An Improved Parallel Algorithm using GPU for Siting Observers on Terrain

Guilherme C. Pena[1], Marcus V. A. Andrade[1], Salles V. G. Magalhães[1], W. R. Franklin[2] and Chaulio R. Ferreira[1]

[1]*Departamento de Informática, Universidade Federal de Viçosa (UFV), Viçosa, MG, Brazil*
[2]*Rensselaer Polytechnic Institute, Troy, NY, USA*
{*guilherme.pena, marcus, salles, chaulio.ferreira*}*@ufv.br, wrf@ecse.rpi.edu*

Keywords: Siting, Terrain Visibility, Viewshed, GPU parallel algorithm

Abstract: This paper presents an efficient method to determine a set of observers (that is, where to site them) such that a given percentage of a terrain is visually covered. Our method extends the method proposed in (Franklin, 2002) including a local search heuristic efficiently implemented using dynamic programming and GPU parallel programming. This local search strategy allows to achieve a higher coverage using the same number of observers as the original method and thus it is possible to obtain a given coverage using a smaller number of observers. It can be an important improvement since an observer can represent an expensive facility such as a telecommunication tower. The proposed method performance was compared with that of other methods and the tests showed that it can be more than 1200 times faster than the sequential implementation (with no use of dynamic programming and no GPU parallel programmming) and, also, more than 20 times faster than a previous parallel method presented in (Magalhães et al., 2011).

1 INTRODUCTION

A large amount of high-resolution geographic data has become available because of the recent advances in remote sensing. Thus, the development of advanced techniques to process these data has been required by Geographic Information Science (GIS) (Laurini and Thompson, 1992). The Earth surface elevation (terrain) data are usually represented approximately by a digital elevation matrix (DEM) that stores the elevations of regularly sampled terrain points. Elevations of intermediate points are usually approximated using some interpolation process (Li et al., 2005).

An important group of GIS applications concerns visibility, i.e., determining the set of points that are visible from some particular point, called observer. The observer can be located at some height above the terrain. These applications include telecommunications, environmental planning, autonomous vehicle navigation and military monitoring (Franklin and Ray, 1994; Li et al., 2005; Nagy, 1994; Andrade et al., 2011). One important related problem is the siting of a given number of observers in order to optimally "cover the terrain". These observers may represent radio, TV, Internet or mobile phone towers, or monitoring cameras or towers (Ben-Moshe, 2005; Ben-Shimol et al., 2007). As described in (Nagy, 1994), this is an NP-Hard problem and, therefore, there is no known efficient algorithm to find its optimal solution.

However, even obtaining approximate solutions for this optimization problem demands a long processing time, particularly when processing large quantities of data. One way to reduce this processing time is to design parallel algorithms based on general purpose graphics processing units (GPGPUs), which are present in most current graphics cards.

This paper deals with an instance of the multiple observers siting problem where the goal is to determine a set of observers on a terrain represented by an elevation matrix such that these observers together can achieve a given visual coverage of the terrain. In (Franklin, 2002), the author presented a solution for this problem based on greedy strategy and, in this paper, we extend that method including a local search heuristic based on a swapping strategy to achieve a better terrain coverage using the same number of observers. As the main contribution of this paper, this heuristic was implemented in parallel using Graphics Processing Units (GPUs) and dynamic programming.

The extended method proposed in this paper, named *SiteGSM* (from Site using GPU's shared memory), was compared with some other methods for siting observers on terrain such as (Franklin, 2002; Franklin and Vogt, 2006; Magalhães et al., 2010a; Magalhães et al., 2011) and, as the tests showed, our method achieves a better performance than all of them: it is faster than (Magalhães et al., 2010a; Magalhães et al., 2011) (in some cases, more than 20 times faster than both) or it can use a smaller number of observers than (Franklin, 2002; Franklin and Vogt, 2006) to cover the terrain (which may represent an important improvement since the observer can be an expensive facility as, for example, a communication tower).

2 BACKGROUND

2.1 Terrain Visibility Definitions

A *terrain* represents a region of the earth surface where the terrain's value at any point is the elevation of the corresponding point of the earth surface above a reference ellipsoid called the *geoid* that represents sea-level. For this paper, a terrain is represented by a matrix of elevation posts on a square grid, whose vertical and horizontal spacing is uniform either in distance, e.g., 10m, or in angle, e.g. 1 arc-second.

An *observer* is a point in the space that "wants" to see or communicate with other points in the space, called *targets*. As usual, the notations for observer and target are O and T. The *base points* of O and T are the points on the geoid directly below O and T respectively, which are denoted as O_b and T_b. Both O and T are at height $h \geq 0$ above O_b and T_b. All symbols that appear in this work are shown in table 2.

The *radius of interest*, R, of O is the radius of the circle centered on O_b that contains all points that can be seen by the observer in the absence of obstructions. E.g., if O is a radio transmitter, R is a function of the transmitter power and receiver sensitivity. For convenience, R is usually compared to the distance between O_b and T_b rather than between O and T, which is equivalent when h is much smaller than the radius of the earth.

A target T is visible from O iff $|T_b - O_b| \leq R$ and there is no terrain point blocking the line segment, called the *Line of Sight (LOS)*, between O and T; see Figure 1. In this Figure, T_1 is visible from O but T_2 is not.

The *viewshed*, V, of O is the set of base points whose corresponding targets are visible from O. In general, V is stored as a bit matrix where 0 represents a non-visible point and 1 represents a visible point.

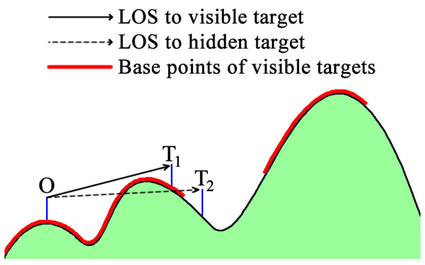


Figure 1: Visibility queries using a line of sight.

The *visibility index*, ω, of O is the number of targets that are visible from O. Points with a large ω are usually good candidate places to site observers in order to maximize the area of the terrain that is seen by at least one observer (Franklin and Ray, 1994).

The *joint viewshed*, $\mathcal{V}$, of a set of observers $\mathcal{S} = \{O_i\}$ is the union of the individual viewsheds V_i, i.e., the bitwise-*or* of their bit matrices.

The *joint visibility index*, Ω, of $\mathcal{S}$ is the number of targets in the terrain that are visible from at least one observer in $\mathcal{S}$. Usually, Ω is normalized as a percentage of the terrain area.

Multi-observer siting means optimizing the locations of a set of observers such that Ω is as large as possible. This is an NP-Hard problem (Nagy, 1994) and has important practical facilities-location applications, such as siting mobile phone towers, fire monitoring towers, and radar systems.

In this paper we will consider the following equivalent multi-observer siting problem: to obtain a set of observers whose joint visibility index Ω is, at least, a given percentage of the terrrain.

2.2 Parallel Programming Using General Purpose GPU

The programming architectures that allow using GPU units' parallel computing power (as, for example, the Compute Unified Device Architecture (CUDA) (NVIDIA, 2013)) have led to the development of many algorithms that achieve high computation performances.

CUDA has made possible the development of algorithms to solve time-consuming problems using the large number of parallel multiprocessors as well as the high memory bandwidth provided by GPUs. To accomplish high-performance computing, it is necessary to develop parallel algorithms that are totally or partially executed on the GPU.

The CUDA-enabled graphics cards are composed of multiple processors, more specifically, Single Instruction Multiple Data (SIMD) processors called Stream Multiprocessors (SMs), which allow the execution of multiple parallel threads. Thus, GPU processors can efficiently execute instructions involving many operations with data parallelism, i.e., when the same operation is applied to different data.

According to NVIDIA, GPUs can provide greater processing power than CPUs because they are specialized in performing parallel tasks involving many calculations. On the other hand, the CPUs are designed to perform tasks involving execution flow control and data cache. The physical difference between both architectures can be visualized in Figure 2: GPUs dedicate most of their area for processing units (in green), while CPUs dedicate most of their area for execution control and data cache (in yellow and orange, respectively).

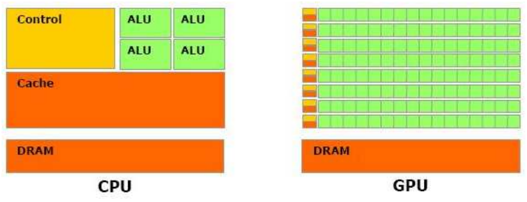


Figure 2: Comparison between GPUs and CPUs structures. Source: (NVIDIA, 2013)

A CUDA application consists in code that is executed on CPU and functions (called *kernels*) that are executed on GPU. The GPU is able to do parallel processing by creating threads such that each thread may execute the kernel operations in different data. Thus, the GPU is used as a coprocessor that is able to perform certain tasks more efficiently than the CPU.

3 RELATED WORK

There are some important work related to the problem addressed on this paper. In (Ben-Moshe, 2005), the author presented an algorithm to site facilities using an approach based on radio locator, frequency allocation and connectivity. The input includes a weighted set of demand locations, a set of feasible facility locations and a distance function that measures the cost of travel between a pair of locations. In (Ben-Shimol et al., 2007), the authors described an algorithm to site a minimal set of fixed-access relay antennas on a given terrain to generate the communication links between multiple base stations. Although the goal of these two works is not to achieve a given coverage of the terrain, they are related to the problem addressed on this paper because they use some important concepts related to our approach.

Other papers describing solutions to site observers on terrains are (Franklin and Vogt, 2004a; Franklin and Vogt, 2004b; Franklin and Vogt, 2006). They are based on the method *Site* proposed in (Franklin, 2002) which is described in section 4 and, as mentioned before, this method uses a greedy strategy to site observers on a terrain. In (Magalhães et al., 2010b), the method *Site* was extended to process huge terrains stored in external memory where the main idea is to subdivide the terrain in smaller pieces (subregions) and process each piece in the internal memory. In order to consider the influence of observers sited near to the borders of the subregions, each subregion is augmented with a band of width R (the observer radius of interest) around it. Additionally, the viewshed representation used in the original method *Site* was improved to require a smaller amount of memory.

In (Magalhães et al., 2010a; Magalhães et al., 2011) were presented two additional methods to site observer on terrains. They are described in section 5 and both are based on the method *Site*. In (Magalhães et al., 2010a), the original method *Site* was extended including a local search heuristic to try to reduce the number of observers selected to achieve the desired coverage. In (Magalhães et al., 2011), some routines used in the method proposed in (Magalhães et al., 2010a) were implemented in parallel using GPU.

In this paper, we present a more efficient parallel implementation of the method described in (Magalhães et al., 2010a) that uses a faster implementation (in parallel) of the local search. In section 5, we present a better description of the method proposed in this paper and also, the differences with the method presented in (Magalhães et al., 2011).

4 THE SITE METHOD

Considering that the observer siting problem is NP-Hard, Franklin (Franklin, 2002; Franklin and Vogt, 2006) proposed an approximate heuristic solution, called *Site*, to find a set of observers to cover the terrain. More precisely, this method uses a greedy approach to obtain a set $\mathcal{S}$ of observers such that a given percentage of the terrain is covered. Initially, $\mathcal{S} = \{\}$ and a set $\mathcal{P} = \{P_i\}$ of candidate observers is selected. Then, at each step, the P_i that will most increase the current joint visibility index of $\mathcal{S}$ is inserted into $\mathcal{S}$. As described in (Franklin, 2002), the details are as follows.

1. Estimate the visibility index of each point in the terrain M. More precisely, determine the points that have a certain minimum visibility index with a certain confidence level. This may be achieved by sampling random targets.

2. Compute $\mathcal{P} = \{P_i\}$ as the set of points with the largest visibility indices. However, do not select two points that are too close together, since their viewsheds will probably overlap considerably, and hence, some of them will be redundant.
3. Compute V_i, the viewshed of each P_i.
4. Initialize $\mathcal{S} = \{\}$. This will accumulate the set of actual observers $\mathcal{S} \subseteq \mathcal{P}$.
5. Initialize $\mathcal{V}$, the joint viewshed of $\mathcal{S}$, that is, the union of the viewsheds of all P_i in $\mathcal{S}$.
6. Repeat the following until a termination condition is satisfied. Typical conditions require $|\mathcal{S}|$ to achieve a certain maximum, or $\mathcal{V}$ to achieve a certain minimum of visible points.
   (a) Iterate through $\mathcal{P}$ to find the P_i that will cause the joint visibility index Ω to increase the most. That involves repeatedly counting the number of 1 bits in the union of the joint viewshed $\mathcal{V}$ and V_i.
   (b) Insert that P_i into $\mathcal{S}$ and update $\mathcal{V}$.

5 OBSERVER SITING IN GPU

In (Magalhães et al., 2010a) was presented an extension of the method *Site*, named *Site*+, where some heuristics were included to achieve a same terrain coverage using fewer observers. In order to make these heuristics more efficient, the method *SiteGPU* (Magalhães et al., 2011) implemented the following operations in GPU: (1) computing the visibility index of viewsheds; (2) finding the candidate observer that will most increase the visibility index of a joint viewshed; and (3) computing the union of viewsheds. In this implementation it was used the following strategy: all data (the viewsheds and joint viewsheds) were kept in the GPU global memory and the heuristics were executed on CPU. Thus, the GPU was used as a coprocessor to efficiently perform operations requested by the heuristics.

Also, to accelerate the viewshed operations, in *SiteGPU* the viewsheds were represented as a small piece of the terrain matrix. More precisely, each viewshed was represented by a $(2R+1)\times(2R+1)$ matrix where R is the radius of interest of the observer which is sited on this matrix center. That strategy improves the algorithm efficiency because the points outside the observer's radius of interest are not considered, since they are, by definition, not visible.

The main operation performed by *Site*+ is the *swap heuristic* that tries to increase the joint visibility index of the current partial solution without changing the number of observers selected. The basic idea is to check whether swapping a selected observer (in the current partial solution) with another observer didn't selected yet will increase the joint visibility index. This checking step considers all pairs composed by one observer in the partial solution and another one not in that solution and selects the pair that causes the highest contribution for the joint visibility index.

Notice that increasing the joint visibility index of a partial solution can reduce the number of steps required by the greedy strategy to achieve the final solution and, thus, the required coverage could be achieved using a smaller number of observers (that could be an important improvement since the observer can be an expensive facility, such as a cellular tower). But, on the other hand, this local search performs several viewshed operations and it is often the bottleneck of both *Site*+ and *SiteGPU*.

In this paper we propose a more efficient method, named *SiteGSM*, to site observers on terrain. It is based on *Site*+ and includes a faster implementation of the local search using dynamic programming and GPU's shared memory which is much faster than the GPU's global memory (used by *SiteGPU*). It is important to mention that both methods *Site*+ and *SiteGSM* obtain exactly the same solution, that is, the same number of observers (sited on same terrain places) and, of course, the same terrain coverage. The difference is that *SiteGSM* is much faster than *Site*+.

5.1 The Local Search - Swap

Given a set with n candidate observers, let $A = \{V_1, \cdots, V_n\}$ be the set with their corresponding viewsheds, that is, V_i is the viewshed of observer i and let S be a subset of A with k viewsheds representing an initial solution for the observer siting problem. The goal of the swap heuristic is to iteratively change S in order to increase the joint visibility index while keeping constant the number of observers in S.

The local search method is based on the concept of neighborhood of a solution which can be defined as follows: given a solution $S = \left\{V_{i_1}, \cdots, V_{i_k}\right\}$, a neighbor of S is a solution S' where an element of S is replaced by another element not in S. See Figure 3. In each iteration the current solution is replaced by its best neighbor (the one with highest visibility index).

The process of replacing the current solution with its best neighbor is repeated until it is obtained a solution having no better neighbor, which is a local optimum.

To simplify the notation, a solution $S = \{V_{i_1}, \cdots, V_{i_k}\}$ will be written as $S = \{i_1, \cdots, i_k\}$ indicating that the solution is, in fact, correspond to

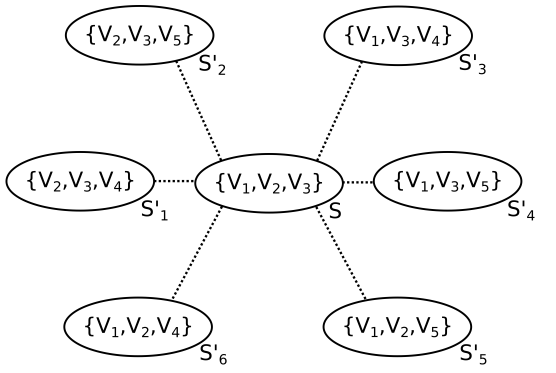


Figure 3: Given $A = \{V_1, V_2, V_3, V_4, V_5\}$, S' are the neighbors of the solution $S = \{V_1, V_2, V_3\}$.

the joint viewshed of the observers whose indices are $i_1, \cdots, i_k$. Thus, the heuristic may be implemented (sequentially) as follows: given the set of candidate viewsheds $A = \{V_1, \cdots, V_n\}$, let S be a solution composed of k viewsheds, i.e., $S = \{i_1, \cdots, i_k\}$ such that the joint viewshed of S is $V_{i_1} \oplus \cdots \oplus V_{i_k}$ where $\oplus$ represents the union operation between two viewsheds.

Furthermore, let $\mathcal{V}_{\bar{r}}$ be the joint viewshed of all viewsheds in S except V_{i_r}. In each iteration, the neighbors of S are generated in order to find the best solution for the next iteration. The visibility indices of the neighbor solutions are calculated by computing the number of visible points in $\mathcal{V}_{\bar{r}} \oplus V_j$ for $r = 1 \cdots k$ and $j = 1 \cdots n$ with $j \neq r$.

The most time-consuming step in this heuristic is computing the joint visibility index for each neighbor solution. Algorithm 1 presents the code for this step that computes the number of visible points in $\mathcal{V}_{\bar{r}} \oplus V_j$ (for all $r = 1 \cdots k$ and $j = 1 \cdots n; j \neq r$) and stores them in the element $Vix[r][j]$. In the next step, this matrix will be used to find the best neighbor of S.

For efficiency improvement, in this work the viewsheds are packed in 64-bit words (where each word represents the visibility of 64 points). Thus, the viewshed unions and visibility indices can be computed using, respectively, bitwise-or operator and bit population count functions, which are available in the hardware of most current computers.

5.2 An Efficient Swap Heuristic Implementation

Notice that, to generate $\mathcal{V}_{\bar{r}}$, for $r = 1 \cdots k$, Algorithm 1 performs $\Theta(k^2)$ union operations $\oplus$, where each union operation involves $\Theta(vsize)$ positions in the viewshed matrices. But, as described below, this step can be improved considerably using dynamic programming.

Given a solution $S = \{i_1, \cdots, i_k\}$, i.e., $V_{i_1} \oplus \cdots \oplus V_{i_k}$, for each $r \in \{1, \cdots, k\}$, we have $\mathcal{V}_{\bar{r}} = (V_{i_1} \oplus \cdots \oplus V_{i_{r-1}}) \oplus (V_{i_{r+1}} \oplus \cdots \oplus V_{i_k})$. Doing $\lambda_r^- = V_{i_1} \oplus \cdots \oplus V_{i_{r-1}}$ and $\lambda_r^+ = V_{i_{r+1}} \oplus \cdots \oplus V_{i_k}$ we can observe that both λ_r^- and λ_r^+ can be obtained by the following recurrence relations:

$$\lambda_1^- = \emptyset \text{ and } \lambda_r^- = \lambda_{r-1}^- \oplus V_{i_{r-1}} \text{for all } r \in 2, \cdots, k$$
$$\lambda_k^+ = \emptyset \text{ and } \lambda_r^+ = V_{i_{r+1}} \oplus \lambda_{r+1}^+ \text{for all } r \in 1, \cdots, k-1$$

Algorithm 1 Calculate the *Vix* matrix where *vsize* is the number of points in each viewshed, k is the number of observers in the solution S and n is the number of candidate observers. The output is matrix *Vix*, where $Vix[r][j]$ is the joint visibility index of a solution replacing observer r with j.

```
1: $Vix[k][n] \leftarrow \{\{0\}\}$
2: $\mathcal{V}[k][vsize] \leftarrow \{\{0\}\}$
3: for $r \leftarrow 1$ to $k$ do
4:     for $m \leftarrow 1$ to $k$ do
5:         if $r \neq m$ then
6:             for $w \leftarrow 1$ to $VSize$ do
7:                 $\mathcal{V}[r][w] \leftarrow \mathcal{V}[r][w]$ or $V[S[m]][w]$
8:             end for
9:         end if
10:    end for
11: end for
12: for $r \leftarrow 1$ to $k$ do
13:    for $j \leftarrow 1$ to $n$ do
14:        for $w \leftarrow 1$ to $vsize$ do
15:            $Vix[r][j] \leftarrow Vix[r][j] + (\mathcal{V}[r][w]$ or $V[j][w])$
16:        end for
17:    end for
18: end for
19: return Vix
```

For example, Figure 4 illustrates $\mathcal{V}_{\bar{r}}$ computation for $k = 5$: in this figure, each row r represents the value of $\mathcal{V}_{\bar{r}}$ where the elements in the left of the r-th column represents λ_r^- and in the right, λ_r^+. Notice that λ_4^- can be computed by joining the viewsheds V_{i_1} and V_{i_2} (which is the value of λ_3^-) with viewshed V_{i_3}. On the other hand, the λ^+ values can be computed in similar way using the reverse order.

Based on these recurrence relations, Algorithm 2 uses dynamic programming to compute a matrix $\mathcal{V}$ that stores $\mathcal{V}_{\bar{r}}$, for $r = 1 \cdots k$. Notice that this algorithm performs only $\Theta(k)$ viewshed unions and can replace the piece of code composed by lines 2 to 11 in Algorithm 1 where are performed $\Theta(k^2)$ viewshed unions.

In *SiteGSM*, the Algorithm 2 was implemented in GPU using the following strategy: the viewsheds were kept in GPU's global memory and, then, the

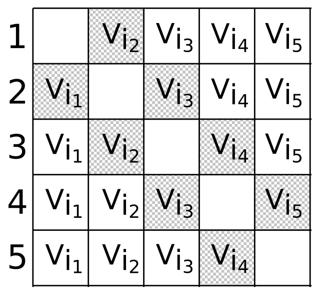


Figure 4: Matrix illustrating $\mathcal{V}_r$ in a solution with $k = 5$ observers.

union of viewsheds (loop in lines 3, 9 and 15 of Algorithm 2) were performed using GPU's threads, that is, each thread performs a bitwise-or operation with one element of a viewshed and the corresponding element of another viewshed.

Algorithm 2 Compute the matrix $\mathcal{V}$ that stores $\mathcal{V}_r$, for $r = 1 \cdots k$ using a dynamic programming strategy.

```
 1: $\mathcal{V}_1[k][vsize] \leftarrow \{\{0\}\}$
 2: for $r \leftarrow 2$ to $k$ do
 3:     for $w \leftarrow 1$ to $vsize$ do
 4:         $\mathcal{V}_1[r][w] \leftarrow \mathcal{V}_1[r-1][w]$ or $V[S[r-1]][w]$
 5:     end for
 6: end for
 7: $\mathcal{V}_2[k][vsize] \leftarrow \{\{0\}\}$
 8: for $r \leftarrow k-1$ to 1 do
 9:     for $w \leftarrow 1$ to $vsize$ do
10:         $\mathcal{V}_2[r][w] \leftarrow \mathcal{V}_2[r+1][w]$ or $V[S[r+1]][w]$
11:     end for
12: end for
13: $\mathcal{V}[k][vsize] \leftarrow \{\{0\}\}$
14: for $r \leftarrow 1$ to $k$ do
15:     for $w \leftarrow 1$ to $vsize$ do
16:         $\mathcal{V}[r][w] \leftarrow \mathcal{V}_1[r][w]$ or $\mathcal{V}_2[r][w]$
17:     end for
18: end for
```

After computing $\mathcal{V}_r$, the next step is to compute the joint visibility index of the neighbor solutions, as performed by lines 12 to 18 in Algorithm 1. A staightforward implementation of this step in GPU was presented in (Magalhães et al., 2011), where all viewsheds are kept in the GPU's global memory and, then, each element of matrix *Vix* (that stores the joint visibility index) is computed using a parallel algorithm to overlap a pair of viewsheds followed by a parallel reduction operation to determine the number of visible points. However, this strategy does not take advantage of the GPU performance efficiently because it requires too many accesses to the global memory, which is much slower than other memories such as the shared memory.

In order to make a better use of the GPU memory hierarchy, we propose a new strategy based on a fast GPU matrix multiplication algorithm. Notice that the joint visibility index, that is, the *Vix* matrix, is obtained in lines 12 to 18 of Algorithm 1, where the matrices $\mathcal{V}$ and V are overlapped using a bitwise-or operation. In this case, the two matrices are swept in a row major order, but the matrices could be reorganized such that the overlapping could be computed using an access pattern similar to matrix multiplication. More precisely, line 15 in Algorithm 1 can be replaced with

$$Vix[r][j] \leftarrow Vix[r][j] + \mathcal{V}[r][w] \ \textbf{or} \ V^T[w][j]$$

where V^T is the transposed matrix of V.

Thus, the *Vix* matrix can be computed using a simple adaptation of some very fast algorithm for matrix multiplication in GPU. In particular, we adapted the algorithm presented in (NVIDIA, 2013), replacing the multiplication operation by a bitwise-or followed by a binary population count operation. This algorithm subdivides the matrices into blocks, which are loaded iteratively in the GPU's shared memory as the multiplication process is performed. Therefore, most of the algorithm accesses are to the shared memory which is much faster than the global memory.

Additionally, since the viewsheds matrices are, usually, sparse (the points outside the observer's radius of interest are always non visible), we adapted the matrix multiplication algorithm to avoid loading and processing matrix blocks where all the elements are 0.

6 EXPERIMENTAL RESULTS

The method *SiteGSM* was implemented in C++/CUDA and compiled using *nvcc* 4.0 with maximum optimization level (-O3). It was compared against *SiteGPU* (Magalhães et al., 2011) and against *Site+* (Magalhães et al., 2010a), a sequential CPU version with no use of dynamic programming. The tests were executed on a computer with Dual Intel Xeon E5-2687 3.1GHz, 128GiB of memory and GPU NVidia Tesla Kepler K20x with 6GiB of global memory, 48KB of shared memory per block and 2688 CUDA processing cores running Ubuntu 12.04 LTS.

The tests used different datasets obtained from NASA SRTM webpage: a terrain with 1201×1201 points (90-meter resolution SRTM3 terrain) representing a region of the Minas Gerais state in Brazil and another terrain with 3601×3601 points (30-meter resolution SRTM1 terrain) representing a region of the New Jersey state in USA.

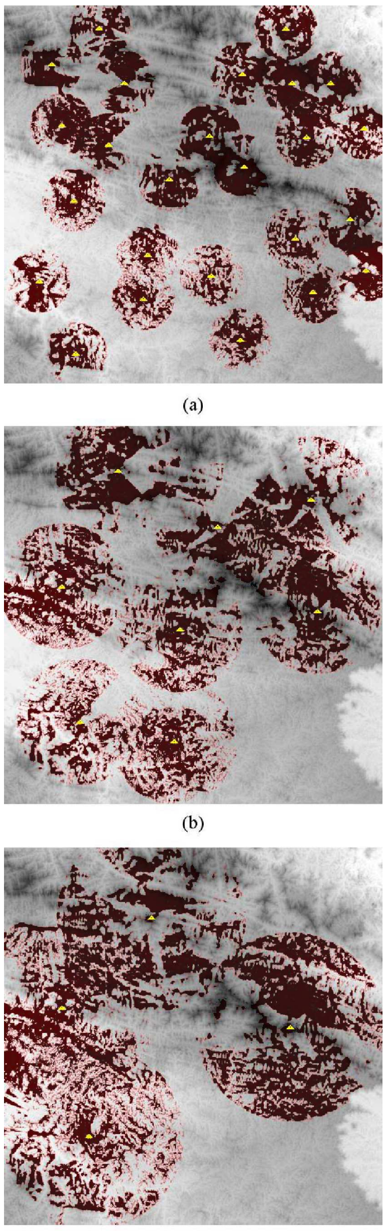
The initial set $\mathcal{P}$ of candidate observers for each terrain was computed using steps 1 to 3 of the *Site* method proposed by (Franklin, 2002) and described in section 4. The *Site* was set up to select 1000 candidates for the smaller terrain and 3000 candidates for the larger one, as in (Magalhães et al., 2011). Also, the viewsheds of all candidate observers were computed using the same viewshed algorithm used by *Site*, with the observers and target points sited 30 meters above the terrain. This value was chosen because, in general, it is the height of real towers, as for example, communication antennas (Delmellea et al., 2005).

The observer siting methods were tested using radii of interest 100, 200 and 300 points for the first terrain; and 200, 300 and 400 points for the second terrain. For each terrain, the desired coverages (joint visibility indices) were 75%, 85% and 95% of the terrain area. Figure 5 shows three examples, with radii of interest 100, 200 and 300 and desired coverage of 25%, indicating how observers were sited on the terrain.

Table 1 presents the processing time (in seconds) spent by each method. Column #Obs shows the number of observers sited in each case and the *SiteGSM*'s speedup is shown in parentheses. The symbol * represents a case for which the set $\mathcal{P}$ of candidate observers is not enough to achieve the desired coverage. The symbol ∞ indicates those cases where the processing time is greater than 7 hours.

As the tests showed, in all cases, the method *SiteGSM* proposed in this paper was faster than *SiteGPU* (Magalhães et al., 2011) - in some cases, more than 20 times - and, as expected, much faster than the *Site*+ (more than 1200 times). Notice that, in almost all cases, the higher the desired coverage, the higher the speedup. This can be explained by the fact that higher coverages require more observers and consequently more viewsheds must be processed.

While much faster than *Site*+, *SiteGSM* obtains the same solution as the former and, as presented in (Magalhães et al., 2010a), the *Site*+ can achieve similar terrain coverage as the original *Site* using about 10% less observers. This can be an important (economic) improvement since the observers can represent some expensive facility, as for example, a communication antenna.


(a)

(b)

(c)

Figure 5: Observers sited in terrain 1201^2 with desired coverage 25% and radii of interest: 100 (a), 200 (b) and 300 (c). The observers are indicated by yellow triangles and visible points are showed in red.

7 CONCLUSION

We presented an efficient implementation of a heuristic to site observers on terrains to achieve a given coverage of this terrain. The method is based on an extension of *Site* (Franklin, 2002), where it was included a local search strategy that tries to improve the coverage obtained by a fixed number of observers. This lo-

Table 1: Processing time (in seconds) of three methods: two parallel methods using GPU (*SiteGSM* and *SiteGPU*) and a sequential one (*Site*+) to site observers on terrains with different sizes considering different radii of interest (*R*) to achieve some desired coverages (Ω).

Ter.	*R*	Ω	#Obs.	Processing Time (in sec.)				
				SiteGSM	*SiteGPU*		*Site*+	
1201 × 1201	100	75%	162	12	180	(15.0)	11010	(917.5)
		85%	299	33	545	(16.5)	∞	(-)
		95%	*	*	*	(-)	*	(-)
	200	75%	55	3	35	(11.7)	1304	(434.7)
		85%	97	6	104	(17.3)	4020	(670.0)
		95%	323	48	888	(18.5)	∞	(-)
	300	75%	34	2	19	(9.5)	479	(239.5)
		85%	62	4	70	(17.5)	1826	(456.5)
		95%	216	28	566	(20.2)	19408	(693.1)
3601 × 3601	200	75%	81	76	422	(5.6)	∞	(-)
		85%	97	110	708	(6.4)	∞	(-)
		95%	125	183	1341	(7.3)	∞	(-)
	300	75%	36	28	160	(5.7)	∞	(-)
		85%	42	41	276	(6.7)	∞	(-)
		95%	54	66	523	(7.9)	∞	(-)
	400	75%	20	14	81	(5.8)	14867	(1061.9)
		85%	24	19	124	(6.5)	22869	(1203.6)
		95%	30	30	270	(9.0)	∞	(-)

cal search was implemented in parallel using graphics processing units and also dynamic programming.

As the tests showed, the method described in this paper is more efficient than a previous method presented in (Magalhães et al., 2011) and, in some cases, more than 20 times faster. Also, it is up to 1200 times faster than the corresponding sequential implementation.

This is an interesting contribution since the extended method *Site*+ using local search heuristic can achieve a desired coverage using a smaller number of observers than the original *Site* and thus, better solutions can be generated faster.

As a next step, to improve the proposed method performance even more, we will try to reduce the viewshed matrix size using only a bounding box containing the observer radius of interest. Thus, it will be possible to reduce the volume of data to be processed and, in this case, it seems that we could compute the joint visibility indices adapting a GPU sparse matrix multiplication algorithm.

ACKNOWLEDGEMENTS

This work was partially supported by FAPEMIG, CNPq, CAPES and NSF IIS-1117277.

REFERENCES

Andrade, M. V. A., Magalhães, S. V. G., Magalhães, M. A., Franklin, W. R., and Cutler, B. M. (2011). Efficient viewshed computation on terrain in external memory. *GeoInformatica*, 15(2):381–397.

Ben-Moshe, B. (2005). *Geometric Facility Location Optimization*. PHD thesis, Ben-Gurion University, Israel, Department of Computer Science.

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

Delmellea, E. M., Rogersonb, P. A., Akellad, M. R., Battae, R., Blattf, A., and Wilsonf, G. (2005). A spatial model of received signal strength indicator values for automated collision notification technology. *Transportation Research Part C: Emerging Technologies*, 13(5-6):432–447.

Franklin, W. R. (2002). 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.

Franklin, W. R. and Ray, C. (1994). Higher isn't necessarily better: Visibility algorithms and experiments. In *Advances in GIS research: sixth international symposium on spatial data handling*, volume 2, pages 751–770. Edinburgh.

Franklin, W. R. and Vogt, C. (2004a). Efficient multiple observer siting on large terrain cells. *GIScience 2004.*

Franklin, W. R. and Vogt, C. (2004b). Multiple observer siting on terrain with intervisibility or lo-res data. In *XXth Congress, International Society for Photogrammetry and Remote Sensing, Istanbul*, pages 12–23.

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

Laurini, R. and Thompson, D. (1992). *Fundamentals od Spatial Information Systems*. Academic Press.

Li, Z., Zhu, Q., and Gold, C. (2005). *Digital terrain modeling: principles and methodology*. CRC Press.

Magalhães, S. V. G., Andrade, M. V. A., and Ferreira, C. (2010a). Heuristics to site observers in a terrain represented by a digital elevation matrix. In *GeoInfo*, pages 110–121.

Magalhães, S. V. G., Andrade, M. V. A., and Ferreira, R. S. (2011). Using gpu to accelerate heuristics to site observers in dem terrains. In *IADIS Applied Computing (AC 2011)*, pages 127–133. Rio de Janeiro.

Magalhães, S. V. G., Andrade, M. V. A., and Franklin, W. R. (2010b). An optimization heuristic for siting observers in huge terrains stored in external memory. In *Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*, pages 135–140. IEEE.

Nagy, G. (1994). Terrain visibility. *Computers & graphics*, 18(6):763–773.

NVIDIA (2013). CUDA programming guide. *NVIDIA Corporation, July*.

Table 2: Table of notations.

Symbol	Description
O	Observer
T	Target
O_b	Observer's base point
T_b	Target's base point
h	Height of an observer or target above terrain
R	Radius of interest of an observer
V	Viewshed of an observer
ω	Visibility index of an observer
$\mathcal{S}$	Set of observers
$\mathcal{V}$	Joint viewshed of a set of observers
Ω	Joint visibility index of a set of observers
$\mathcal{P}$	Set of candidate observers
n	Number of candidate observers
A	Set of candidate observers
S	Subset of A
k	Number of observers in S
S'	Neighbor solution of S
$i_1, \cdots, i_k$	Observers Indices
$\oplus$	Union operation between two viewsheds
V_{i_k}	Viewshed of the observer k in S
$\mathcal{V}_{\bar{r}}$	Joint viewshed of all viewsheds in S except V_{i_r}
$vsize$	Number of points in each viewshed
$Vix[r][j]$	Joint visibility index of a solution replacing observer r with j
λ_r^-	Union between the viewsheds $V_{i_1} \cdots V_{i_{r-1}}$
λ_r^+	Union between the viewsheds $V_{i_{r+1}} \cdots V_{i_k}$