cache pages [Pekhimenko et al. 2012].

Since external memory accesses are about 10^6 times slower than internal accesses [Dementiev et al. 2005], we need to minimize data transfer operations in addition to CPU time. A common model [Aggarwal and Vitter 1988] considers the cost to be the number of I/O operations, each the transfer of one disk block of size B between external and internal memory. CPU time is assumed to be comparatively insignificant (within reasonable limits).

This article extends our earlier algorithm, TILEDVS [Ferreira et al. 2012], an efficient method to compute viewsheds on terrains in external memory. TILEDVS was an adaptation of RFVS, an internal memory algorithm proposed in Franklin and Ray [1994]. TILEDVS allows efficient manipulation of large terrains. It reduces the number of disk accesses by using a custom virtual memory manager (VMM) to manage the data movement between external and internal memory. The novel component of the algorithm in this article is a fast lossless compression technique to reduce the processing by 30% to 40%. This work also reports the results of many new experiments and comparisons on various hardware configurations. Compared to the previously best published methods (EMVIEWSHED [Andrade et al. 2011] and IO_RADIAL2, IO_RADIAL3, and IO_CENTRIFUGAL [Fishman et al. 2009]), TILEDVS is much simpler and up to four times faster. TILEDVS has processed terrains with more than 20 billion points on a modest computer.

TILEDVS and its implementation are publicly available for other nonprofit researchers and educators to use and extend.

2. DEFINITIONS

We work with a region of interest $\mathcal{R}$ of the earth's geoid (i.e., the hypothetical sea-level extended also to cover the land [National Digital Elevation Program 2015]) that is small enough that the earth's curvature can be ignored. Thus, $\mathcal{R}$ can be considered to be planar. (If $\mathcal{R}$ is large enough that the curvature is significant but still small compared to the radius of the earth, then a one-time correction to elevations will remove the first-order error.) We impose a two-dimensional coordinate system (x, y) on $\mathcal{R}$, with $x, y \in \mathbb{R}$, the real numbers.

A *terrain* τ is a scalar-valued function from $\mathcal{R}$ to real numbers h called *heights*. A point on the terrain is denoted by $(x, y, h_{x,y})$. We call $(x, y) \in \mathcal{R}$ the *base* of that point. $h_{x,y}$ is the height of the point above the geoid.

There are two common data structures to represent a terrain: a triangulated irregular network (TIN) and the raster digital elevation model (DEM). A TIN is a piecewise planar triangular spline over $\mathcal{R}$. (The use of a higher-degree triangular spline has apparently not yet been studied in GIS.) The two-dimensional projection of the triangulation is usually Delaunay, which has various desirable properties, such as maximizing the minimum angles in the triangles, compared to other possible triangulations, such as minimizing the total projected edge length.

In contrast, the raster DEM partitions $\mathcal{R}$ into a square grid of cells, each with a point (or elevation post [National Digital Elevation Program 2015]) in its center and stores a height $h_{x,y}$ for each point. The meaning of $h_{x,y}$ depends on the data collection technology but is probably a convolution of the terrain's scalar height field with a spread function depending on the wavelength and effective lens size of the beam of radiation, whether light or radar.

More confusingly, the term *DEM* been used both to refer to a simple elevation array, usually rectangular, and to that array plus metadata including size and accuracy standards defined by the USGS [MindSites Group 2016]. This article does not use such metadata.

Fig. 1. Targets' visibility: T_1 and T_3 are not visible, but T_2 is.

Both TIN and DEM have advantages and disadvantages, so neither is clearly better than the other. Nevertheless, one objection to the DEM is that it uses too much space because it does not adapt to the varying information content of different regions of the terrain. However, this can be handled by postprocessing with an adaptive compression technique, such as that of Stookey et al. [2008]. Factoring the compression out into a separate step also follows good software design principles. In contrast, storing the topology (i.e., the incidence and adjacency relationships) in a TIN usually takes much more space than is required to store the elevations [Li et al. 2005], although trade-offs between space and data structure complexity are possible. In the limit, if the TIN is Delaunay, no topology needs to be stored at all since the triangulation could be rederived as needed (at some cost). There are also intermediate data structures to compactly represent planar graphs [Rossignac 1999].

An *observer* is a point from where we attempt to see other points, the *targets*. Each is defined by a base point (x, y) in $\mathcal{R}$ and a height above the corresponding terrain point. For example, the observer at point $(20, 30)$ may be 10m above the terrain at $(20, 30)$, which is itself 100m above the geoid there. We often assume that the observer can see only targets that are closer than some given *radius of interest*, ρ . In this work, ρ is large enough to cover the entire terrain. In practice, we bound the circle with a square of side $2\rho + 1$, equivalent to using an L^∞ distance metric. For convenience, this distance is measured in the plane in $\mathcal{R}$ because that is simpler and effectively the same as measuring exactly in three dimensions. Apart from that, target T is visible from O if and only if the straight line, the line of sight, from O to T is always strictly above τ (Figure 1).

The *viewshed* of O is the set of all base points corresponding to targets that can be seen by O . Thus, if the target height $h_t = 2$, the viewshed of O would represent the set of the terrain points where 2m tall people would be visible from O . We represent the viewshed by a square $(2\rho + 1) \times (2\rho + 1)$ bitmap centered on the observer, with 1 indicating that the associated target is visible.

Both terrain models are lossy and store elevation data at only a finite number of points. When an elevation is needed at a point that is not stored, some interpolation rule is required. (With other representations, such as Fourier series or wavelets, the interpolation rule is part of the representation. However, the representation is still lossy but with the lost information expressed in a different format, such as missing high frequency components. It is not clear that such a terrain representation would be an improvement.)

In any case, what is an appropriate rule is an open topic, but it must be considered when determining visibility. A line of sight rarely goes exactly above a terrain point but almost always runs between them (Figure 2). Even if the original dataset is a soup of LIDAR points, some notion of what those points imply about the underlying continuous surface (i.e., some interpolation rule) is required.

Our interpolation rule will follow Ray [1994]. In other words, for lines with absolute slope less than 1, for each integral value of x along the line, we use the elevation of the closest point in $\mathcal{R}$; otherwise, for each integral value of y along the line, we use the

Fig. 2. The rasterization of the line of sight projection.

elevation of the closest point in $\mathcal{R}$. We present more about this in Section 3.2. There are other alternatives, so different viewshed algorithms obtain slightly different results.

3. RELATED WORK

3.1. Viewshed Algorithms

A raster DEM can be processed by Van Krevelde's algorithm [Van Krevelde 1996] and RFVS [Franklin and Ray 1994]. These two algorithms are very efficient—that is, they are economical in their use of resources such as computing time as the size of the input grows. Both have been recently extended to efficient external memory viewshed algorithms. An adaptation of Van Krevelde's algorithm was presented in Fishman et al. [2009], and an adaptation of RFVS was presented in Andrade et al. [2011]. The next sections will describe these two algorithms.

3.1.1. The RFVS Algorithm. RFVS [Franklin and Ray 1994] is an efficient algorithm whose running time is $\Theta(n)$, where $n = \rho^2$. In other words, it runs in linear time in the number of points within the radius of interest. The process goes as follows:

- (1) Consider a square box (region of interest) of side $2\rho + 1$ points centered on the observer.
- (2) Order the 8ρ points on its perimeter by their angle relative to the observer, which is at $(0,0)$: $(\rho, 0), (\rho, 1), (\rho, 2), \dots, (\rho, \rho - 1), (\rho, \rho), (\rho - 1, \rho), \dots, (\rho, -1)$.
- (3) Rotate a line of sight λ from the observer to each of these 8ρ targets t in turn. For each λ , do the following:
 - (a) Determine the ordered list P of points $p_0, p_1, \dots, p_k = t$ that are sufficiently close to λ . For target p_i with $|y_i/x_i| \leq 1$, those are points differing in y by $\leq 1/2$. For the other targets, use the difference in x . Many points (mainly those close to the observer) are processed more than once. On average, each point is processed twice since there are 8ρ rays and each ray has ρ points, so $8\rho^2$ points are processed in all, whereas the terrain has $4\rho^2$ points. Use any line rasterization algorithm.
 - (b) Compute the vertical slope α_i of the line from the observer to the base point of each p_i as follows, where the observer's point is $(0, 0, h_0)$, its height above the terrain is h_o , the target's point is (x_i, y_i, h_i) , and its height above the terrain is h_t :

$$\alpha_i = \frac{h_i - h_{00} - h_o}{\sqrt{x_i^2 + y_i^2}}.$$

- (c) Iterate along P updating μ_i , the maximum slope seen so far. Initially $\mu_0 = -\infty$, then $\mu_i = \max(\mu_{i-1}, \alpha_i)$.

- (d) p_i is visible if $h_i + h_t \geq \mu_i \times \sqrt{x_i^2 + y_i^2} + h_{00} + h_o$ —that is, if it is not below the line with the highest angle seen so far. If a point is processed more than once, report it as visible if it is visible on any line of sight.

Each point within a square within the radius of interest is processed twice (on average), at a constant time per point, and so the execution time is linear in n , the number of points within the radius of interest.

3.1.2. Van Krevelde's Algorithm. Van Krevelde's fast viewshed algorithm [Van Krevelde 1996] runs in $\Theta(n \log n)$ time. It rotates a sweep line λ around the observer that computes the visibility of each point as λ passes over it. It maintains a balanced binary tree (the *agenda*) to store the slopes of the lines from the observer to the center points of the cells currently being intersected by λ , keyed by their distance from the observer. When λ passes over the center point of a cell c , the agenda is searched to check its visibility. In more detail:

- (1) Three types of events are defined for each c : *enter*, *center*, and *exit*, which indicate, respectively, when λ starts intersecting c , passes over c 's center point, and stops intersecting c . The algorithm maintains a list E of these events for all c inside the region of interest, sorted by their center points' azimuth angle.
- (2) The algorithm then sweeps E , processing each type of event as follows:
 - (a) *Enter*: Insert c into the agenda.
 - (b) *Center*: Search the agenda for any cell closer to the observer than c that has slope greater or equal to the slope of the line of sight to c . Thus, c is visible if and only if no such cell is found.
 - (c) *Exit*: Remove c from the agenda.

The agenda performs insertions, removals, and queries in $\Theta(\log n)$ time, making the algorithm's time $\Theta(n \log n)$. As ρ increases, the cost of maintaining the ever deeper balanced binary tree grows, and thus this algorithm's performance becomes relatively worse and worse compared to RFVS.

3.2. Comparative Accuracies

The preceding two algorithms solve the same problem, albeit with different efficiencies and approximate visibility models. Both approximate the terrain as piecewise horizontal. Van Krevelde assumes a square cell around each point, whereas RFVS assumes a more complicated shape. Since computed viewshed is only an approximation whose precision is dependent upon the resolution and accuracy of the terrain representation [Fisher 1993; Goodchild and Lee 1989; Lee et al. 1992], neither is clearly better, but RFVS is faster.

We might compute visibility with a more sophisticated terrain approximation model that interpolates the elevation between adjacent points. In that case, changing the interpolation rule from min to interpolate to max may cause the computed visibility to change for one-half of all the targets [Kaucic and Zalik 2002]. In a test against reality, Maloy and Dean [2001] compared predicted viewsheds (computed using different methods and different datasets for 11 viewpoints) with actual viewsheds from field surveys. They showed that accuracy can vary widely depending on the data resolution and the visibility criteria.

Fisher [1996] also showed that the standard binary viewshed is very sensitive to multiple decisions made during algorithm design. They include the definitions of the following essential items: how the elevation is inferred (grid, triangular, grid constraint, or stepped); how the observer and target locations are defined (point to point, cell to point, point to cell, or cell to cell); how the elevations are computed (height, gradient, or

Table I. Accuracies of Various Viewshed Algorithms

Algorithm	f_v	f_i
VIZ-ITER	0.1%	4.1%
IO-RADIAL3	53.3%	13.9%
IO-CENTRIFUGAL	7.6%	32.9%
TILEDVS	6.9%	7.2%

Note: Data assembled from Haverkort et al. [2013].

integer height); the interpolation rule used (linear interpolation, average, or something else); and, for raster DEMs, the rasterization method.

Haverkort et al. [2013] report a detailed comparison of accuracies for various algorithms to compute viewsheds on two 500×500 terrains, one real and one synthetic. They use the GRASS module `r.los` as a reference. Then they pick a number of topologically interesting observers on valleys and ridges, and they count how many points' visibilities are computed differently from the reference. However, they generally do not report running times, except asymptotically. `VIS-ITER`'s time is $\Theta(n \log n)$. Haverkort et al. [2013] report that on the very large SRTM region 6 data, `TILEDVS` is 2.5 times faster than `VIS-ITER`. Table I summarizes numbers extracted from it, where $f_v \triangleq$ *false visible* (and, respectively, $f_i \triangleq$ *false invisible*) are the percentage of points that are not visible (visible) in `r.los` and visible (not visible) in the algorithm under evaluation. `IO-RADIAL` and `IO-CENTRIFUGAL` are quite inaccurate. `TILEDVS` has a 15.1% difference from the reference implementation compared to `VIS-ITER`'s 4.2%. If the reference implementation is taken to be correct, then there is a trade-off between speed and accuracy. However, given how many points' visibility can be changed by minor changes in the elevation interpolation, more study is required to learn whether this is statistically significant, and whether it extends to other terrains or to lines of sight more than 500 points long.

3.3. External Memory Viewshed Algorithms

Since the viewshed algorithms described earlier are not well suited for external memory, the following I/O-efficient versions have been created.

3.3.1. *EMViewshed*. Andrade et al. [2011], based on `RFVS`, maintains an external memory list Q describing the terrain points, sorted by the order that they will be processed by `RFVS`. Sweeping that list sequentially avoids random external memory accesses. Q contains pairs (p, k) , where p is a terrain point and k is the order in which it will be processed. Points that need to be processed more than once will have several entries in Q .

`EMVIEWSHED` creates the entries in Q and then orders them with an external sort [Dementiev et al. 2005]. Then it sweeps Q to calculate the viewshed, making only sequential accesses to external memory. This is quite I/O efficient, but our new algorithm, shown later, is more than 10 times faster.

3.3.2. *Algorithms Based on Van Kreveld's Algorithm*. Van Kreveld's method was adapted for external memory processing by various authors using different strategies. `IOVIEWSHED` [Haverkort et al. 2009] creates a list of events and sorts them externally. These events are used to process the terrain with a sweep-line approach. This algorithm was renamed `IO-RADIAL1` in Fishman et al. [2009], which additionally describes two other algorithms also based on Van Kreveld: `IO-RADIAL2` and `IO-RADIAL3`. They sweep the terrain by rotating a ray around the viewpoint while maintaining the terrain profile along the ray (similar to Van Kreveld). The difference between the two algorithms lies in the preprocessing step before sweeping the terrain. In `IO-RADIAL2`, the grid points are sorted

Fig. 3. Partitioning the elevation array into tiles and reorganizing the points in external memory to store the points of each tile in sequence. The arrows indicate the writing sequence.

into concentric bands around the viewpoint, whereas in IO-RADIAL3, the grid points are sorted into sectors around the viewpoint. Fishman et al. [2009] describe another algorithm, IO-CENTRIFUGAL, which is not based on Van Kreveld. Instead, it sweeps the terrain centrifugally, growing a star-shaped region around the observer while maintaining an approximate visible horizon of the terrain within the swept region.

IOVIEWSHED (or IO-RADIAL1) was implemented as an add-on to GRASS (versions 6.x/7.x), named R.VIEWSHED [Toma et al. 2010]. It is much slower than IO-RADIAL2 and IO-RADIAL3 [Fishman et al. 2009], which is the fastest among these three methods but slower than IO-CENTRIFUGAL. Our new algorithm is about eight times faster than IO-RADIAL3 and four times faster than IO-CENTRIFUGAL.

4. TILEDVS ALGORITHM

4.1. Algorithm Description

Here we present a new external memory viewshed algorithm that improves on TILEDVS and RFVS. Recall that RFVS sweeps the terrain by rotating a line of sight that connects the observer to points on the boundary of a square region of interest. For each line of sight, the points along it are processed in order to compute their visibility.

As each line of sight is processed, there is two-dimensional locality of reference: each new point is adjacent in E^2 to the previous one. However, with the usual row-major order for array storage, points in adjacent rows are often in different disk blocks so that each block might need to be read many times. Even if the terrain fits in internal memory, the row-major order might be inefficient because of poor cache locality.

TILEDVS mitigates this problem by using a custom external memory management library named *TiledMatrix* [Silveira et al. 2013]. This library subdivides the big array of points into small square tiles of $\omega \times \omega$ points each, which are then stored in external memory following the pattern shown in Figure 3. To access a given point, it loads the whole tile containing that point into internal memory. Since the block size is much smaller than the internal memory size, *TiledMatrix* keeps a cache of several of these tiles in internal memory at any time, managed with a least recently used (LRU) replacement policy. This cache is named *MemTiles*.

When a tile is accessed, it is labeled with a timestamp, and when necessary to evict a tile from the cache to load a new one, the tile with the smallest timestamp is chosen. When a tile is evicted, it is checked for whether it has been updated, and if so, it is written back to disk.

The tile size is chosen to be several times the physical disk block size. This facilitates our next I/O optimization: using the fast lossless compression algorithm LZ4 [Collet

2012] to compress the tiles before writing to disk and to uncompress them after reading. For file management simplification, the space reserved for each tile on the disk is the original uncompressed size. However, when a tile is transferred, only its compressed size (recorded in an auxiliary array) is transferred. Experimentally, this reduces the tile size by 44% on average, which reduces the amount of I/O.

Each tile is loaded in the cache, kept there while being accessed, and evicted when it is no longer needed. With the memory size assumptions described later, most of the tiles are read only once. The tiles in a line to the right of the observer are read twice.

A further optimization would be possible with a filesystem, such as *ext4* in Linux, that supports a sparse mode. Here, a block in a file is not reserved on the disk until the block is written. (One application is large sparse database hash tables.) In this case, when a tile is considerably compressed, the unused disk blocks are not only not written but do not even take space on the disk.

4.2. TILEDVS Analysis

Assume without loss of generality that the slope of a line of sight is not greater than 1. (Otherwise, swap x and y .) Remember that a line of sight, from the origin to a border point on the square region of interest, contains $\rho + 1$ points, one per y -value, and that each square tile contains $\omega \times \omega$ points. A line of sight crosses through at most two tiles in each column of tiles, except for one tile in the column at the observer. Therefore, it will contain points from at most

$$2 \left\lceil \frac{\rho + 1}{\omega} \right\rceil + 1$$

tiles. Defining a *word* as the storage required by one point (probably 2 or 4 bytes), the internal memory required to store the tiles for one line of sight is

$$\left(2 \left\lceil \frac{\rho + 1}{\omega} \right\rceil + 1 \right) \omega^2$$

words. If the memory size is not smaller than this value, then each tile will be read only once since a tile is read into memory when it is first needed and evicted only after the line of sight has rotated past all of its points. The exception is the line of tiles covering the first line of sight, which extends in the positive x direction from the observer. Therefore, the total number of tiles read is

$$\left\lceil \frac{2\rho + 1}{\omega} \right\rceil^2 + \left\lceil \frac{\rho + 1}{\omega} \right\rceil + 1.$$

TILEDVS's memory requirements can be easily satisfied for two reasons. First, as stated in Demaine [2002], it is common to assume the computers have a “tall cache”—that is, the number of blocks can fit in memory is larger than the block size ($\frac{M}{B} \geq cB$, for a given $c \geq 1$). Second, the tile size is chosen to be a relatively small multiple of the number of disk blocks, and the number of tiles that need to be kept in the memory is small if compared to the size of the terrain.

For example, a viewshed with $\rho = 250,000$ would have $500,001 \times 500,001$ points within the radius of interest, requiring 500GB before compression if each point uses 2 bytes. Choosing $\omega = 250$, each tile would take 125KB before compression. TILEDVS needs to keep $2 \left\lceil \frac{\rho + 1}{\omega} \right\rceil + 1 = 2,003$ tiles in memory, which requires only $2\omega^2(2 \left\lceil \frac{\rho + 1}{\omega} \right\rceil + 1) = 250\text{MB}$, which is a small fraction of the available memory on even modest current computers.

A machine with 1GB of internal memory could process a terrain with a viewshed radius of interest $\rho = 1,000,000$ points, or 10^{12} points total, requiring an external memory of 8TB before compression.

Thus, **TILEDVS** can be classified as a cache-aware algorithm [Frigo et al. 2012], as its efficiency depends on the tile size whose value is defined based on the internal memory size available.

5. TILEDVS COMPLEXITY

5.1. I/O Complexity

TILEDVS uses two external memory arrays: the elevation array *Elev* and the viewshed array *V*. Initially, **TILEDVS** reads the terrain and initializes *Elev*. Then the two arrays are accessed during the viewshed computation, and finally, the array *V* is written to the output file.

From **RFVS**, the arrays' dimension is $(2\rho + 1) \times (2\rho + 1)$, and each array will be divided into at most $\lceil \frac{2\rho+1}{\omega} \rceil^2$ tiles, each with $\omega \times \omega$ points. As mentioned in Section 4.2, each array has a *MemTiles* data structure with at least $2(\frac{\rho}{\omega} + 2)$ tiles, with each tile requiring several disk blocks.

In the first step, array *Elev* is initialized with the point elevation values and subdivided into tiles that are stored in external memory. Since the *MemTiles* data structure has enough slots to store all of the tiles in ω array rows, the *Elev* array initialization, subdivision, and writing can be done using a standard row-major sweep. Since the array *Elev* has $\lceil \frac{2\rho+1}{\omega} \rceil$ rows of tiles, the whole process reads and writes $\lceil \frac{2\rho+1}{\omega} \rceil^2$ tiles. Therefore, this first step performs $\Theta(\text{scan}(\rho^2))$ I/O operations.

During the viewshed computation, *Elev* array's tiles are read at most twice and never written. The number of tiles read is $\Theta((\rho/\omega)^2)$. The viewshed array *V*'s tiles are each read and written at most twice, for a total of $\Theta((\rho/\omega)^2)$ tiles read and written. The I/O cost is $\Theta(\text{scan}(\rho^2))$, or $\Theta(\text{scan}(n))$ since there are n points within the radius of interest of the observer.

5.2. CPU Complexity

The array of points within the radius of interest of the observer contains $(2\rho + 1)^2$ points, with 8ρ perimeter points, unless the observer is near the border, in which case it is smaller. **TILEDVS** shoots 8ρ rays, each with at most $\rho + 1$ points, and thus will process at most $8\rho^2 + 8\rho$ points. Since there are $(2\rho + 1)^2$ distinct points, each point will be processed $\frac{8\rho^2+8\rho}{(2\rho+1)^2} < 2$ times on average. Therefore, **TILEDVS** takes linear time in the number of points and thus is asymptotically optimal since each point needs to be read at least once.

6. EXPERIMENTAL RESULTS

We implemented **TILEDVS** in C++, compiled it with g++ 4.3.4, and compared it against the most efficient recently published algorithms: **IO-RADIAL2**, **IO-RADIAL3**, and **IO-CENTRIFUGAL**, proposed in Fishman et al. [2009], and **EMVIEWSHED**, proposed in Andrade et al. [2011]. We also assessed its performance in small memory environments, the influence of compressing the tiles, and the effect of using the operating system's VMM instead of *TiledMatrix* for managing the tiles.

In all experiments, the running time corresponds to the average elapsed time (not the CPU time) of five executions and includes the total time taken by the whole process—that is, loading the data in its original format, partitioning the grid, writing the data into tiles, processing the terrain, and writing the output file (the viewshed) in row-major order. **TILEDVS** automatically optimizes ω from the terrain and memory size to

Fig. 4. Subdivision of the SRTM datasets into regions. Source: NASA SRTM.

maintain the minimum requirements for the number of blocks that needs to fit in memory.

These results improve on Ferreira et al. [2012] by 30% to 40%, mostly because of the new fast lossless compression strategy. There are also smaller improvements, such as using buffer arrays for reading the input and writing the output.

Except when noted otherwise, our experiments used real USA terrain datasets from Rabus et al. [2003], sampled at approximately 30m resolution. To obtain comparable results with other implementations, the terrain files use one 4-byte integer elevation per point, although 2 bytes would suffice given the data accuracy (and would halve the I/O). Within the datasets, the regions are subdivided and numbered according to Figure 4.

6.1. Comparing TILEDVS to the Algorithms of Fishman et al.

We compared TILEDVS to the published results in Fishman et al. [2009], as we lacked access to that code. We used the same datasets and took great pains to match the compute platform as closely as possible. Although they use HP 220 blade servers, each with an Intel Xeon 2.83GHz processor and a 5400RPM SATA hard drive, we used a computer with an Intel Core 2 Duo E7500 2.93GHz processor, 4GB of RAM memory, and a 5400RPM SATA hard drive (Samsung Seagate ST1000LM024 1TB). The computer was rebooted with 512MB of RAM. The operating system used was Linux, 12.04 64-bit Ubuntu. Although these two platforms are very similar, our CPU is a little slower, according to the benchmarks in Passmark Software [2013].

To be as fair as possible, we also performed additional tests of TILEDVS with a 5400RPM external hard drive with a USB 2.0 interface (Samsung M3 HXM101TCB-G), which is very likely slower than the internal hard drive used in the experiments of Fishman et al.

Like our algorithm, the algorithm of Fishman et al. apparently requires a tall cache. However, their algorithm is cache oblivious (it does not need to know the size of the memory or block), whereas we need to know the memory size. This makes their algorithm more general and ours more tunable.

Our results are presented in Table II, which reproduces the values presented in Fishman et al. [2009], with two additional columns with the results from TILEDVS, using both types of hard drives. Here we used datasets from the same regions used by Fishman et al. [2009] used, such as some of the regions shown in Figure 4 and datasets of Cumberland and Washington regions. We also extended the table to include the

Table II. Running Times in Seconds of the Algorithms of Fishman et al., IO-RADIAL2 (IO-R2), IO-RADIAL3 (IO-R3), and IO-CENTRIFUGAL (IO-CENT), and of Our Algorithm, TILEDVS, Using an Internal (int.) and a USB External (ext.) Hard Disk

Dataset	Terrain Size			Algorithms of Fishman et al.			TILEDVS	
	Columns	Rows	GB	IO-R2	IO-R3	IO-CENT	(int.)	(ext.)
Cumberland	8,704	7,673	0.25	72	104	35	17	25
USA DEM 6	13,500	18,200	0.92	2,8044	458	115	86	103
USA DEM 2	11,000	25,500	1.04	1,883	735	121	112	120
Washington	31,866	33,454	3.97	13,780	3,008	676	374	449
SRTM1-reg03	50,401	43,201	8.11	37,982	6,644	2,845	828	893
SRTM1-reg04	82,801	36,001	11.10	—	8,834	5,341	1,226	1,268
SRTM1-reg04	68,401	111,601	28.44	—	26,193	12,186	3,215	3,478
Reg02 interp.	150,000	91,000	50.85	—	—	—	5,577	6,397
Reg02 interp.	200,000	122,000	90.89	—	—	—	10,638	17,058

Note: In all experiments, the memory size was 512MB.

Fig. 5. Comparing the running times of the four methods.

processing time of TILEDVS on very large terrains generated by interpolation of the Region02 dataset (see the last two rows). The results show that TILEDVS is faster than the others in all situations, and on large terrains it is about four times faster (see the processing time for SRTM-region04).

Even when using the slow external hard drive, TILEDVS is faster than the algorithms of Fishman et al. This suggests that TILEDVS performs significantly fewer external memory accesses. The results from Table II are plotted in Figure 5, showing that the bigger the terrain, the better TILEDVS is in comparison to the other methods.

Figure 6 plots the number of points processed per second versus the terrain size. All methods asymptotically process a constant number of points per second. Again, TILEDVS is the fastest by a factor of about 4.

Finally, once the terrain has been stored as tiles, it may be reread multiple times to compute other viewsheds with different observers and parameters such as observer and target heights and radius of interest.

6.2. Comparing TILEDVS to EMVIEWSHED

We also compared our new algorithm TILEDVS against our previous one, EMVIEWSHED [Andrade et al. 2011], using different datasets generated from two USA regions (02 and

Fig. 6. Number of points processed per second by each method.

Table III. Running Times (Seconds) for *EMViewshed* (EMVS) and *TILEDVS* with 512MB of RAM

Terrain Size			EMVS	TILEDVS
Columns	Rows	GB		
10,000	10,000	0.37	46	27
20,000	20,000	1.49	1,464	129
30,000	30,000	3.35	4,295	317
40,000	40,000	5.96	9,970	575
50,000	50,000	9.31	17,368	883

Table IV. *TILEDVS* Running Times (Seconds) with Either 128MB or 512MB of RAM

Terrain Size			RAM Size	
Columns	Rows	GB	128MB	512MB
37,000	37,000	5	490	481
52,000	52,000	10	1,191	993
73,500	73,500	20	2,711	2,238
104,000	104,000	40	7,439	5,271

03) sampled at different resolutions. Table III presents the results. *TILEDVS* is about 10 times faster than *EMVIEWSHED*, processing 2.8M points per second in the largest case.

6.3. *TILEDVS* in Small Memory Environments

Table IV presents the *TILEDVS* running times (in seconds) for different terrain sizes using small internal memory sizes: 128MB and 512MB. *TILEDVS* is able to process datasets that are far too large to fit into internal memory. For example, it was able to process a 40GB terrain using only 128MB of RAM in 7,439 seconds. The extra memory is less beneficial than might be expected because *TILEDVS*'s working set size is so small.

6.4. The Influence of Compression

Silveira et al. [2013] report several experiments concerning *TiledMatrix*'s performance when compressing the tiles to reduce the amount of data needing to be transferred to/from disk. The compression is fast enough that the time penalty for the compression is small. In addition, decompression is faster than compression, and the data is read a little more often than it is written.

Table V. Speedups and Compression Ratios Obtained When the Compression Strategy Is Used for Real and Artificial Terrains

Terrain Size			Terrain Type					
Columns	Rows	GB	Real		Flat		Random	
			Speedup	Ratio	Speedup	Ratio	Speedup	Ratio
10,000	10,000	0.37	1.28	3.06	2.17	250.22	1.00	1.08
20,000	20,000	1.49	1.61	3.39	2.07	252.49	1.01	1.20
30,000	30,000	3.35	1.56	3.61	2.16	253.22	1.02	1.22
40,000	40,000	5.96	1.21	2.22	1.89	253.58	1.04	1.23

Note: Each terrain point to be compressed contains a 4-byte elevation plus a 1-byte visibility.

To test this on **TILEDVS**, we compared the earlier version of **TILEDVS** presented in Ferreira et al. [2012] (without compression) with our new implementation (with compression). Each terrain point of the data being compressed consists of a 4-byte elevation and a 1-byte visibility.

The results were quite favorable. We tested both real terrains from SRTM datasets (region 02) and two types of artificially generated terrains: “flat” terrains (where all elevation values are zero) and “random” terrains, where all elevation values are random. These two types of terrains represent the best and the worst theoretical scenarios for data compression. Table V presents the results, including the speedups obtained by using compression and the data compression ratios (uncompressed size/compressed size). The random terrain is compressible by about 20% because the included computed viewshed array compresses very well.

As expected, the performance of the real terrains lies between that of the worst and best artificial terrains.

6.5. *TiledMatrix* Compared Against the OS’s Virtual Memory System

Since **TILEDVS** uses a straightforward implementation of the LRU caching strategy also used by many operating systems, the obvious suggestion is that similar running times could be obtained by reorganizing the data into tiles (as shown in Figure 3) and allowing the OS VMM to manage data swapping. We tested this with an implementation (named *VMM_VS*) that subdivides and reorganizes the terrain array but does not manage the data accesses, letting the Linux VMM do that. We compared this implementation against the old version of **TILEDVS** (with no compression strategy).

First, the terrain was subdivided into tiles with $1,000 \times 1,000$ points, and execution times were compared using *VMM_VS* and **TILEDVS** for a terrain with $30,000^2$ points. The result was that *VMM_VS* required 1,909 seconds, whereas **TILEDVS** executed in only 350 seconds.

We believe the main reason why **TILEDVS**’s custom VMM is faster than the OS VMM is because **TILEDVS** knows the tile size, and thus it can read an entire tile by performing only one seek operation and then transferring all the tile data to the memory. The VMM, on the other hand, may need to perform several I/O operations to read a tile. In other words, since a VMM page has 4,096 bytes, a $1,000 \times 1,000$ point tile will use 977 pages. Therefore, when the VMM reads one page of a tile, it will probably not read all of the pages. However, **TILEDVS** reads the points in a tile repeatedly in various diagonal directions as it rotates the line of sight. Therefore, the VMM will need to read the pages in each tile repeatedly. Thus, in this application with this data access pattern, a custom VMM beats the OS one.

The obvious response is for **TILEDVS** to reduce ω to 32 so that one tile fits in one virtual memory page, and also to ensure that things are properly aligned so that tiles do not straddle page boundaries. In this case, *VMM_VS* is faster, running in 1,398

seconds, whereas TILEDVS requires 4,381 seconds. However, both of these times are much worse than TILEDVS with a larger, multiblock, tile size.

Indeed, Silveira et al. [2013] show that *TiledMatrix*'s performance decreases with small tiles, because transferring small, randomly located chunks of data does not amortize the disk seek and latency times. Additionally, the disk's internal cache cannot optimize this access pattern. When the tile size is increased to $1,000^2$ points, TILEDVS's performance improves, but the performance of *VMM_VS* becomes worse since each tile requires several pages, which are loaded separately by the VMM.

Fishman et al. [2009] found similar results, concluding that “one of our findings is that relying purely on VMM, even for a theoretically I/O-efficient data access, is slow,” and “by telling the algorithms explicitly when to load a memory-size block (and not using the VMM), we obtained significant speedups (without sacrificing I/O-efficiency for the levels of caching of which the algorithm remained oblivious, and without sacrificing CPU-efficiency).”

7. CONCLUSION AND FUTURE WORK

We have presented TILEDVS, a new algorithm for viewshed computation on large grid terrains stored in external memory, which is an extension of the internal memory algorithm RFVS [Franklin and Ray 1994]. TILEDVS uses a special data structure to manage the data transfer between internal and external memories, thereby reducing the number of I/O operations. For terrains with n points, its I/O complexity is $\Theta(\text{scan}(n))$. It also reinforces the point that a custom VMM can outperform the operating system's one.

The main contributions of this article, compared to Ferreira et al. [2012], are the use of a lossless compression technique that was able to reduce the processing time by 30% to 40% and reports on many new experiments, including on various hardware configurations.

The results show that TILEDVS is the fastest viewshed computation algorithm by a factor of about 4. It is also simpler and requires small amounts of internal memory. For example, the viewshed of a 40GB terrain was computed in 7,439 seconds using 128MB of RAM.

Finally, TILEDVS's source code (in C++) is freely available under the GNU General Public License for other researchers and teachers to use and to extend at <http://www.dpi.ufv.br/~marcus/TiledVS.htm>.

ACKNOWLEDGMENTS

We would like to thank Mauricio Gruppi for helping with the experiments.

REFERENCES

- Alok Aggarwal and Jeffrey S. Vitter. 1988. The input/output complexity of sorting and related problems. *Communications of the ACM* 31, 9, 1116–1127.
- Troy Alderson and Faramarz Samavati. 2015. Optimizing line-of-sight using simplified regular terrains. *Visual Computer* 31, 4, 407–421. DOI: <http://dx.doi.org/10.1007/s00371-014-0936-3>
- Marcus V. A. Andrade, Salles V. G. Magalhães, Mirella A. Magalhães, W. Randolph Franklin, and Barbara M. Cutler. 2011. Efficient viewshed computation on terrain in external memory. *Geoinformatica* 15, 2, 381–397. DOI: <http://dx.doi.org/10.1007/s10707-009-0100-9>
- Esther M. Arkin, Alon Efraty, and Joseph S. B. Mitchell. 2014. Hybrid algorithms for scheduling sensors for guarding polygonal domains. In *Proceedings of the 30th European Workshop on Computational Geometry (EuroCG'14)*.
- Yehuda Ben-Shimol, Boaz Ben-Moshe, Yoav Ben-Yehezkel, Amit Dvir, and Michael Segal. 2007. Automated antenna positioning algorithms for wireless fixed-access networks. *Journal of Heuristics* 13, 3, 243–263.

- Sergei Bespamyatnikh, Zhixiang Chen, Kanliang Wang, and Binhai Zhu. 2001. On the planar two-watchtower problem. In *Proceedings of the 7th International Computing and Combinatorics Conference*. 121–130.
- Richard J. Camp, David T. Sinton, and Richard L. Knight. 1997. Viewsheds: A complementary management approach to buffer zones. *Wildlife Society Bulletin* 25, 3, 612–615.
- Danny C. Champion and John E. Lavery. 2002. Line of sight in natural terrain determined by L_1 -spline and conventional methods. In *Proceedings of the 23rd Army Science Conference*.
- Yann Collet. 2012. Extremely Fast Compression Algorithm. Retrieved May 20, 2016, from <https://github.com/Cyan4973/lz4>.
- Leila de Floriani, Paola Magillo, and Enrico Puppo. 2000. Applications of computational geometry to geographic information systems. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia (Eds.). North-Holland, Amsterdam, 333–388. DOI: <http://dx.doi.org/10.1016/B978-044482537-7/50008-5>
- Dell Inc. 2016. Dell Chromebook 11. Retrieved May 20, 2016, from http://www.dell.com/us/p/chromebook-11-3120/pd?ref=PD_OC.
- Erik D. Demaine. 2002. Cache-oblivious algorithms and data structures. In *Lecture Notes from the EEF Summer School on Massive Data Sets*. BRICS, University of Aarhus, Denmark, 1–29.
- Roman Dementiev, Lutz Kettner, and Peter Sanders. 2005. *STXXL: Standard Template Library for XXL Data Sets*. Technical Report. Fakultät für Informatik, Universität Karlsruhe. <http://stxxl.sourceforge.net/>.
- Alon Efrat, Mikko Nikkilä, and Valentin Polishchuk. 2013. Sweeping a terrain by collaborative aerial vehicles. In *Proceedings of the 21st SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, 4–13. DOI: <http://dx.doi.org/10.1145/2525314.2525355>
- Stephan Eidenbenz. 2002. Approximation algorithms for terrain guarding. *Information Processing Letters* 82, 2, 99–105.
- Chaulio R. Ferreira, Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and André M. Pompermayer. 2012. More efficient terrain viewshed computation on massive datasets using external memory. In *Proceedings of the 20th SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'12)*. ACM, New York, NY, 494–497. DOI: <http://dx.doi.org/10.1145/2424321.2424398>
- Peter F. Fisher. 1993. Algorithm and implementation uncertainty in viewshed analysis. *International Journal of Geographical Information Science* 7, 4, 331–347.
- Peter F. Fisher. 1996. Extending the applicability of viewsheds in landscape planning. *Photogrammetric Engineering and Remote Sensing* 62, 11, 1297–1302.
- Jeremy Fishman, Herman Haverkort, and Laura Toma. 2009. Improved visibility computation on massive grid terrains. In *Proceedings of the 17th SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'09)*. ACM, New York, NY, 121–130. DOI: <http://dx.doi.org/10.1145/1653771.1653791>
- W. Randolph Franklin and Clark Ray. 1994. Higher isn't necessarily better: Visibility algorithms and experiments. In *Advances in GIS Research: 6th International Symposium on Spatial Data Handling*, T. C. Waugh and R. G. Healey (Eds.). Taylor & Francis, Edinburgh, Scotland, 751–770.
- W. Randolph Franklin and Christian Vogt. 2006. Tradeoffs when multiple observer siting on large terrain cells. In *Progress in Spatial Data Handling: 12th International Symposium on Spatial Data Handling*, A. Riedl, W. Kainz, and G. Elmes (Eds.). Springer, Vienna, Austria, 845–861. ISBN 978-3-540-35588-5.
- Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 2012. Cache-oblivious algorithms. *ACM Transactions on Algorithms* 8, 1, 4. DOI: <http://dx.doi.org/10.1145/2071379.2071383>
- Michael F. Goodchild and Jay Lee. 1989. Coverage problems and visibility regions on topographic surfaces. *Annals of Operations Research* 18, 1, 175–186.
- Herman Haverkort, Laura Toma, and Bob PoFang Wei. 2013. On IO-efficient viewshed algorithms and their accuracy. In *Proceedings of the 21st SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, 24–33. DOI: <http://dx.doi.org/10.1145/2525314.2525369>
- Herman Haverkort, Laura Toma, and Yi Zhuang. 2009. Computing visibility on terrains in external memory. *Journal of Experimental Algorithmics* 13, 5:1.5–5:1.23. DOI: <http://dx.doi.org/10.1145/1412228.1412233>
- Ferran Hurtado, Maarten Löffler, Inês Matos, Vera Sacrisán, Maria Saumell, Rodrigo I. Silveira, and Frank Staals. 2014. Terrain visibility with multiple viewpoints. *International Journal of Computational Geometry and Applications* 24, 04, 275–306. DOI: <http://dx.doi.org/10.1142/S0218195914600085>
- Branko Kaucic and Borut Zalik. 2002. Comparison of viewshed algorithms on regular spaced points. In *Proceedings of the 18th Spring Conference on Computer Graphics (SCCG'02)*. ACM, New York, NY, 177–183. DOI: <http://dx.doi.org/10.1145/584458.584487>

- Iain R. Lake, Andrew A. Lovett, Ian J. Bateman, and Ian H. Langford. 1998. Modelling environmental influences on property prices in an urban environment. *Computers, Environment and Urban Systems* 22, 2, 121–136.
- Niel Lebeck, Thomas Mølhave, and Pankaj K. Agarwal. 2014. Computing highly occluded paths using a sparse network. In *Proceedings of the 22nd SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'14)*. ACM, New York, NY, 3–12. DOI: <http://dx.doi.org/10.1145/2666310.2666394>
- Jay Lee, Peter K. Snyder, and Peter F. Fisher. 1992. Modeling the effect of data errors on feature extraction from digital elevation models. *Photogrammetric Engineering and Remote Sensing* 58, 10, 1461–1467.
- Jay Lee and Dan Stucky. 1998. On applying viewshed analysis for determining least-cost paths on digital elevation models. *International Journal of Geographical Information Science* 12, 8, 891–905.
- Zhilin Li, Qing Zhu, and Christopher Gold. 2005. *Digital Terrain Modeling—Principles and Methodology*. CRC Press, Boca Raton, FL.
- Salles V. G. Magalhães, Marcus V. A. Andrade, and W. Randolph Franklin. 2011. Multiple observer siting in huge terrains stored in external memory. *International Journal of Computer Information Systems and Industrial Management* 3, 143–149. <http://www.mirlabs.org/ijcisim/volume1.html>.
- Mark A. Maloy and Denls J. Dean. 2001. An accuracy assessment of various GIS-based viewshed delineation techniques. *Photogrammetric Engineering and Remote Sensing* 67, 11, 1293–1298.
- MindSites Group. 2016. USGS SDTS Format Digital Elevation Model Data (DEM). Retrieved May 20, 2016, from <http://data.geocomm.com/dem/>.
- George Nagy. 1994. Terrain visibility. *Computers and Graphics* 18, 6, 763–773.
- National Digital Elevation Program. 2015. Digital Elevation Glossary of Terms. Retrieved May 20, 2016, from <http://www.ndep.gov/glossary.html>.
- Passmark Software. 2013. CPU Benchmarks. Retrieved May 20, 2016, from <http://www.cpubenchmark.net/>.
- Gennady Pekhimenko, Todd C. Mowry, and Onur Mutlu. 2012. Linearly compressed pages: A main memory compression framework with low complexity and low latency. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. ACM, New York, NY, 489–490. DOI: <http://dx.doi.org/10.1145/2370816.2370911>
- Valentin Polishchuk, Esther M. Arkin, Alon Efrat, Christian Knauer, Joseph S. B. Mitchell, Guenter Rote, Lena Schlipf, Topi Talvitie, and Valentin Polishchuk. 2016. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry* 07, 02, 77–100. <http://jocg.org/index.php/jocg/article/view/264>.
- Bernhard Rabus, Michael Eineder, Achim Roth, and Richard Bamler. 2003. Shuttle Radar Topography Mission. Retrieved May 20, 2016, from <http://www2.jpl.nasa.gov/srtm/>.
- Clark K. Ray. 1994. *Representing Visibility for Siting Problems*. Ph.D. Dissertation. Rensselaer Polytechnic Institute, Troy, NY.
- Jarek Rossignac. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics* 5, 1, 47–61.
- Jaqueline A. Silveira, Salles V. G. Magalhães, Marcus V. A. Andrade, and Vinicius S. Conceição. 2013. A library to support the development of applications that process huge matrices in external memory. In *Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS'13)*. 305–310.
- Jared Stookey, Zhongyi Xie, Barbara Cutler, W. Randolph Franklin, Daniel M. Tracy, and Marcus V. A. Andrade. 2008. Parallel ODETLAP for terrain compression and reconstruction. In *Proceedings of the 16th ACM SIGSPATIAL SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'08)*. ACM, New York, NY, 17. DOI: <http://dx.doi.org/10.1145/1463434.1463456>
- Laura Toma, Yi Zhuang, and William Richard. 2010. R.viewshed. Retrieved May 20, 2016, from <https://trac.osgeo.org/grass/browser/grass-addons/raster/r.viewshed?rev=45442>.
- Tom's Guide. 2016. Smartphones Buying Guide. Retrieved May 20, 2016, from <http://www.tomsguide.com/us/smartphone-buying-guide,review-1971.html>.
- Marc Van Kreveld. 1996. Variations on sweep algorithms: Efficient computation of extended viewsheds and class intervals. In *Proceedings of the 7th International Symposium on Spatial Data Handling*. 13–15.

Received February 2014; revised February 2016; accepted March 2016