# Efficient viewshed computation on terrain in external memory

Marcus V. A. Andrade[1,2], Salles V. G. Magalhães[1], Mirella A. Magalhães[1], W. Randolph Franklin[2], and Barbara M. Cutler[3]

[1]DPI, Universidade Federal de Viçosa, Brasil

[2]ECSE Dept., Rensselaer Polytechnic Institute, USA

[3]CS Dept., Rensselaer Polytechnic Institute, USA

**Abstract**

The recent availability of detailed geographic data permits terrain applications to process large areas at high resolution. However the required massive data processing presents significant challenges, demanding algorithms optimized for both data movement and computation. One such application is *viewshed computation*, that is, to determine all the points visible from a given point $p$. In this paper, we present an efficient algorithm to compute viewsheds on terrain stored in external memory. In the usual case where the observer's radius of interest is smaller than the terrain size, the algorithm complexity is $\theta(scan(n^2))$ where $n^2$ is the number of points in an $n \times n$ DEM and $scan(n^2)$ is the minimum number of I/O operations required to read $n^2$ contiguous items from external memory. This is much faster than existing published algorithms.

## 1 Introduction

Terrain modeling is an important application in Geographic Information Science (GIS). One aspect is the computation of all points that can be viewed from a given point (the observer). The region composed of the visible points is called the *viewshed* [15, 19]. This problem has many applications, such as determining the minimum number of cellular phone towers required to cover a region [5, 9, 11], optimizing the number and position of guards to cover a region [14, 20], analyzing influences on property prices in an urban environment [25], and optimizing path planning on a DEM [26].

The recent technological advances in data collection (such as LIDAR and IFSAR) have produced a huge volume of data about the Earth's surface [31]. For example, modeling a $100\,km \times 100\,km$ terrain at $1m$ resolution requires $10^{10}$ points. Since most computers cannot store or process this huge volume of data internally, an external memory algorithm is required. Since the time required to access and transfer data to and from external memory is generally much larger than the internal processing time, the algorithm must try to minimize the external memory I/O [4, 21]. More specifically, such external memory algorithms should be optimized under a computational model whose cost is the number of data transfer operations instead of CPU time. One such model was proposed by Aggarwal and Vitter [1].

In this work, we present an efficient algorithm to compute the viewshed of a point on a terrain stored in external memory. The algorithm is an adaptation of Franklin and

Ray's method [18, 19] to allow efficient manipulation of huge terrains (6GB or more). The large number of disk accesses is optimized using the STXXL library [13]. Our algorithm is more than six times faster and much easier to implement than the excellent algorithm proposed by Haverkort et al. [22].

This paper is organized as follows. Section 2 gives a brief description of viewshed computation and I/O-efficient algorithms for general problems as well as for viewshed computation. Section 3 formally presents the viewshed concepts. Section 4 briefly describes the I/O-efficient computational model. Section 5 describes the algorithm in detail. Section 6 analyzes its complexity. Section 7 gives the tests results, and Section 8 presents the conclusions.

## 2 Related Work

### 2.1 Terrain representation

Terrain is generally represented either by a *Triangulated Irregular Network (TIN)* or a *Raster Digital Elevation Model (DEM)* [16, 27]. A TIN, first implemented in GIS by Franklin [17], is a partition of the surface into planar triangles, i.e., a piecewise linear triangular spline. The elevation of any point $p$ is a bilinear interpolation of the elevations of the three vertices of the triangle whose projection onto the $xy$ plane contains the projection of $p$.

A DEM is simply a matrix of the elevations at regularly spaced positions or posts. The spacing may be either a constant number of meters or a constant angle in latitude and longitude.

Both representations are seen because neither is clearly better than the other [16, 24].

A DEM uses a simpler data structure, is easier to analyze and has higher accuracy at high resolution, but requires more memory space than a TIN. In this paper, we use the, simpler, DEM data structure.

## 2.2 Terrain visibility

Terrain visibility has been widely studied. Stewart [30] shows how the viewshed can be efficiently computed for each point of a DEM. His interest is positioning radio transmission towers. Kreveld [32] proposes a sweep-line approach to compute the viewshed in $\theta(n \log n)$ time[1] on a $\sqrt{n} \times \sqrt{n}$ grid. In [18, 19], Franklin and Ray describe experimental studies for fast implementation of visibility computation and present several programs that explore trade-offs between speed and accuracy. Young-Hoon, Rana and Wise in [33] analyze two strategies to use the viewshed for optimization problems. Ben-Moshe et al. [6, 7, 8] have worked on visibility for terrain simplification and for facility positioning. For surveys on visibility algorithms, see [15, 28].

## 2.3 External memory processing

Aggarwal and Vitter [1] discuss some problems, and propose a computational model to evaluate algorithm complexity based on the number of input/output (I/O) operations. Goodrich et al. [21] present some variants for the sweep plane paradigm considering external processing, while Arge et al. [4] describe a solution for externally processing line segments in GIS. This technique was also used to solve hydrology problems, such as computing water flow and watershed [3] on huge terrain.

[1] $\theta(f(n))$ grows proportionally to $f(n)$ as $n \to \infty$. Formally, $g(n) = \theta(f(n)) \Rightarrow \exists n_0 > 0, c_1 > c_2 > 0$ such that $n > n_0 \Rightarrow c_1 f(n) > g(n) > c_2 f(n)$. Hein [23, page 334].

Recently, Haverkort et al. [22] adapted van Kreveld's method to compute viewsheds on terrain stored in external memory; its I/O complexity is $\theta(sort(n))$, where $n$ is the number of points on the terrain and $\theta(sort(n))$ is the minimum number of I/O operations required to sort $n$ contiguous items stored in external memory.

The EMVS (*external memory viewshed*) algorithm, which we present here, also has a worst-case I/O complexity of $\theta(sort(n))$, but its execution time is lower because of a more efficient strategy - it uses very simple data structures and is based on sorting process. Also, it is much easier to implement than the Haverkorth et al. method.

## 3 The Viewshed Problem

Visibility problems can be classified into two major categories: visibility queries and the visibility structure computation. Visibility queries consist of checking whether a given point $\mathcal{P}$, the *target*, is visible from another point $\mathcal{O}$, the *observer* or *source*. Both $\mathcal{O}$ and $\mathcal{P}$ are usually slightly above the terrain. For example, consider a radio tower communicating with a cellphone user. $\mathcal{P}$ is visible from $\mathcal{O}$ iff a straight line, *the line of sight*, from $\mathcal{O}$ to $\mathcal{P}$ is always strictly above the terrain. See Figure 1.

The visibility structure computation consists of determining some features such as the horizon and viewshed. Formally, the viewshed of a point $p$ on a terrain $\mathcal{T}$ can be defined as:

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

When computing the viewshed, it is common to consider only points within a given

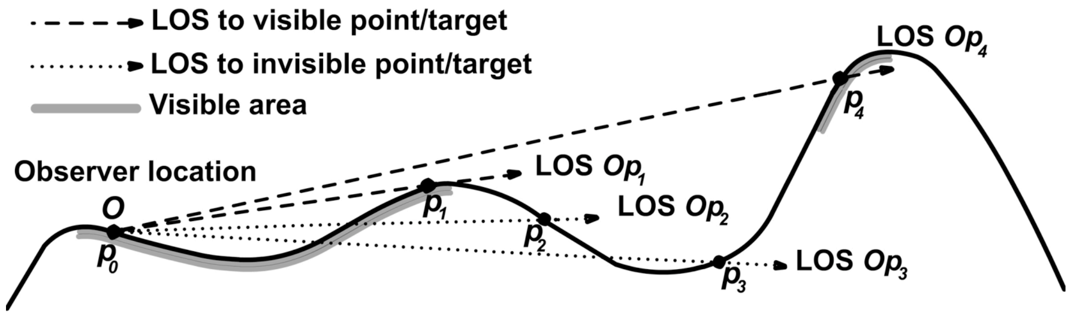

Figure 1: Point Visibility: $p_1$ and $p_4$ are visible from $p_0$; $p_2$ and $p_3$ are not visible from $p_0$.

distance $r$, the *radius of interest*, of $p$. In this case,

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

We will generally assume $r$ and simplify the notation of $viewshed(p,r)$ to $viewshed(p)$. Since we are working with raster DEMs, we represent a viewshed by a square $2r \times 2r$ matrix of bits.

## 4 I/O efficient Algorithms

As mentioned before, for large datasets, I/O is the bottleneck. However, many GIS algorithms are designed to optimize internal processing, and so do not scale up. A more appropriate computational model is needed. One common model, proposed by Aggarwal and Vitter [1], defines an I/O operation as the transfer of one disk block of size $B$ between external and internal memory. The measure of performance is 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 stored in external memory. Those are

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

where $M$ is the internal memory size. Because $B \gg 1$, usually $scan(N) < sort(N) \ll N$, and it is important to organize the data in external memory to decrease the number of I/O operations.

## 5 External Memory Viewshed Computation (EMVS)

Our algorithm, External Memory Viewshed (EMVS), is based on the method proposed by Franklin and Ray [19] that computes the viewshed of a point on a terrain represented as an internal memory matrix. That is summarized below.

### 5.1 Franklin and Ray's Method

Given a terrain $\mathcal{T}$ represented by an $n \times n$ elevation matrix $\mathcal{M}$, a point $p$ on $\mathcal{T}$, a radius of interest $r$, and a height $h$ above the local terrain for the observer and target, this algorithm computes the viewshed of $p$ within a distance $r$ of $p$, as follows:

1. Let $p$'s coordinates be $(x_p, y_p, z_p)$. Then the observer $\mathcal{O}$ will be at $(x_p, y_p, z_p + h)$.

2. Imagine a square in the plane $z = 0$ of side $2r \times 2r$ centered on $(x_p, y_p, 0)$.

3. Iterate through the cells $c$ of the square's perimeter. Each $c$ has coordinates $(x_c, y_c, 0)$, where the corresponding point on the terrain is $(x_c, y_c, z_c)$.

   (a) For each $c$, run a straight line in $\mathcal{M}$ from $(x_p, y_p, 0)$ to $(x_c, y_c, 0)$.

   (b) Find the points on that line, perhaps using Bresenham's algorithm. In order from $p$ to $c$, let them be $q_1 = p, q_2, \cdots q_{k-1}, q_k = c$. A potential target $\mathcal{D}_i$ at $q_i$ will have coordinates $(x_i, y_i, z_i + h)$.

   (c) Let $m_i$ be the slope of the line from $\mathcal{O}$ to $\mathcal{D}_i$, that is,

$$m_i = \frac{z_k - z_i + p}{\sqrt{(x_i - x_p)^2 + (y_i - y_p)^2}}.$$

   (d) Let $\mu$ be the greatest slope seen so far along this line. Initialize $\mu = -\infty$.

   (e) Iterate along the line from $p$ to $c$.

      i. For each point $q_i$, compute $m_i$.

      ii. If $m_i < \mu$, then mark $q_i$ as hidden from $\mathcal{O}$, that is, as not in the viewshed (which is simply a $2r \times 2r$ bitmap).

      iii. Otherwise, mark $q_i$ as being in the viewshed, and update $\mu = m_i$.

The total execution time is linear in $N$, the number of points in the terrain. In contrast, earlier algorithms that ran a separate line of sight to each potential target had an execution time of $\theta(N^{3/2})$.

This algorithm can be used to compute the viewshed on terrain in external memory. However, since the cells are accessed in a sequence defined by the radial sweep, that would require random access to the file, and the execution time would be unacceptably long. This random access order can be avoided using the adaptation described below.

## 5.2 The EMVS algorithm

The basic idea is to generate a list with the terrain points sorted by the processing order, that is, the points will appear in the list in the sequence given by the radial sweep and by the processing order along each line of sight. Thus, during the viewshed computation, the algorithm follows the list, avoiding accessing the file randomly.

This list stored in external memory is managed by STXXL (*the Standard Template Library for Extra Large Data Sets*) [13], which implements containers and algorithms to process huge volumes of data. This library allows an efficient manipulation of external data and, as stated by the authors, "it can save more than half the number of I/Os performed by many applications".

Specifically, the algorithm creates a list $L$ of pairs $(c, i)$ where $c$ is a terrain position and $i$ is an index indicating the order in which $c$ should be processed.

To compute the cell indices, the lines of sight (originating at the observer $p$) are numbered in counterclockwise order starting at the positive $x$-axis, which is number 0 — see Figure 2. Thus, the cells are numbered increasingly along each line of sight; when a line of sight ends, the enumeration proceeds from the observer (again numbered) following the next line of sight. Of course, a same cell (point) can receive multiple indices since it can be intercepted by many lines of sight, especially if it is close to the observer. This means that a same point can appear in multiple pairs in the list $L$, but each pair will have a different index. Also, if the observer is near to the terrain border, that is, if the distance between the observer and the terrain border is smaller than the radius of interest $r$, some cells in a line of sight can be outside the terrain. In this case, those cells still will be numbered but they will be ignored (i.e. they will not be inserted in the list $L$). This is done to simplify the indices computation by avoiding many additional conditional tests.

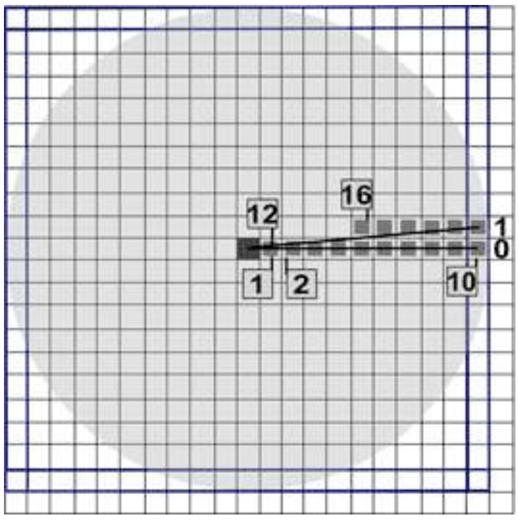

Figure 2: Line of sight enumeration

Even when computing the cell indices as described above, the file still would be randomly accessed, as in the original algorithm. So, to build the list $L$, the algorithm reads the terrain cells sequentially from the external file and for each cell $c$, it determines (the number of) all lines of sight that intercept that cell.

Since the cells have a finite size (they are not points), we can determine the cells intercepted by a line of sight using a process similar to line rasterization [10]. That is, let $s$ be the side of each (square) cell and suppose the cell is referenced by its center. Also, let $a$ be a line of sight whose slope is $\alpha \ : \ 0 < \alpha \leq 45°$ [2]. So, given a cell $c = (c_x, c_y)$, see Figure 3, the line of sight $a$ "intersects" the cell $c$ if and only if the intersection point between $a$ and the vertical line $c_x$ is a point in the segment $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$; more precisely, given $(q_x, q_y) = a \cap c_x$, $a$ intersects $c$ if and only if $c_y - 0.5s \leq q_y < c_y + 0.5s$. In this case, a line of sight as the dashed line in the Figure 3 is assumed to intersect the cell above $c$.

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

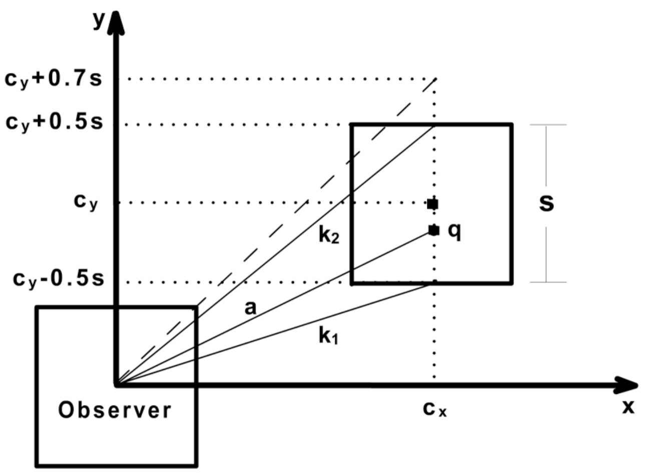

Figure 3: Lines of sight intersecting a cell

Then, all lines of sight intersecting the cell $c$ are those between the two lines connecting the observer to the points $(c_x, c_y - 0.5s)$ and $(c_x, c_y + 0.5s)$ — Figure 3. Let $k_1$ and $k_2$ be the numbers of these two lines respectively. Considering the line of sight definition (a segment connecting the observer and the center of cells on the square border) and the line enumeration, we have that the number of the lines intersecting the cell $c$ can be determined by the cells in the border whose center are between $k_1$ and $k_2$ - see Figure 4. For example, in this Figure, cell $c$ is intersected by the lines 3 and 4.

Now, given a cell $c$, let $\kappa$ be the number of a line of sight intercepting $c$. Then, the index $i$ of the cell $c$ associated with $\kappa$ is given by the formula $i = \kappa * n + d$, where $n$ is the number of cells in each ray (this number is constant for all rays) and $d$ is the (horizontal or vertical) distance between points $c$ and $p$ — see Figure 5. Note that the distance $d$ is defined as the maximum between the number of rows and columns from $p$ to $c$.

Next, the list $L$ is sorted by the elements' index, and then the cells are processed in the sequence given by the sorted list. When a cell $c$ is processed, all the "previous" cells

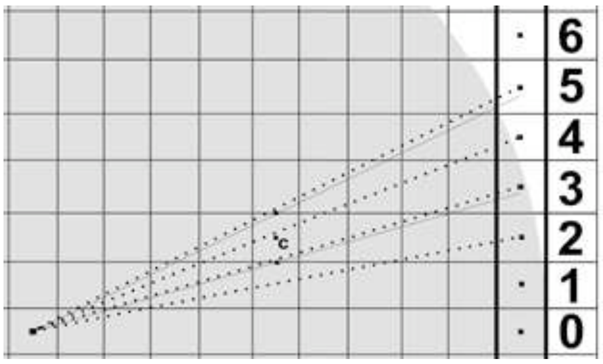

Figure 4: Number of the lines of sight intersecting a cell.

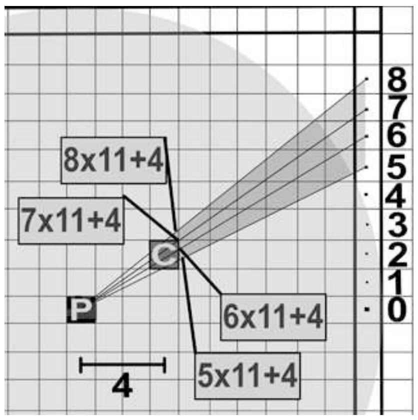

Figure 5: The index cell computation.

that could block the visibility of $c$ have already been processed. So, the visibility of $c$ can be computed, as described before, just by checking the height of the cells along the line of sight. When a cell located on the square border is processed, it means that the line of sight processing has finished. The next cell in the list will be the observer's cell, indicating that the processing of a new line of sight will start.

To improve the algorithm's efficiency, another list $L'$ (also stored externally and managed by *STXXL*) is used to keep only the visible cells. When the algorithm determines that a cell $c$ is visible, this cell is inserted in $L'$. The size of $L'$ is much smaller than

$L$ since $L'$ does not keep the indices, and usually many points are not visible. To avoid random access, before storing the viewshed, the list $L'$ is sorted lexicographically by $x$ and $y$. Then the viewshed is stored in an external file where a visible position (a point in $L'$) is indicated by 1 and not visible point by 0.

Additionally, a piece of the terrain matrix is stored in the internal memory. The idea is to store the cells around the observer since these cells are processed more times than the farther ones. All cells inside a square centered at the observer position are stored internally, and are not inserted in the lists $L$ and $L'$. When a cell needs to be processed, the algorithm checks if it is in the internal memory. If so, the cell is processed normally; otherwise, it is read from list $L$.

## 6 Algorithm complexity

Let $T$ be some terrain represented by an $n \times n$ elevation matrix. Let $p$ be the observer's position, and $r$ be the radius of interest. As described in Section 5.2, the algorithm considers the cells that are inside the $2r \times 2r$ square centered at $p$. Assuming that each cell's side is $s$, there are, at most[3], $8r/s$ cells on the square's perimeter (each square side has $2r/s$ cells). Let $K = r/s$. Thus, the algorithm shoots $8K$ lines of sight and since each line of sight has $K$ cells, the list $L$ has, in the worst case, $\theta(K^2)$ elements.

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

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

cell's visibility. Thus, the total number of I/O operations is:

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

Since, in general, $r \ll n$, then $K < n$ which implies that the number of I/O operations is $\theta(\frac{n^2}{B}) = \theta(scan(n^2))$. In the rare worst case when $r$ is big enough to cover almost the whole terrain, the number of I/O operations is

$$\theta\left(\frac{K^2}{B}\log_{\left(\frac{M}{B}\right)}\left(\frac{K^2}{B}\right)\right) = \theta(sort(K^2))$$

.

The algorithm also uses an additional external list $L'$ to keep the visible cells and this list needs to be sorted. Since the list size is much smaller than the size of $L$, the number of I/O operations executed in this step does not change the algorithm complexity.

## 7 Experimental results

EMVS was implemented in C++, in *g++ 4.1.1* under Mandriva Linux, on a 2.8 GHZ Pentium PC with 1 GB of RAM and an 80 GB 7200 RPM serial ATA hard drive. To better evaluate the I/O operation cost, we considered two configurations. The first used the whole 1 GB of RAM and allowed our program to use 800 MB for data. The second used only 256 MB and allowed 200 MB for data. Although 256 MB main memory is quite small for modern PCs, we used it for two reasons: to illustrate the behaviour and trends of the algorithm as the difference between dataset and memory size increases, and to give an idea of the algorithm performance on portable devices with big hard drives but little

RAM.

The tests used the data sets (from NASA STRM home page [29]) of the regions 1, 2 and 3 shown in Figure 6 sampled with a resolution of 1 arc of second (approximately 30m). From this terrain, we selected pieces of different sizes and each piece was obtained by defining the observer position and the radius of interest[4]. Since these datasets contain a very small percentage (less than 1.5%) of no-data (i.e., points for which the elevation is unknown or invalid), our results are not influenced by no-data points.

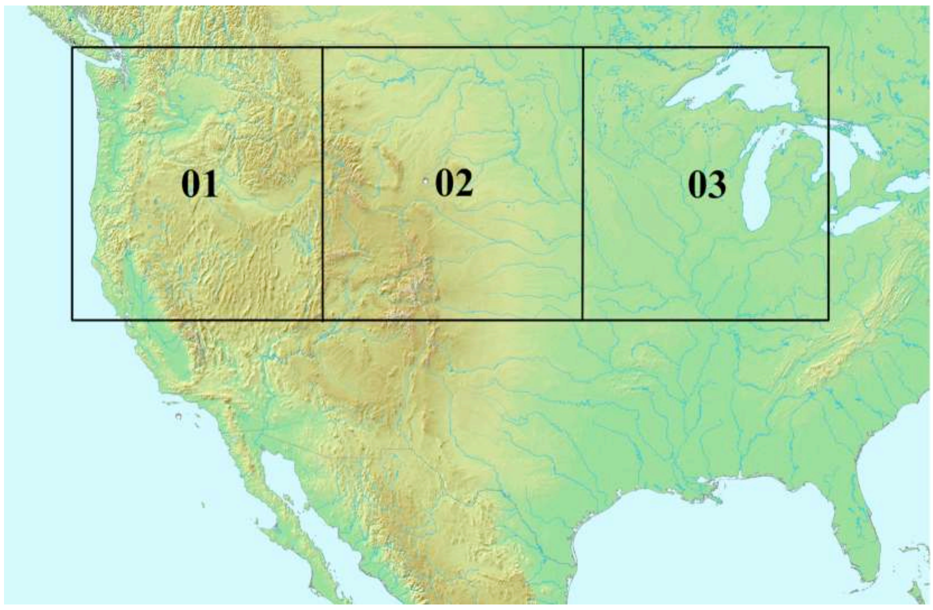

Figure 6: Regions 1, 2 and 3 of the USA used in the tests.

Tables 1 and 2 show the EMVS execution time with either 1G or 256 MB of RAM. We always considered the worst case radius of interest, i.e., big enough to cover the whole terrain. External processing time (I/O, external sorting, file access, etc) is shown separately (column *Ext.*) from the total running time (column *Tot*). To evaluate the

[4]To compare the efficiency of our algorithm and the Haverkort et al. algorithm, we used terrain of size similar to those used by them.

influence of the number of visible points (the column *# Vis. Pts.*) on the time, the observer was positioned at different heights (1, 50, 100, 1000 and 10000 meters) above the terrain. 1000 and 10000 meters are presented to demonstrate the algorithm's scalability.

**Running time with 1GB RAM**

| | | Region 1 | | | Region 2 | | | Region 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Terrain Size | Obs. Hgt | # Vis. Pts | Time (sec.) Ext. | Tot. | # Vis. Pts | Time (sec.) Ext. | Tot. | # Vis. Pts | Time (sec.) Ext. | Tot. |
| | 1 | $1.2 \times 10^6$ | 17 | 43 | $4.5 \times 10^4$ | 19 | 48 | $2.1 \times 10^5$ | 20 | 47 |
| $9029^2$ | 50 | $6.7 \times 10^6$ | 17 | 45 | $3.6 \times 10^7$ | 17 | 50 | $6.3 \times 10^7$ | 18 | 55 |
| 311 MB | $10^2$ | $1.1 \times 10^7$ | 17 | 46 | $6.3 \times 10^7$ | 18 | 56 | $6.4 \times 10^7$ | 18 | 57 |
| | $10^3$ | $5.6 \times 10^7$ | 18 | 54 | $6.3 \times 10^7$ | 19 | 59 | $6.4 \times 10^7$ | 18 | 57 |
| | $10^4$ | $6.3 \times 10^7$ | 19 | 56 | $6.3 \times 10^7$ | 19 | 59 | $6.4 \times 10^7$ | 20 | 59 |
| | 1 | $4.2 \times 10^5$ | 67 | 209 | $1.3 \times 10^5$ | 70 | 208 | $1.1 \times 10^5$ | 69 | 209 |
| $17150^2$ | 50 | $2.6 \times 10^6$ | 67 | 208 | $1.5 \times 10^8$ | 66 | 219 | $1.6 \times 10^8$ | 69 | 241 |
| 1122 MB | $10^2$ | $9.5 \times 10^6$ | 68 | 210 | $2.1 \times 10^8$ | 66 | 243 | $2.1 \times 10^8$ | 67 | 247 |
| | $10^3$ | $1.5 \times 10^8$ | 67 | 237 | $2.2 \times 10^8$ | 66 | 246 | $2.2 \times 10^8$ | 67 | 238 |
| | $10^4$ | $2.2 \times 10^8$ | 69 | 249 | $2.2 \times 10^8$ | 67 | 244 | $2.2 \times 10^8$ | 67 | 248 |
| | 1 | $1.2 \times 10^6$ | 1716 | 2598 | $4.3 \times 10^3$ | 1766 | 2652 | $1.1 \times 10^3$ | 1756 | 2630 |
| $33433^2$ | 50 | $7.3 \times 10^6$ | 1759 | 2624 | $1.1 \times 10^7$ | 1756 | 2654 | $2.2 \times 10^8$ | 1763 | 2709 |
| 4264 MB | $10^2$ | $1.5 \times 10^7$ | 1769 | 2648 | $1.9 \times 10^8$ | 1776 | 2728 | $6.6 \times 10^8$ | 1777 | 2799 |
| | $10^3$ | $6.4 \times 10^8$ | 1716 | 2716 | $8.5 \times 10^8$ | 1764 | 2821 | $8.6 \times 10^8$ | 1776 | 2851 |
| | $10^4$ | $8.7 \times 10^8$ | 1750 | 2797 | $8.6 \times 10^8$ | 1766 | 2840 | $8.6 \times 10^8$ | 1758 | 2811 |
| | 1 | $3.5 \times 10^5$ | 3117 | 4546 | $3.9 \times 10^6$ | 3981 | 5434 | $2.1 \times 10^6$ | 3109 | 4599 |
| $40000^2$ | 50 | $2.9 \times 10^6$ | 3108 | 4555 | $7.6 \times 10^7$ | 3960 | 5530 | $1.7 \times 10^8$ | 4290 | 5906 |
| 6103 MB | $10^2$ | $1.2 \times 10^7$ | 4241 | 5697 | $1.9 \times 10^8$ | 4401 | 6046 | $6.4 \times 10^8$ | 4603 | 6739 |
| | $10^3$ | $7.7 \times 10^8$ | 4410 | 6244 | $9.7 \times 10^8$ | 4492 | 6322 | $1.0 \times 10^9$ | 4484 | 6370 |
| | $10^4$ | $1.2 \times 10^9$ | 5064 | 6962 | $1.2 \times 10^9$ | 4852 | 6808 | $1.0 \times 10^9$ | 4544 | 6431 |

Table 1: EMVS running time (in seconds) at 1 GB RAM on pieces of terrain with different sizes from Regions 1, 2 and 3 and varying the observer height above the terrain (generating viewshed with different number of visible points - shown in the column # Vis. Pts). In all cases, the radius of interest cover the whole terrain.

Figure 7 summarizes the internal and external processing time on Region 1 terrain,

**Running time with 256MB RAM**

| | | Region 1 | | | Region 2 | | | Region 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Terrain Size | Obs. Hgt | # Vis. Pts | Time (sec.) Ext. | Tot. | # Vis. Pts | Time (sec.) Ext. | Tot. | # Vis. Pts | Time (sec.) Ext. | Tot. |
| | 1 | $1.2 \times 10^6$ | 21 | 48 | $4.5 \times 10^4$ | 21 | 49 | $2.1 \times 10^5$ | 22 | 51 |
| $9029^2$ | 50 | $6.7 \times 10^6$ | 21 | 51 | $3.6 \times 10^7$ | 23 | 58 | $6.3 \times 10^7$ | 21 | 59 |
| 311 MB | $10^2$ | $1.1 \times 10^7$ | 21 | 50 | $6.3 \times 10^7$ | 21 | 60 | $6.4 \times 10^7$ | 22 | 63 |
| | $10^3$ | $5.6 \times 10^7$ | 21 | 59 | $6.3 \times 10^7$ | 22 | 61 | $6.4 \times 10^7$ | 22 | 61 |
| | $10^4$ | $6.3 \times 10^7$ | 21 | 59 | $6.3 \times 10^7$ | 22 | 62 | $6.4 \times 10^7$ | 21 | 61 |
| | 1 | $4.2 \times 10^5$ | 524 | 712 | $1.3 \times 10^5$ | 521 | 705 | $1.1 \times 10^5$ | 525 | 709 |
| $17150^2$ | 50 | $2.6 \times 10^6$ | 524 | 714 | $1.5 \times 10^8$ | 529 | 747 | $1.6 \times 10^8$ | 528 | 750 |
| 1122 MB | $10^2$ | $9.5 \times 10^6$ | 527 | 718 | $2.1 \times 10^8$ | 523 | 750 | $2.1 \times 10^8$ | 533 | 763 |
| | $10^3$ | $1.5 \times 10^8$ | 526 | 745 | $2.2 \times 10^8$ | 523 | 753 | $2.2 \times 10^8$ | 527 | 759 |
| | $10^4$ | $2.2 \times 10^8$ | 530 | 750 | $2.2 \times 10^8$ | 510 | 737 | $2.2 \times 10^8$ | 521 | 751 |
| | 1 | $1.2 \times 10^6$ | 4666 | 5621 | $4.3 \times 10^3$ | 3876 | 4834 | $1.1 \times 10^3$ | 3886 | 4842 |
| $33433^2$ | 50 | $7.3 \times 10^6$ | 4842 | 5798 | $1.1 \times 10^7$ | 4281 | 5247 | $2.2 \times 10^8$ | 4817 | 5863 |
| 4264 MB | $10^2$ | $1.5 \times 10^7$ | 4957 | 5930 | $1.9 \times 10^8$ | 4945 | 5979 | $6.6 \times 10^8$ | 5301 | 6462 |
| | $10^3$ | $6.4 \times 10^8$ | 5361 | 6514 | $8.5 \times 10^8$ | 5275 | 6498 | $8.6 \times 10^8$ | 5309 | 6528 |
| | $10^4$ | $8.7 \times 10^8$ | 5570 | 6789 | $8.6 \times 10^8$ | 5326 | 6554 | $8.6 \times 10^8$ | 5308 | 6535 |
| | 1 | $3.5 \times 10^5$ | 6861 | 8244 | $3.9 \times 10^6$ | 6707 | 8102 | $2.1 \times 10^6$ | 6689 | 8084 |
| $40000^2$ | 50 | $2.9 \times 10^6$ | 7020 | 8408 | $7.6 \times 10^7$ | 6838 | 8281 | $1.7 \times 10^8$ | 7077 | 8565 |
| 6103 MB | $10^2$ | $1.2 \times 10^7$ | 7030 | 8421 | $1.9 \times 10^8$ | 7712 | 9199 | $6.4 \times 10^8$ | 7726 | 9369 |
| | $10^3$ | $7.7 \times 10^8$ | 7759 | 9431 | $9.7 \times 10^8$ | 7958 | 9714 | $1.0 \times 10^9$ | 8095 | 9870 |
| | $10^4$ | $1.2 \times 10^9$ | 8343 | 10185 | $1.2 \times 10^9$ | 8427 | 10284 | $1.0 \times 10^9$ | 7956 | 9728 |

Table 2: EMVS running time (in seconds) at 256 MB RAM on pieces of terrain with different sizes from Regions 1, 2 and 3 and varying the observer height above the terrain (generating viewshed with different number of visible points - shown in the column # Vis. Pts). In all cases, the radius of interest cover the whole terrain.

using 256 MB and 1 GB of RAM (the results for Regions 2 and 3 are quite similar). As expected, the external processing time is much larger than the internal processing time on terrain that is much bigger than the internal memory size. See charts (c), (d) and mainly (b) where the external processing time is longer (resp. shorter) than the internal processing time when using 256 MB (resp. 1 GB). The difference in (b) occurs because the 1122 MB terrain can be processed almost completely in 1 GB internal memory, requiring few I/O operations. However, with only 256 MB, many I/O operations are necessary.

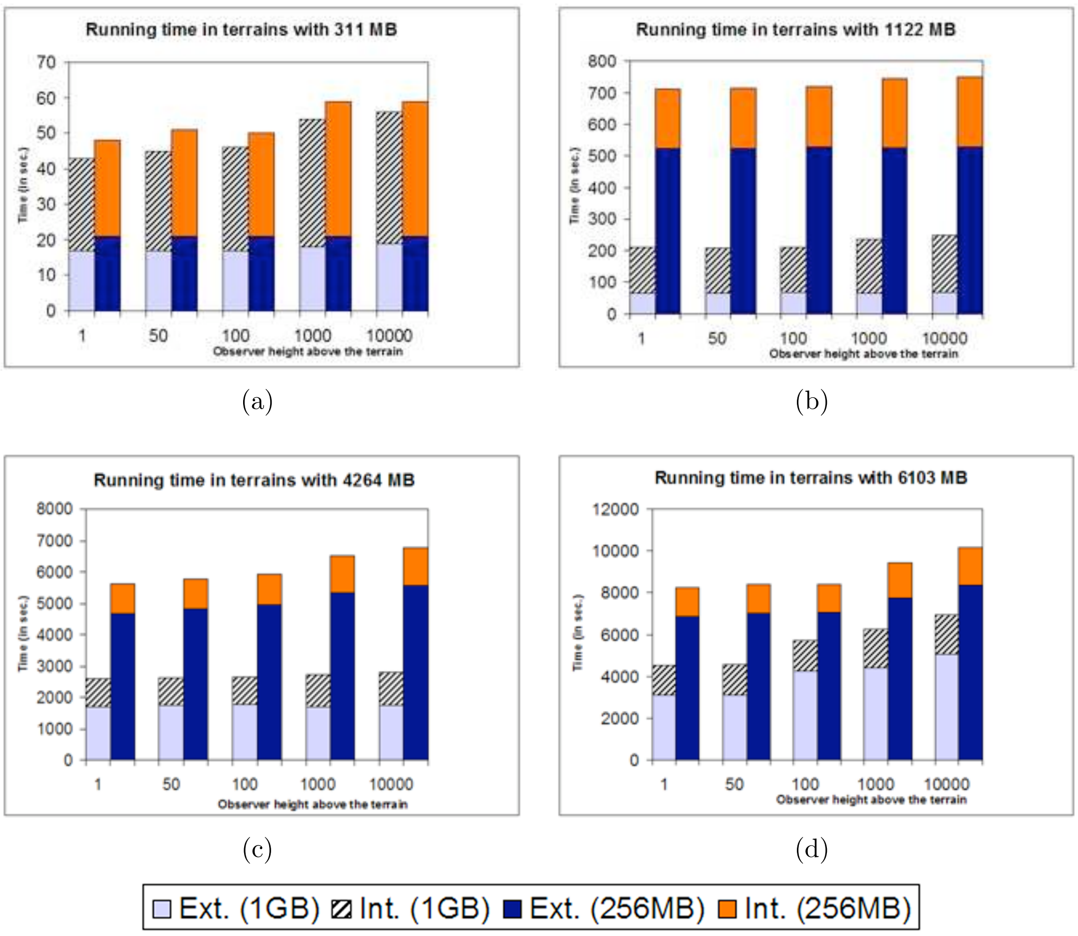

Figure 7: The internal and external processing time using 256MB and 1GB of RAM on pieces of terrain from Region 1 with different sizes.

As the terrain size increases, the total processing time is essentially determined by the external processing time. That seems to converge to about 80% and 70% of the total time when using 256 MB and 1 GB of RAM respectively.

We also compared EMVS to the IO_VS algorithm of Haverkort et al. That is an adaptation for external processing of a method proposed by Van Kreveld [32] to compute the viewshed using a radial sweep of a terrain. Basically, Van Kreveld's algorithm uses a plane sweep technique. Starting with a grid and a viewpoint $p$, it rotates a sweep line around $p$, computing the visibility of each cell in the terrain when the sweep-line passes over its center. It implements this with an active-structure data structure to contain the cells currently intersected by the sweep-line (the active cells). When a cell is intersected by the sweep-line, it is inserted in the active structure; when a cell stops being intersected by the sweep-line, it is deleted from the active structure. When the center of a cell is intersected by the sweep line, the active structure is queried to find out if that cell is visible. Thus, each cell in the grid has three associated events: when it is first intersected by the sweep-line and entered in the data structure, when the sweep-line passes over its center, and when it is last intersected by the sweep-line and removed from the data structure.

Haverkort et al. extended that into an algorithm to compute the viewshed on terrain stored in external memory, where the cells are sorted based on when they will be processed by the radial sweep.

Table 3 compares EMVS to the results reported for IO_VS in [22], with the observer 1 meter above the terrain. The EMVS values were averaged from the three corresponding values for Regions 1, 2 and 3 listed in Tables 1 and 2. From Table 3, we see that EMVS is more than 6 times faster than IO_VS. Further, IO_VS was tested by its author on a Power Macintosh G5 dual 2.5 GHz, 1GB RAM and 80 GB 7200 RPM, which is significantly faster than the machine we used. Therefore, EMVS's relative advantage is probably even greater. Finally, EMVS is much simpler to implement.

Why? EMVS uses a simple data structure — a sorted list. After the external sort, no list updates (insert or delete) are required. On the other hand, IO_VS uses more complex data structures, manipulated using recursive searching and updating.

**1GB**

| Terrain Size | EMVS | IO_VS |
|---|---|---|
| 119 MB | 18 | 353 |
| 311 MB | 46 | 865 |
| 1122 MB | 209 | 3546 |
| 4264 MB | 2627 | 16895 |

**256MB**

| Terrain Size | EMVS | IO_VS |
|---|---|---|
| 119 MB | 22 | 364 |
| 311 MB | 49 | 916 |
| 1122 MB | 709 | 4831 |
| 4264 MB | 5099 | 40734 |

Table 3: Execution time (seconds) comparison between EMVS and IO_VS for 1GB and 256 MB of RAM

## 8 Conclusions

We have presented EMVS, a very efficient algorithm to compute the viewshed of a point in a huge raster DEM terrain stored in external memory. EMVS is more than 6 times faster than the algorithm of Haverkort et al [22] and also, it can process very large datasets; we used it on a 6.1GB terrain. Finally, EMVS algorithm is quite simple to understand and to implement. It is available at Andrade [2] as an open source code distributed under Creative Common GNU GPL license [12].

## Acknowledgment

This work was partially supported by CNPq - the Brazilian Council of Technological and Scientific Development, FAPEMIG - the Research Support Foundation of the State of Minas Gerais (Brazil) and by NSF grants CCR-0306502 and DMS-0327634 and by DARPA/DSO/GeoStar.

## References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] M. V. A. Andrade. EMViewshed Project. `http://www.dpi.ufv.br/~marcus/projects/EMViewshed.htm`, 2007.

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

[4] L. Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. *Algorithmica*, 47(1):1–25, 2007.

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

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

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

[8] B. Ben-Moshe, M.J. Katz, J.S.B. Mitchell, and Y. Nir. Visibility preserving terrain simplification - an experimental study. *Comp. Geom., Theory and Applications*, 28(2-3):175–190, 2004.

[9] S. Bespamyatnikh, Z. Chen, K. Wang, and B. Zhu. On the planar two-watchtower problem. In *In 7th International Computing and Combinatorics Conference*, pages 121–130, London, UK, 2001. Springer-Verlag.

[10] J. Bresenham. An incremental algorithm for digital plotting. *IBM Systems Journal*, 1965.

[11] R. J. Camp, D. T. Sinton, and R. L. Knight. Viewsheds: A complementary management approach to buffer zones. *Wildlife Society Bulletin*, 25:612–615, 1997.

[12] Creative Commons. `http://creativecommons.org/license/cc-gpl` (accessed February 2008), 2007.

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

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

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

[16] L. De Floriani, E. Puppo, and P. Magillo. Applications of computational geometry to geographic information systems. In J. Urrutia J. R. Sack, editor, *Handbook of Computational Geometry*, pages 333–388. Elsevier Science, 1999.

[17] W. R. Franklin. Triangulated irregular network program. `http://www.ecse.rpi.edu/~wrf/wiki/Research/tin73.tgz` [accessed 30 Oct 2008], 1973.

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

[19] W. R. Franklin and C. Ray. Higher isn't necessarily better - visibility algorithms and experiments. In *6th Symposium on Spatial Data Handling*, pages 751–770, Edinburgh, Scotland, 1994. Taylor & Francis.

[20] W. R. Franklin and C. Vogt. Tradeoffs when multiple observer siting on large terrain cells. In Springer-Verlag, editor, *12th International Symposium on Spatial Data Handling*, pages 845–861, 2006.

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

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

[23] J. L. Hein. *Discrete Mathematics.* Jones & Bartlett Publishers, 2002. ISBN 0763722103, 9780763722104.

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

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

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

[27] Z. Li, Q. Zhu, and C. Gold. *Digital Terrain Modeling - principles and methodology.* CRC Press, 2005.

[28] Line-of-Sight Technical Working Group (LOS TWG). Line-of-sight (LOS) compendium. Technical report, U.S. Army Corps of Engineers, Engineer Research and Development Center/ Topographic Engineering Center (ERDC/TEC), 2004. `http://www.tec.army.mil/operations/programs/LOS/LOS%20Compendium.doc`, [accessed 30 Oct 2008].

[29] The Shuttle Radar Topography Mission (SRTM). `http://www2.jpl.nasa.gov/srtm/` (accessed February 2008), 2007.

[30] A. J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. In *IEEE Trans. Visualization Computer Graphics*, volume 4, pages 82 – 93, Piscataway, NJ, USA, 1998. IEEE Educational Activities Department.

[31] US Geological Survey. The USGS Center for LIDAR Information Coordination and Knowledge. `http://lidar.cr.usgs.gov/` (accessed February 2008), 2007.

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

[33] K. Young-Hoon, S. Rana, and S. Wise. Exploring multiple viewshed analysis using terrain features and optimization techniques. *Computers and Geosciences*, 30:1019–10323, 2004.