RESEARCH ARTICLE

Data relocation approach for terrain surface analysis on multi-GPU systems: total viewshed problem as a case study

ARTICLE HISTORY
Compiled October 28, 2020

ABSTRACT
Digital Elevation Models (DEMs) are important datasets for modelling line-of-sight phenomena such as radio signals, sound waves and human vision. These are commonly analysed using rotational sweep algorithms. However, such algorithms require large numbers of memory accesses to 2D arrays which, despite being regular, result in poor data locality in memory. This paper proposes a new methodology called skewed Digital Elevation Model (sDEM), which substantially improves the locality of memory accesses and largely increases the inherent parallelism involved in the computation of rotational sweep-based algorithms. In particular, sDEM applies a data restructuring technique before accessing the memory and performing the computation. To demonstrate the high efficiency of sDEM, we use the problem of total viewshed computation as a case study. Different implementations for single-core, multi-core, single-GPU and multi-GPU platforms are provided. We conducted two experiments in which sDEM is compared with (i) the most used geographic information systems (GIS) software and (ii) the state-of-the-art algorithm. In the first experiment, sDEM is in average 8.8x faster than current GIS software despite being able to consider only few points because of their limitations. In the second experiment, sDEM is 827.3x faster than the state-of-the-art algorithm, considering the best studied case.

KEYWORDS
Data relocation, Digital Elevation Model (DEM); Geographic Information Systems (GIS); Total viewshed; Visibility analysis

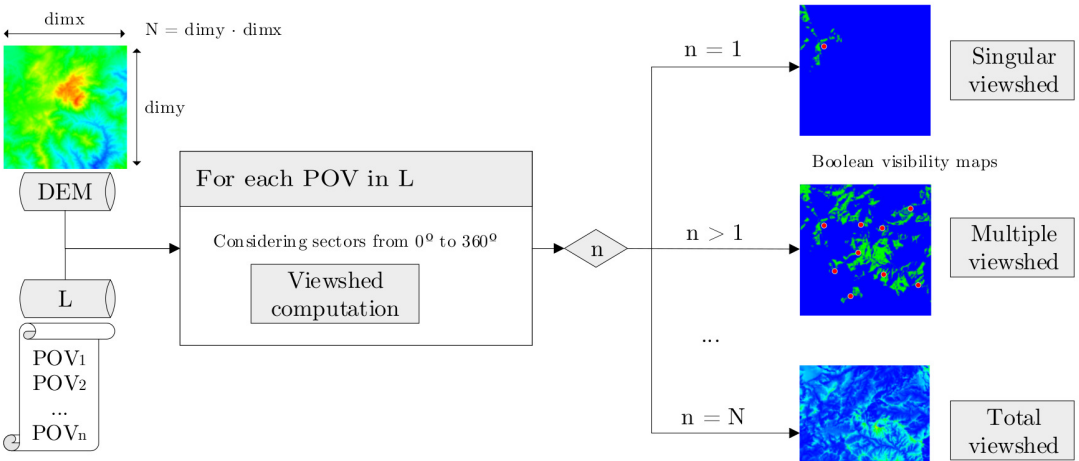
1. Introduction

There are some problems of terrain surface analysis which require the evaluation of the data around a reference point. This is the case of the viewshed computation where the reference point is usually called point of view (POV) in the Digital Elevation Model (DEM). This topic has been thoroughly studied in the recent literature (Wang and Dou 2019, Dou *et al.* 2019), being usually addressed using rotational plane sweep-based algorithms, or just rotational sweep (Choset *et al.* 2005). In particular, a line is traced from the POV, which works as a vertex in the plane. This line will be rotated by 2π radians and all the points that cross that line are analyzed with respect to the vertex. Another related approach involves the discretization of the plane in azimuthal sectors radially starting from the reference point (Tabik *et al.* 2014). Every azimuthal sector is represented by an axis placed in its centre, and points crossed over by the axis are compared to the reference location. In this approach, the statistical representativeness of every axis progressively decreases as we move away from the vertex since the width of the sector increases linearly with the radius. However, in most cases, the required accuracy will also be reduced to the same extent. This issue is exploited in some

situations where the reference location works as the transmitter or receiver of certain signals whose strength decreases with the square of the distance, such as radio signals, sound waves, and line of sight.

The azimuthal sector discretization method is also included in some visibility modules from geographic information systems (GIS) applications such as ArcGIS (ESRI 2010), GRASS GIS (Neteler *et al.* 2012), and Google Earth. A very common tool provided by these programs is the viewshed computation, which is of great interest in many areas such as telecommunications, environmental planning, ecology, tourism, and archaeology (Cauchi-Saunders and Lewis 2015, Qarah and Tu 2019, Wang and Dou 2019). In these diverse fields, knowing the visibility in terrain is almost a requirement to achieve optimal results.

There exists three types of viewshed problems in the literature (Figure 1) depending on the number of observers considered for calculation: (i) singular, (ii) multiple, and (iii) total viewshed. The first is the simplest visibility problem which comprises the computation of the viewshed from one single observer at a certain elevation with respect to the ground. To address this problem, the DEM is usually divided into angular sectors in the plane around the POV. Then, different lines of sight (LoS), which correspond to the axis of every sector, start from the POV and are radially distributed towards the most distant areas. Every target point crossed over by this line is sequentially compared to the POV based on the elevation values in order to compute its visibility. The result is a boolean map containing visible and non-visible points from the POV in the terrain. Likewise, the multiple viewshed for several POVs can also be obtained by repeating the above procedure on every one of them and adding up the viewshed results obtaining a similar map. A complete visibility analysis involves knowing the viewshed of every point in the terrain in all directions. This information can be used to address well-known problems such as siting multiple observers (Cervilla



DEM	Digital Elevation Model
N	Total number of points in DEM
L	List of target POVs in DEM
POV	Point of view in DEM
n	Total number of POVs in L

Figure 1. Illustration of the different viewshed problems existing in the literature related to singular, multiple, and total viewshed. This classification is based on the number of target points of view (POVs) in the Digital Elevation Model (DEM) chosen as input, which produces different outputs for each of them. Our work focus on the most computational demanding problem which is the total viewshed computation.

et al. 2015), and path planning with surveillance aims (Li *et al.* 2010). The first one is related to finding the fewest possible number of POVs providing maximum viewshed for a certain area. The second one involves designing a near optimal path aiming to achieve maximum terrain coverage. Both problems would be substantially simplified if the visibility of every point in the area is known beforehand (Franklin and Ray 1994). This problem is known as total viewshed and it is one of the most challenging visibility calculations due to its complexity and high computational cost. It involves obtaining the viewshed for each point in the DEM considered as POV and then, accumulate their visibility results. This results in a new map where every point contains the viewshed value of the corresponding location in the terrain measured, e.g., in km^2. Nowadays, most GIS software include specific modules for singular viewshed computation. Few GIS software provide multiple viewshed calculation but using task queues, which demonstrate poor computational performance as it is based on single point viewshed.

In this work, we propose a new methodology called skewed Digital Elevation Model (sDEM) considering the total viewshed problem as a case study. It involves a complete restructuring of the DEM data in memory carried out prior to the computation of the total viewshed. The DEM is transformed into a new structure named skwDEM in which the data are aligned in memory to improve data locality in accessing the memory and, therefore, increasing the speed of processing. Through this approach, it is unnecessary to apply common techniques that reduce computational cost in total viewshed problems, such as considering a maximum visibility distance within a circular area around every POV. This methodology could improve the performance of other applications that analyse relevant topographic features of the terrain surface such as slope and elevation.

The contributions of the paper are as follows:

- We design a new methodology named sDEM (skewed Digital Elevation Model) for faster processing in terrain surface analysis which largely improves data locality in memory. In particular, this approach fully exploits the intrinsic parallelism of the total viewshed computation, achieving maximum performance through efficient memory access.
- We present different implementations for single-core, multi-core, single-GPU, and multi-GPU platforms. Each of them are compared with the state-of-the-art regarding the total viewshed problem.

The remainder of this paper is organized as follows: Section 2 presents the state-of-the-art in regards to the total viewshed computation. Section 3 reviews the background related to this research. Section 4 explains the proposed sDEM methodology for computing the total viewshed and presents the implementation for multi-GPU systems. Section 5 compares the sDEM algorithm with the most used GIS software and the state-of-the-art. Section 6 discusses the results of this study.

2. Related work

Terrain visibility, commonly known as viewshed analysis, is related to the problem of obtaining the area of the terrain visible from a given POV located at a certain elevation above the ground. This issue has been widely studied for many years given the mass of interpolation computations required to produce precise results (Atallah 1983, Cabral *et al.* 1987, Fisher 1992, Franklin and Ray 1994, Floriani and Magillo 1994). Authors

usually use LoS-based algorithms such as R3, R2 or DDA (Franklin *et al.* 1994, Kaučič and Zalik 2002). These methods project rays starting from the observer toward the boundary of the DEM to obtain the points included in processing. Another related strategy is XDraw (Franklin *et al.* 1994) which computes the LoS function in stages arranged as concentric squares centered on the position of the observer.

Many algorithms are developed for calculating the viewshed from one single POV, or from a small number of POVs at best. In Gao *et al.* (2011) a singular viewshed implementation was developed for built-in GPU systems based on the LoS method and texture memory with bilinear interpolation. They achieve a speed-up up to 70x with respect to the sequential CPU implementation. The GPU implementation proposed by Stojanović and Stojanović (2013) achieves remarkable results in obtaining a boolean raster map instead of a map containing viewshed values. A novel reconfiguration of the XDraw algorithm for GPU context is described in Cauchi-Saunders and Lewis (2015) which outperforms CPU and GPU implementations of well-known viewshed analysis algorithms such as R3, R2, and XDraw. Furthermore, an efficient implementation of the R2 viewshed algorithm is carried out in Osterman *et al.* (2014) with particular focus on input/output efficiency and obtaining significant results in contrast to R3 and R2 sequential CPU implementations. The algorithm described in Zhao *et al.* (2013) focuses on a two-level spatial domain decomposition method to speed-up data transfers and thus performs better than other well-known sequential algorithms. Other extended approaches are focused on obtaining the viewshed for multiple points (Strnad 2011, Song *et al.* 2016). More recent research is presented in Wang and Dou (2019) where fast candidate viewpoints are obtained for multiple viewshed planning. These authors have also investigated parallel XDraw analysis (Dou *et al.* 2018, 2019) improving the results obtained by previous XDraw algorithms.

Nevertheless, few studies address the total viewshed computation problem and most of them focus on tackling a simplified version of it. For example, the total viewshed in Dungan *et al.* (2018) is obtained by drastically reducing the number of grid points to be processed. Likewise, the approach used in Brughmans *et al.* (2018) computes the visibility of small areas and not for specific points. So far, the only algorithm that addresses the total viewshed problem on high resolution DEMs is the TVS algorithm proposed by Tabik *et al.* (2013, 2014). It considers the closest points to the line of sight as a sample set of points stored in a structure called band of sight (BoS). In this approach, the distance to the axis determines the number of points in the BoS (Floriani and Magillo 1994). Maximum memory utilization was achieved by reusing the points contained in the list and obtaining the viewshed for every aligned point in the particular sector. However, this algorithm has important limitations:

- For a given POV, the analysis of the points inside the BoS is performed sequentially because it is impossible to know whether a target point is visible without knowing the state of the previous one.
- The implementation of the data reuse of the BoS produces a significant overhead caused by the selection of the corresponding points for every direction.
- It is not appropriate for implementation on high-throughput systems such as GPUs and Xeon-Phi architectures, because parallelism is limited to sector level.

In this study, we propose computing total viewshed based on a compact and stable data structure with the aim of increasing data and computation reuse, outperforming the most commonly used GIS software. Our proposal will also be compared to the TVS algorithm (Tabik *et al.* 2014).

3. Background: viewshed analysis

In this section, the basic concepts of the viewshed analysis are presented. Section 3.1 describes the singular viewshed problem. Section 3.2 explains the complexity of the viewshed analysis. Section 3.3 deals with the total viewshed problem.

3.1. *Singular viewshed*

As a starting point, most viewshed computation algorithms perform an azimuthal partition of the area. This division is carried out by splitting the area that surrounds the observer (POV) into ns azimuthal sectors. Every sector is represented by its axis, and the closest points to this structure are usually considered for the viewshed analysis.

Algorithm 1 presents a general approach for the calculation of the viewshed (VS) considering one single POV on a particular DEM using a regular Cartesian grid. The coordinates and height of the observation point are denoted as i, j, and POV_h, considering the observer is slightly above the ground (h); $axis$ is the set of points included in a particular sector s, and $selectAxisPointSet$ adds candidate points within the sector. This methodology analyzes the visibility in all the sectors based on the $linearViewshed$ function (Algorithm 2). It calculates the visible area for a certain sector with respect to an observer located on the axis of that sector. In practice, a

Algorithm 1 $singularViewshed(i, j, h)$

```
global point POV = DEM[i][j]
POV_h += h
float VS = 0
for s = 0, ns do
   pointSet axis = selectAxisPointSet(DEM, POV, s)
   VS += linearViewshed(axis, true)   // forward
   VS += linearViewshed(axis, false) // backward
end for
VS *= (π/ns)   // Papus theorem scaling
```

Algorithm 2 $linearViewshed(axis, forward)$

```
global bool visible = true
global float maxθ = −∞   // Max. angle
global pointSet visibleSet = emptySet
do
   T = axis.next()
   pointViewshed(axis.POV, T)
while T != axis.last()
return visibleSet.measure()
```

Algorithm 3 $pointViewshed(POV, T)$

```
float dist = sqrt((T_j − POV_j)^2 + (T_i − POV_i)^2)
float θ = (POV_h − T_h) / dist
bool prevVisible = visible
if (θ > maxθ) visible = 1
bool startRS = !prevVisible & visible
if (startRS) dist0 = dist
bool endRS = prevVisible & !visible
if (endRS) visibleSet.add(dist, dist0)
```

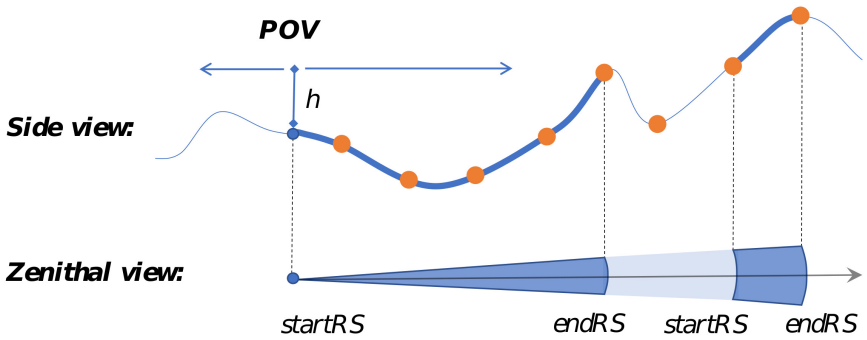


Figure 2. Side and zenithal views for a particular POV, with a specific height h, from which two segments are visible (represented both by blue thick segments). The corresponding visible ring-sectors are obtained for each one considering their starting points ($startRS$) and ending points ($endRS$).

linearized set of target points is considered as the visibility of every remaining point in the $axis$ is computed following the direction from the nearest point to the furthest point.

As shown in Algorithm 3, a target point T is visible from the POV if its angular altitude θ is higher than all the previous ones considered in the $axis$ ($max\theta$). Visible points on the $axis$ are included in a set of points called $visibleSet$. In order to improve efficiency, only the starting and ending points of a segment are measured ($visibleSet.add$) and considered in processing. This methodology uses $startRS$ and $endRS$ variables to indicate whether a sequence of visible points has been found. First, the distance between the POV and the first point found belonging to a visible section is measured in $dist0$. Then, the final visible point of this visible segment is found and its distance with respect to the POV is measured ($dist$). This process is repeated until all points on the axis are analyzed as shown in the side view in Figure 2. The projection of all visible segments throughout the sector results in the generation of visible sections, commonly known as sector rings. The area of every visible section (A_{vs}), considering sectors of one degree of opening, is computed as follows:

$$A_{vs} = (\pi/360) \cdot (R^2 - r^2)$$

where R and r are the radius of the visible ring sector related to the $endRS$ and $startRS$ values, respectively, with respect to a particular POV. Considering all the above mentioned, the viewshed for one single location results from the addition of the area of every visible section ($visibleSet.measure$). This approach reduces memory accesses and mathematical calculations as proven in Tabik *et al.* (2014).

3.2. *Real problem complexity*

The rotational sweep method significantly reduce the complexity of the calculations required in the viewshed analysis from at least $O(N)$ to $O(s \cdot N^{1/2})$, where N is the size of the 2D array measured in points. Considering that a common DEM widely exceeds several millions of points and the discretization of the sector is rarely above the required accuracy, the accomplished reduction is between one and three orders of magnitude (Stewart 1998). For example, the complexity to obtain the viewshed using point–to–point algorithms such as R3 is $O(N^{3/2})$, whereas it is reduced up to $O(s \cdot N^{1/2})$ using rotational sweep. However, there remains a large number of oper-

ations, which makes parallelism and supercomputing highly recommended for these sorts of approaches. In particular, one of the visibility problems that was considered unapproachable is the total viewshed computation, which is described in detail below.

3.3. *Total viewshed*

The problem of addressing the viewshed for all points in a particular area, represented by a DEM with N points of observation, was almost impossible not long ago. The computation of the singular viewshed is very high demanding in computational terms and, therefore, repeating this procedure for every single point in the DEM would have been incredibly time-consuming on CPU. The inherent complexity of the problem is up to $O(N^3)$ if a non-optimized approach is applied N times over a problem of $O(N^2)$ complexity. Nevertheless, using rotational sweep, the problem complexity can be reduced up to $O(s \cdot N^{3/2})$. Algorithm 4 introduces the steps required to address the total viewshed problem, considering a DEM represented by a Cartesian grid with *dimy* x *dimx* points. The viewshed value, i.e, the visible terrain area for every point in the DEM is stored in the total viewshed matrix (TVS). This matrix has the same dimensions as the DEM and every cell is filled with a particular viewshed value obtained after performing the corresponding singular viewshed computation (Algorithm 1). Some authors have observed that swapping the loops of Algorithm 4 and Algorithm 1 can significantly improve data locality in memory (Stewart 1998, Tabik *et al.* 2014). This is one of the pillars on which our proposal is based.

Algorithm 4 Total viewshed problem

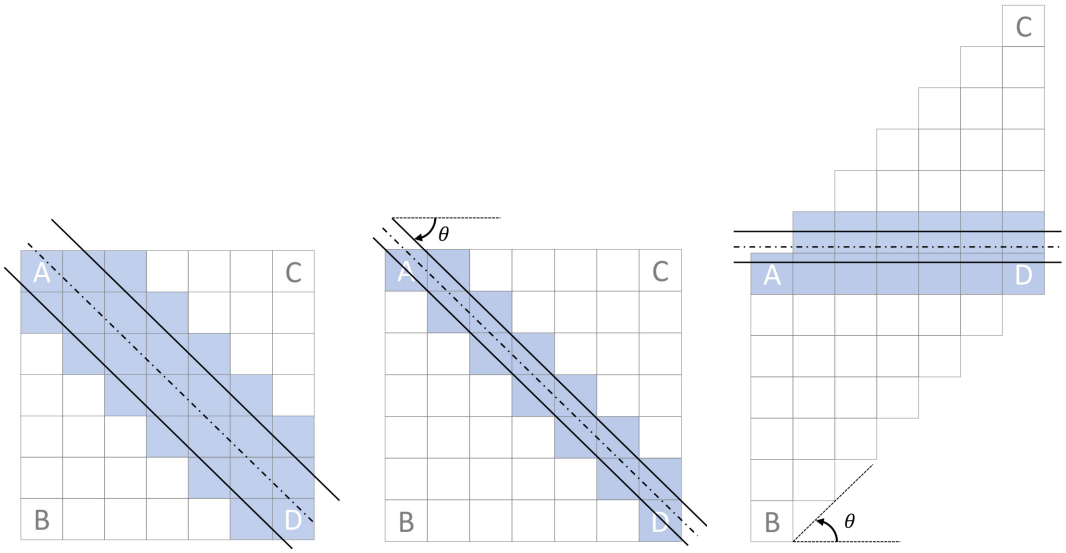
```
for i = 0, dimy do
  for j = 0, dimx do
    TVS[i][j] = singularViewshed(i, j, h)
  end for
end for
```

4. sDEM: a grid reorganization approach

This section describes our proposed methodology called skewed Digital Elevation Model (sDEM) designed to improve data locality in memory for terrain surface analysis using the total viewshed computation problem as a case study. This approach takes into account the technical features of CPU (host) and GPU (device) processing units to take full advantage of the intrinsic parallelism of the total viewshed computation. For the sake of simplicity, in the rest of the paper we will refer to the original DEM as DEM and original DEM. We will refer to the modification of this structure as skwDEM.

4.1. *Proposed methodology*

The data reuse is the key to achieve proper optimization in the total viewshed computation so first, we have to introduce the structure that will manage this important process. This structure is called band of sight (BoS) and it will serve as the basis for the process of restructuring the DEM for every POV and sector. The BoS is used to find the closest points to the line of sight for any reference POV and given sector.



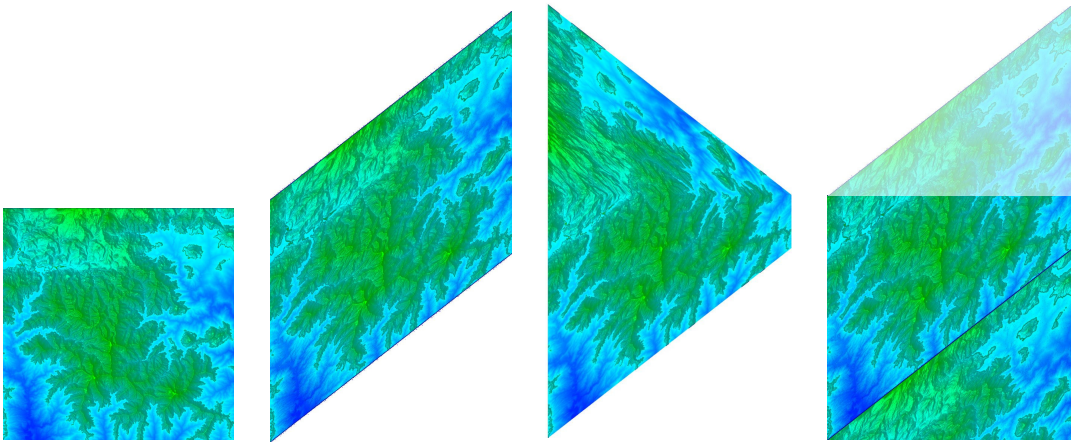
(a) DEM; 2.5 $\sqrt{N}$ BoS (b) DEM; $\sqrt{N}$ BoS (c) skwDEM; $\sqrt{N}$ BoS

Figure 3. Three examples of band of sight (BoS), each one with different width and layout on the plane, considering a sector $s = 45^o$ for simplicity. The cells of the DEM that contribute to the BoS computation are shown in blue. (a) and (b) show two different BoS widths for the *dimy* x *dimx* original DEM; whereas (c) presents the restructuring of the $2 \cdot dimy$ x *dimx* skwDEM considering a BoS width of $\sqrt{N}$. For the sake of clarity, A-D labels are located in the corner points of the DEM so that the restructuring approach can be visualized, only one BoS is shown as well.

Thus, choosing the right size for this structure is vital to improve data locality in accessing the memory. Figure 3(a) and 3(b) show two BoS widths of 2.5 $\sqrt{N}$ and $\sqrt{N}$, respectively, considering a sector $s = 45^o$ for the sake of simplicity; only the points of the grid in dark color are considered for the visibility computation. The extensive statistical study conducted in Tabik *et al.* (2014) proves that the size of this structure is not a determining factor, as long as it is of the order of $\sqrt{N}$. Therefore, our sDEM proposal uses the latter BoS size in order to address the data repetitive problem.

Once the BoS size has been fixed, complete relocation of the data is performed from the original DEM (Figure 3(b)) to the skwDEM (Figure 3(c)). This is a new DEM which is more or less skewed in shape depending on the width chosen for the BoS. The use of this structure allows the exploitation of the existing parallelism without adversely affecting the precision of the results based on the following considerations:

- We apply the Stewart sweep method (Stewart 1998) that places first the loop that runs through all the sectors instead of all the points in the DEM. It is the only model that guarantees the reuse of data aligned in every direction.
- Given a sector, all the possible parallel bands of sight that cross the DEM are built simultaneously. We apply the interpolation method based on a simplified version of the Bresenham's algorithm, which is commonly used for line rasterization. This algorithm was chosen for its high speed as well as maintaining sufficient fidelity to the problem under consideration.
- For each sector, the relocation is applied only once to the entire DEM. For example, in the particular case of considering 180 sectors, the data relocation takes place 180 times and always before starting the viewshed computation. Thus, the relocation only depends on the selected sector. Another advantage is that this method is especially appropriate for processing on the GPU. It aims



(a) DEM (b) Reorganized skwDEM (c) Compacted$_1$ skwDEM (d) Compacted$_2$ skwDEM

Figure 4. The original DEM and three possible results when applying our array redistribution procedure considering sector $s = 45^o$, for the sake of simplicity. (a) presents the input DEM, (b) shows the skwDEM used in this work; (c) and (d) introduce two possible ways of compacting the data.

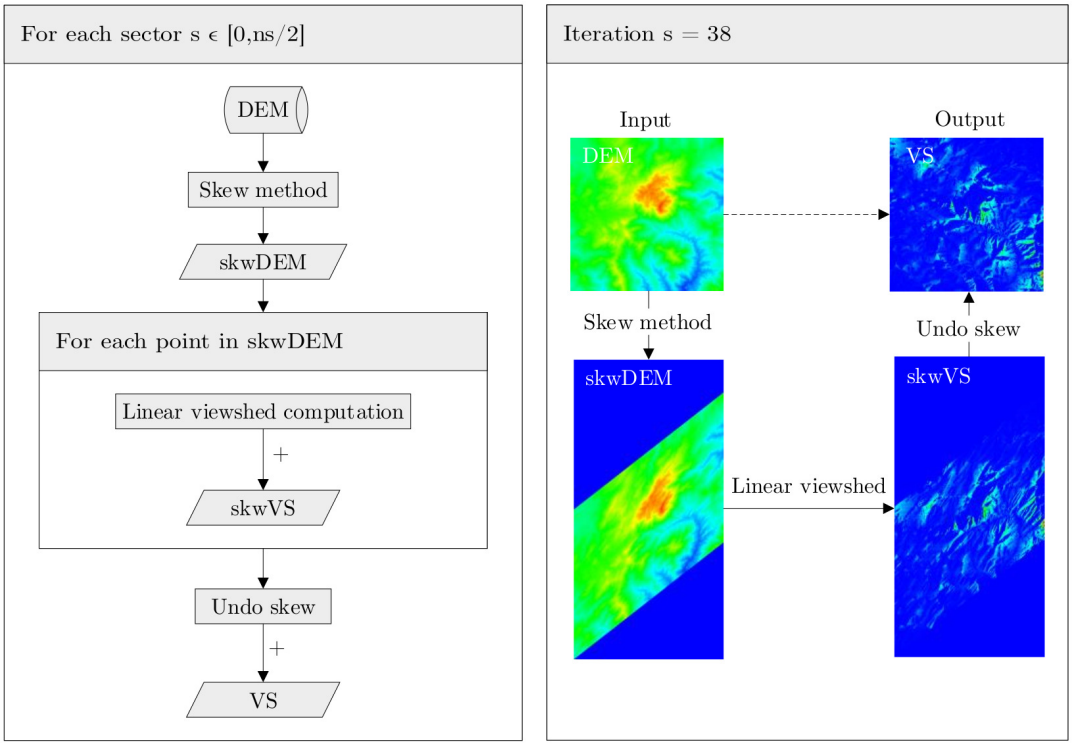
to reduce the conditional structures to the maximum, hence avoiding the well-known thread divergence penalty.

Figure 4 shows the different possible redistributions of rows and columns using the DEM of the * Natural Park (*, Spain), complementing Figure 3. The data of the same latitudes are stored contiguously in memory in the original DEM (Figure 4(a)); that is, the external loop runs from north to south, whereas the internal loop runs from west to east. In Figure 4(b), the skewed model (skwDEM) is graphically represented. Using the interpolation method, all parallel segments from Figure 4(a) were projected into Figure 4(b) so that the size of both structures matches (the number of non-null elements). In this reorganized dataset, unlike the original, all the points in a given sector are placed in the same row and, therefore, memory accesses are sequentially performed increasing locality.

The reorganized matrix shown in Figure 4(b) could later be compacted by: aligning all data to the left of the structure (Figure 4(c)), or relocating the data within the upper light color triangle to the lower right area of the structure, thus forming a *dimy* x *dimx* square structure (Figure 4(d)). This last method aims to further compact the information to make memory access as regular as possible but increasing the complexity and hence, the building time. Although both approaches seem to fit better for GPU processing in theory, they have not revealed significant differences in practice. Therefore, only the simplest and fastest approach shown in Figure 4(b) for the skwDEM structure is used in all implementations of the sDEM algorithm.

Given the above, our sDEM methodology can be described as follows (Figure 5):

(1) For each sector $s \in [0, ns/2]$, do:
  (a) Create the $2 \cdot dimy$ x $dimx$ *skwDEM*, which is unique to each sector, from the $dimy$ x $dimx$ *DEM* and s.
  (b) Calculate the horizontal (A,D) and vertical (B,C) limits of the *skwDEM* structure, which depend on the sector s.
  (c) Let $POV_{i,j}$ be the chosen point of view with i and j coordinates. For each point $POV_{i,j} \in skwDEM$ with $i \in [A, D]$ and $j \in [B, C]$, do:
    (i) Compute the linear viewshed considering sectors s and $s + 180^o$, i.e.,



(a) Main loop (b) Single iteration

Figure 5. Flowchart of our sDEM proposal for total viewshed computation showing (a) the steps inside the main loop which runs through all the sectors and (b) the outputs obtained in a single iteration of the same loop. In the last, null and low values are represented in blue; whereas red cells represent maximum values.

analyze to the right and to the left the points in the row to which $POV_{i,j}$ belongs in the *skwDEM*.

(ii) Accumulate the viewshed result in *skwVS*.

(d) Transform *skwVS* into *VS* by undoing the operations performed in Step 1a (this procedure includes Pappus's theorem and also corrects the deformation introduced by the *skwDEM*). Accumulate the results in *VS*.

The viewshed computation of all sectors is an embarrassingly parallel task because the computation is independent of each other. This reduces the total viewshed problem to calculating $ns/2$ times the singular viewshed problem for every point in the $skwDEM$. Also, by skewing before carrying out the computation sDEM achieves that all the points placed in the same row in the $skwDEM$ constitute a common static BoS, for a given sector, that must be reconstructed only $ns/2$ times. On the contrary, the BoS used in (Tabik *et al.* 2014) must be reconstructed for each point and sector.

4.2. *Multi-GPU implementation*

Since this particular problem is similar to matrix processing, our proposal to accelerate the calculation of the total viewshed focuses on exploiting the intrinsic parallelism of this procedure through the use of GPU processing units. In practice, the $ns/2$ sectors are distributed among all available devices so that each one is in charge of processing a similar number of given sectors, which will depend on the chosen scheduling. Every device sequentially launches three kernels, which will be further described, in order to

process the viewshed of all points in the DEM for the corresponding sectors. In this way, each device contains partial viewshed results which are added up by the host in a final stage to obtain the total viewshed. Our method is used to avoid dependencies between threads while performing the viewshed computation.

We will denote block and thread identification numbers as b_{id} and t_{id}, respectively, and thread block dimension as b_{dim}.

4.2.1. Kernel-1: obtaining the skwDEM structure

This kernel is in charge of transforming the original DEM into the skwDEM structure for a given sector (Algorithm 5). In this new model and for the chosen direction, points located consecutively in the terrain are also stored sequentially in memory, which improves the performance of the memory accesses. This is achieved by using the interpolation based on Bresenham's algorithm to soften the projection of the points. Every thread is in charge of interpolating the corresponding point according to its 2D thread identification number defined by i and j variables.

Regarding the implementation, this kernel is launched using $C_{by} = dimy/8$ and $C_{bx} = dimx/8$ 2D threads blocks with 8 threads per block, so as not to exceed the maximum register file size shared between thread blocks, avoiding schedule problems.

Algorithm 5 The kernel in charge of generating the $skwDEM$ structure from the DEM given the sector s

int $i = b_{idy} \cdot b_{dim} + t_{idy}$
int $j = b_{idx} \cdot b_{dim} + t_{idx}$
float $y = tan(s) \cdot j$
int $dest = y$
float $r = y - dest$
int $p = dimy + i - dest$
if $i < dimy$ & $j < dimx$ **then**
  $skwDEM[p \cdot dimy][j] \mathrel{+}= (1 - r) \cdot DEM[dimy \cdot i][j]$
  $skwDEM[p \cdot dimy - 1][j] \mathrel{+}= r \cdot DEM[dimy \cdot i][j]$
end if

4.2.2. Kernel-2: viewshed computation on the skwDEM

This kernel computes the viewshed for a given sector and every point located in the $skwDEM$, obtaining as a result the $skwVS$ matrix. The pseudo-codes of this kernel are shown in Algorithms 6 and 7, where each thread manages a particular point $POV_t \in skwDEM, t = \{i, j\}$ considering t as the corresponding two dimensional

Algorithm 6 The kernel in charge of the $skwVS$ computation on the skwDEM

int $i = b_{idy} \cdot b_{dim} + t_{idy}$
int $j = b_{idx} \cdot b_{dim} + t_{idx}$
float $r = (1.0/cos(s))^2$
float $cv = 0$
if $i < 2 \cdot dimy$ & $j < dimx$ **then**
  float $h \mathrel{+}= sDEM[i][j]$
  $cv \mathrel{+}= linearViewshed(i, j, h, skwDEM, true)$
  $cv \mathrel{+}= linearViewshed(i, j, h, skwDEM, false)$
  $skwVS[i][j] = cv \cdot r$
end if

Algorithm 7 $linearViewshed(i, j, h, skwDEM, forward)$

int $k = j$
int $dir = 0$
if $(forward)$ **then** $dir = 1$ **otherwise** $dir = -1$
bool $visible, above, opening, closing$
float $max\theta = -\infty$
while $k \in nzSet$ **do**
  $\Delta d = |k - j|$
  $\theta = (skwDEM[i][k] - h)/\Delta d$
  if $(\theta > max\theta)$ **then** $above = 1$
  $opening = above$ & $!visible$
  $closing = !above$ & $visible$
  $visible = above$
  $max\theta = max(\theta, max\theta)$
  if $(opening)$ **then** $open\Delta d = \Delta d$
  if $(closing)$ **then** $cv \mathrel{+}= \Delta d \cdot \Delta d - open\Delta d \cdot open\Delta d$
  $k \mathrel{+}= dir$
end while

thread. The variable h contains both the observer and location heights in this case. Every thread obtains its computation range of non-zero values within the corresponding row (contained in the $nzSet$ structure) from the $skwDEM$ matrix. Then, every thread computes its visibility forward and backward across the row to which it belongs. The resulting viewshed value is thereafter stored in its corresponding position of the $skwVS$ matrix.

This kernel is launched with $2 \cdot C_{by}$ and C_{bx} 2D threads blocks using twice as many thread blocks as in the y-dimension according to the size of the $skwDEM$ matrix.

4.2.3. Kernel-3: obtaining the final viewshed on the DEM

Once the viewshed is computed on the $skwDEM$ for every POV and stored in the $skwVS$ matrix, this kernel transforms the latter structure by undoing the rotation performed in Kernel-1 to obtain the final viewshed VS matrix on the original DEM. The pseudo-code in charge of performing this procedure is presented in Algorithm 8. This kernel is also launched with the same configuration as Kernel-1.

Algorithm 8 The kernel in charge of transforming the $skwVS$ on the skwDEM to the VS structure on the original DEM

int $i = b_{idy} \cdot b_{dim} + t_{idy}$
int $j = b_{idx} \cdot b_{dim} + t_{idx}$
float $y = tan(s) \cdot j$
int $dest = y$
float $r = y - dest$
int $p = dimy + i - dest$
if $i < dimy$ & $j < dimx$ **then**
  float $skwVS_a = skwVS[p \cdot dimy][j]$
  float $skwVS_b = skwVS[(p-1) \cdot dimy][j]$
  $VS[i][j] \mathrel{+}= (1 - r) \cdot skwVS_a + r \cdot skwVS_b$
end if

Algorithm 9 Host code in charge of scheduling the work for the different devices

```
for d = 0, nd do
    dev_d ← Allocate(|DEM|, |skwDEM|, |skwVS|, |VS|)
end for
for d = 0, nd do
    dev_d ← MemcpyAsyncH2D(DEM)
end for
for s = 0, ns/2 do
    int d = s % nd
    dev_d ← Kernel − 1, 2, 3
    dev_d ← MemcpyAsyncD2H(VS_d)
end for
for d = 0, nd parallel do
    VS += VS_d
end for
```

4.2.4. Scheduling multi-GPU processing on the host

In order to perform the total viewshed calculation in a multi-GPU system, several steps must be performed as shown in Algorithm 9. First, we must reserve the required memory spaces to allocate the different matrices in all the available devices. Then, the original DEM can be transferred from the host to each device (dev) of the total available devices (nd). The target number of sectors $ns/2$ will be distributed among the different devices so that the work load is balanced. Each device will execute the three above-mentioned kernels accumulating the result of the viewshed computation, considering all the points and every target sector, in their private VS_d structure. Finally, these matrices are transferred from the devices to the host so that a final parallel reduction can be performed, obtaining the total viewshed VS result.

5. Experiments

This section assesses the performance of our sDEM proposal with respect to well-known GIS software and the state-of-the-art. Section 5.1 explains the experimental setup. Section 5.2 presents the comparison between sDEM and GIS software in addressing the multiple viewshed problem. Section 5.3 evaluates the computational performance of sDEM compared to the state-of-the-art algorithm using the total viewshed problem as a case study in three scenarios: (i) sector viewshed computation for a random direction, (ii) average sector viewshed computation, and (iii) total viewshed map generation.

5.1. *Experimental setup*

We select two operating systems (OSs) for the experiments according to their requirements:

- Windows OS: Windows 10 with an Intel(R) Core(TM) i5-6500 CPU @3.20GHz with 4 cores (4 threads) and 8GB DDR4 RAM.
- Linux OS: Ubuntu 16.04.5 LTS with an Intel(R) Xeon(R) E5-2698 v3 @2.30GHz with 16 cores (32 threads) and 256GB DDR4 RAM, along with four GTX 980 Maxwell GPUs with 2048 CUDA cores, 16 SMs, 1.12GHz, and 4GB GDDR5 each one.

Table 1. Different DEMs used in the experiments.

		UTM		
Dataset	Dimension	Zone[a]	Easting	Northing
DEM10m-2k	2000x2000	30S	0310000mE	4070000mN
DEM10m-4k	4000x4000	30S	0360000mE	4100000mN
DEM10m-8k	8000x8000	30S	0360000mE	4140000mN

[a]Latitude band designator.

The experiment presented in Section 5.2 is executed on Windows OS because GIS software is usually developed for this specific operating system; whereas the experiment described in Section 5.3 is executed on Linux OS to obtain an optimal measurement of the computational performance. These experiments were designed to be as representative as possible of a real problem where obtaining the visibility in a particular direction or region is necessary. This is the reason why three DEMs of the * Natural Park with 10 meters of resolution and different dimensions, in relation to the number of points in vertical and horizontal directions, were considered (Table 1). The calculation of the total viewshed is not only limited to this area of interest, but includes the surrounding area beyond it which is very geographically diverse. Observers are considered to be located 1.5 meters above the ground.

Regarding the implementation, the OpenMP API is used to enable the multi-threaded execution of every selected sector with dynamic scheduling since it has proved to obtain the best performance. Host codes are compiled using the g++ 5.4 open-source compiler with ffast-math, fopenmp, and O2 optimization flags. CUDA files make use of the NVIDIA NVCC compiler from the CUDA compilation tools V10.0.130. The multi-core implementation of our proposal is launched with the maximum number of threads available in the system, the same way that single-GPU and multi-GPUs implementations are configured to operate the devices at full capacity.

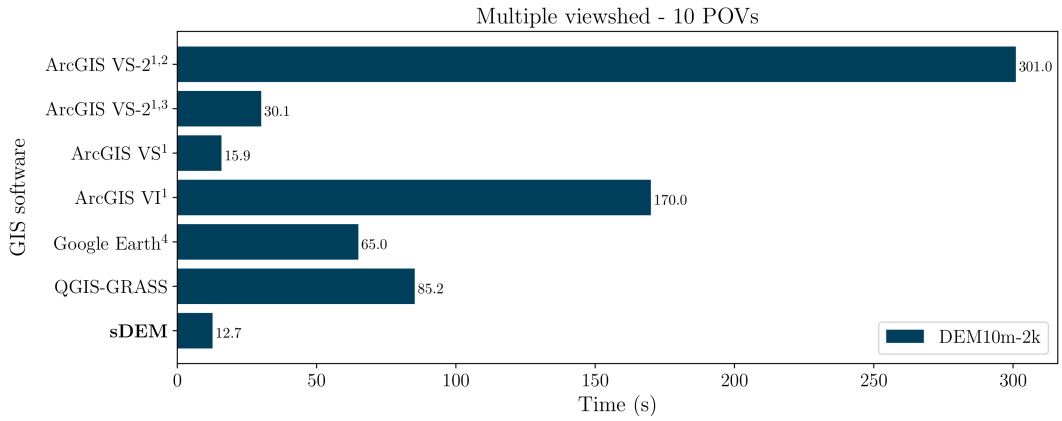


Figure 6. Computational performance comparison between the sDEM proposal and the most used GIS software in solving the multiple viewshed problem considering 10 POVs randomly located in the DEM10m-2k. Single-thread execution is considered to obtain the run-time of those programs not otherwise indicated. VS, VS-2, and VI stand for the first and second version of the Viewshed tool and the Visibility one, respectively, inside the Spatial Analyst extension of ArcGIS. QGIS-GRASS uses the r.viewshed module. [1]multi-thread execution was required using the maximum number of cores available. [2]using PERIMETER_SIGHTLINES parameter. [3]using ALL_SIGHTLINES parameter. [4]Google Earth does not have multiple/total viewshed computation capability so the average time in computing singular viewshed has been multiply by the number of POVs considered.

The GIS software used for comparison includes ArcGIS 10.7, specifically the Spatial Analyst extension which contains the Viewshed 2 (VS-2), Viewshed (VS), and Visibility (VI) tools. Google Earth Pro 7.3 and QGIS-GRASS 3.10.2 are also used.

5.2. *Comparison with GIS software*

A fair comparison in the context of this work would be comparing the total viewshed computation using our approach and other GIS software/tools. However, as there does not exist any public software/tool to compute total viewshed, we will compare the results only for few points. This experiment assesses the computational performance of sDEM and the most used GIS software in solving the multiple viewshed problem. In particular, the object is to compute the accumulated viewshed considering 10 POVs randomly located in the DEM10m-2k. Single-threaded implementations were used for our sDEM proposal, QGIS-GRASS, and Google Earth; whereas ArcGIS has to be executed using all available cores. Moreover, the execution time obtained using Google Earth results from extrapolating the singular viewshed computation to the current case of 10 POVs since this software does not support this operation. Figure 6 shows the time each software requires to complete the multiple viewshed task. Our sDEM proposal outperforms every analyzed GIS software: ArcGIS VS-2 with two different configurations (23.7x, 2.4x), ArcGIS VS (1.3x), ArcGIS VI (13.4x), Google Earth (5.1x), and QGIS-GRASS (6.7x). Although sDEM achieves significant results in this multiple viewshed computation, the greatest gain is given in the total viewshed computation discussed below.

5.3. *Total viewshed analysis*

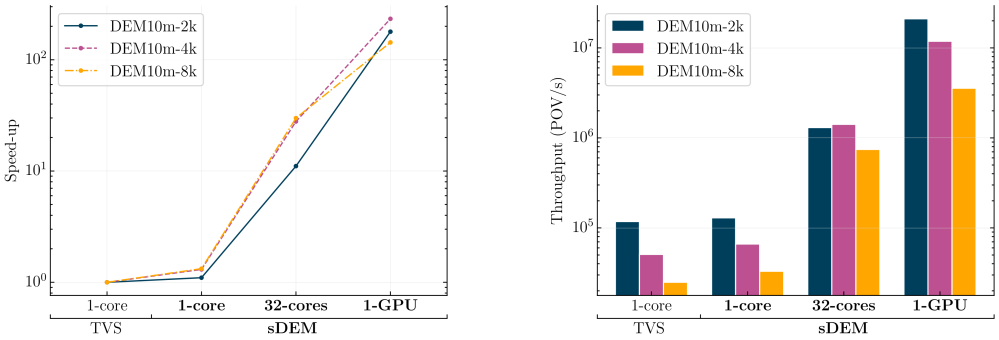
To the best of our knowledge, our sDEM proposal and the TVS algorithm (Tabik *et al.* 2014) are the only approaches in the literature capable of performing the total viewshed computation on entire datasets without carrying out prior reductions in workload. In order to achieve a fair analysis, a size of $dimx$ has been chosen for the BoS in the case of the TVS algorithm so that this structure coincides with the number of points processed per row in the sDEM algorithm. Thus, the workload is similar for both TVS and sDEM algorithms making it possible to perform a computational performance comparison using speed-up and throughput values. We implement single-threaded, multi-threaded, single-GPU, and multi-GPU versions of our sDEM proposal and compared them to the single-thread implementation of the TVS algorithm. Three experiments are set to assess the performance in the total viewshed computation.

5.3.1. Sector viewshed considering a random sector

The computational performance of our sDEM proposal is analyzed and compared with the TVS algorithm in computing total viewshed considering a single random sector, where the sector 10^o is selected. Figure 7 presents the acceleration curves and the throughput results (POVs processed per second) using three DEMs. Best-studied cases show that our proposal outperforms the TVS algorithm, achieving a maximum acceleration up to 232.8x using the 1-GPU implementation on DEM10m-4k. Throughput results show that this variable increases about 177.7x for the same implementation on DEM10m-2k. Multi-GPU implementations are not considered due to the low workload when distributing one single sector among more than one device.

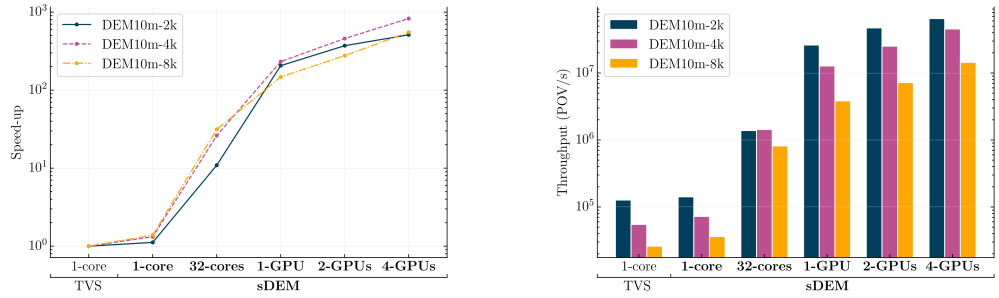
5.3.2. Sector viewshed based on average values

In this experiment, unlike the prior one, the direction range is selected from 0^o up to 45^o to obtain average values per sector. This choice of design lies in the fact that single-threaded executions of TVS and sDEM required several weeks to complete when using a higher range. Besides, results within this range are representative and can be extrapolated to any target range. Figure 8 introduces the acceleration curves and the throughput results achieved. Best-studied cases show that the maximum speed-up result achieved is up to 827.3x with the 4-GPUs implementation with respect to the TVS algorithm considering DEM10m-4k. Throughput results show that this variable increases about 511.1x for the same implementation on DEM10m-2k.



(a) Speed-up curves (b) Diagram of throughputs

Figure 7. Speed-up curves and throughput diagrams for the state-of-the-art total viewshed algorithm, TVS (Tabik *et al.* 2014) and the different implementations of our sDEM proposal using single-core, multi-core, single-GPU, and multi-GPU platforms to compute singular viewshed for a randomly selected sector, $s = 10^o$ (BoS size of *dimx* points for the TVS algorithm). Every color is related to a particular dataset. Logarithmic scale is used.



(a) Speed-up curves (b) Diagram of throughputs

Figure 8. Speed-up curves and throughput diagrams for the state-of-the-art total viewshed algorithm, TVS (Tabik *et al.* 2014) and the different implementations of our sDEM proposal using single-core, multi-core, single-GPU, and multi-GPU platforms to compute sector viewshed. Directions fulfilling $0^o < s < 45^o$ are considered to obtain average values per sector (BoS size of *dimx* points for the TVS algorithm). Every color is related to a particular dataset. Logarithmic scale is used.

5.3.3. Total viewshed map generation

The final outcome from computing our proposed sDEM algorithm to obtain the total viewshed map of the * Natural Park (*, Spain) is presented in Figure 9. No substantial differences have been found after analyzing the values of absolute and relative differences when comparing the total viewshed results from the TVS and sDEM algorithms. The DEM10m-2k was used for this analysis, where the absolute difference found is up to 1.18%, whereas the relative difference is up to 4.21%. All these values are fully within the limits recommended in this field (Tabik *et al.* 2013).

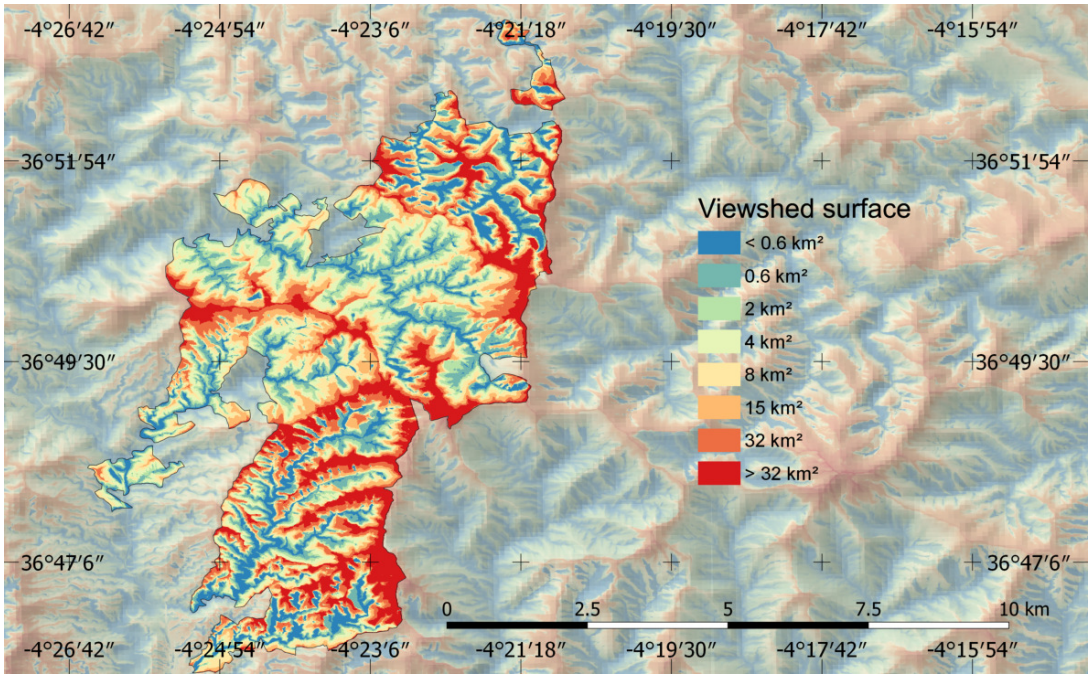


Figure 9. Total viewshed map of the * Natural Park and its surroundings in the province of *, Spain.

6. Conclusions

In this paper, we present a new methodology called skewed Digital Elevation Model (sDEM) to speed-up terrain surface analysis. Total viewshed computation was selected as a case study to assess the performance of this new methodology, which is designed from scratch and differs from state-of-the-art methods in the way that operations are performed. It focuses on increasing the performance of memory accesses by applying a data restructuring before starting the computation. The proposed data reorganization opens the door for intensive use of GPUs in many algorithms for which it had never been considered due to their irregularity and low efficiency.

Different versions of our algorithm have been proposed for single-core, multi-core, single-GPU and multi-GPU platforms, along with intensive performance studies compared to the literature. sDEM has been tested on Windows and Linux operating systems using two different systems and three DEMs of up to 64 millions of points from the * Natural Park (*, Spain). Our implementations have proved to perform better than the most used GIS software regarding the multiple viewshed computation. In fact, this difference would be much greater when considering all the points in the terrain but

current GIS software is unable to carry out this computation. Moreover, sDEM largely outperforms the state-of-the-art algorithm in terms of speed-up and throughput for the three evaluated DEMs. In particular, our approach accelerates this computation up to 827.3x for the best studied case with respect to the baseline single-threaded implementation on a given DEM formed by 16 million POVs.

Our algorithm can be used for analyzing the surface of any terrain. For example, the computation of the visibility map of any terrain is faster with sDEM than with the approaches reported in the literature. Also, the analysis of other topographic features such as slope and elevation could be improved by applying our methodology. Once this paper is accepted, we will release a cross-platform plug-in so that the international scientific community can reproduce our experiments with the sDEM algorithm and use it with their own DEMs to obtain total viewshed maps.

Data and codes availability statement

The data and codes of the sDEM algorithm that support the findings of this study are available in the 'figshare.com' repository with the identifier 'https://figshare.com/s/010bfe676260d3bd41b4'.

Funding

*

References

Atallah, M.J., 1983. Dynamic computational geometry. *In*: *Foundations of Computer Science, 1983., 24th Annual Symposium on*. IEEE, 92–99.

Brughmans, T., van Garderen, M., and Gillings, M., 2018. Introducing visual neighbourhood configurations for total viewsheds. *Journal of Archaeological Science*, 96, 14–25.

Cabral, B., Max, N., and Springmeyer, R., 1987. Bidirectional reflection functions from surface bump maps. *In*: *ACM SIGGRAPH Computer Graphics*. ACM, 273–281.

Cauchi-Saunders, A.J. and Lewis, I.J., 2015. Gpu enabled xdraw viewshed analysis. *Journal of Parallel and Distributed Computing*, 84, 87–93.

Cervilla, A., Tabik, S., and Romero, L., 2015. Siting multiple observers for maximum coverage: An accurate approach. *In*: *Proceedings of the 2015 International Conference on Computational Science, ICCS2015*.

Choset, H.M., *et al.*, 2005. *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Dou, W., Li, Y., and Wang, Y., 2018. A fine-granularity scheduling algorithm for parallel xdraw viewshed analysis. *Earth Science Informatics*, 11 (3), 433–447.

Dou, W., Li, Y., and Wang, Y., 2019. An equal-area triangulated partition method for parallel xdraw viewshed analysis. *Concurrency and Computation: Practice and Experience*, 31 (17), e5216.

Dungan, K.A., *et al.*, 2018. A total viewshed approach to local visibility in the chaco world. *antiquity*, 92 (364), 905–921.

ESRI, 2010. *Arcgis software. version 9.3*. Environmental Systems Research Institute.

Fisher, P.F., 1992. First experiments in viewshed uncertainty: simulating fuzzy viewsheds. *Photogrammetric engineering and remote sensing*, 58, 345–345.

Floriani, L.D. and Magillo, P., 1994. Visibility algorithms on triangulated digital terrain models. *International Journal of Geographical Information Systems*, 8 (1), 13–41.

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*. Taylor & Francis Edinburgh, vol. 2, 751–770.

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

Gao, Y., *et al.*, 2011. Optimization for viewshed analysis on gpu. *In*: *2011 19th International Conference on Geoinformatics*. IEEE, 1–5.

Kaučič, B. and Zalik, B., 2002. Comparison of viewshed algorithms on regular spaced points. *In*: *Proceedings of the 18th spring conference on Computer graphics*. ACM, 177–183.

Li, J., Zheng, C., and Hu, X., 2010. An effective method for complete visual coverage path planning. *In*: *2010 Third International Joint Conference on Computational Science and Optimization*. IEEE, vol. 1, 497–500.

Neteler, M., *et al.*, 2012. GRASS GIS: a multi-purpose Open Source GIS. *Environmental Modelling & Software*, 31, 124–130.

Osterman, A., Benedičič, L., and Ritoša, P., 2014. An io-efficient parallel implementation of an r2 viewshed algorithm for large terrain maps on a cuda gpu. *International Journal of Geographical Information Science*, 28 (11), 2304–2327.

Qarah, F.F. and Tu, Y.C., 2019. A fast exact viewshed algorithm on gpu. *In*: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 3397–3405.

Song, X.D., *et al.*, 2016. Parallel viewshed analysis on a pc cluster system using triple-based irregular partition scheme. *Earth Science Informatics*, 9 (4), 511–523.

Stewart, A.J., 1998. Fast horizon computation at all points of a terrain with visibility and shading applications. *Visualization and Computer Graphics, IEEE Transactions on*, 4 (1), 82–93.

Stojanović, N. and Stojanović, D., 2013. Performance improvement of viewshed analysis using gpu. *In*: *2013 11th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*. IEEE, vol. 2, 397–400.

Strnad, D., 2011. Parallel terrain visibility calculation on the graphics processing unit. *Concurrency and Computation: Practice and Experience*, 23 (18), 2452–2462.

Tabik, S., *et al.*, 2014. Efficient data structure and highly scalable algorithm for total-viewshed computation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8 (1), 304–310.

Tabik, S., Zapata, E.L., and Romero, L.F., 2013. Simultaneous computation of total viewshed on large high resolution grids. *International Journal of Geographical Information Science*, 27 (4), 804–814.

Wang, Y. and Dou, W., 2019. A fast candidate viewpoints filtering algorithm for multiple viewshed site planning. *International Journal of Geographical Information Science*, 1–16.

Zhao, Y., Padmanabhan, A., and Wang, S., 2013. A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. *International Journal of Geographical Information Science*, 27 (2), 363–384.