High-performance three-horizon composition algorithm for large-scale terrains

Siham Tabik*, Luis Felipe Romero and Emilio López Zapata

Department of Computer Architecture, University of Málaga, Campus de Teatinos, Málaga, Spain

(*Received 18 Februay 2009; final version received 21 June 2009*)

This work presents a high-performance algorithm to compute the horizon in very large high-resolution DEMs. We used Stewart's algorithm as the core of our implementation and considered that the horizon has three components: the ground, near, and far horizons. To eliminate the edge-effect, we introduced a multi-resolution halo method. Moreover, we used a new data partition approach, to substantially increase the parallelism in the algorithm. In addition, several optimizations have been applied to considerably reduce the number of arithmetical operations in the core of the algorithm. The experimental results have demonstrated that by applying the above-described contributions, the proposed algorithm is more than twice faster than Stewart's algorithm while maintaining the same accuracy.

Keywords: three-horizon composition; multi-resolution halo; four-overlapping grid tiling partition; parallel computing; multi-core nodes

1. Introduction

In the last decade, new larger Digital Elevation Models (DEMs) of higher resolution are being created. For example, a DEM of the entire world of size 15×10^{15} points and precision 3 arc seconds ($\sim 90 \times 90$ m^2 at the equator) is now available in SRTM (Shuttle Radar Topography Mission) Database (CGIAR-consortium for spatial information 2007), in addition to a large number of DEMs of many regions of the world with resolution greater than 1 arc second (30×30 m^2). Moreover, in the near future, new global products of higher resolution are expected, as is the case for the DEM of the United States of resolution 10×10 m^2. This fact is generating a huge need for new efficient high-performance algorithms.

Knowledge of the visible parts of a terrain from a given point is important in many fields, in shading and visibility applications, in the development of geographic information systems, and particularly in solar irradiation models. For example, the computation of the horizon at all the points of a terrain is essential for an accurate calculation of direct and diffuse irradiation in all advanced solar radiation models (Dubayah, and Rich 1995, Fu and Rich 2007, Romero *et al.* 2008). In the case of ESRA (European Solar Radiation Atlas) models (Schamer and Greif 2000), the principal mathematical reference for several solar irradiation softwares (Marcel and Jaroslav 2004, Romero *et al.* 2008), the horizon computation is the most costly part of these models even for small areas (Romero *et al.* 2008). Moreover, running these

*Corresponding author. Email: stabik@uma.es

algorithms on single processor computers is practically impossible as the DEM becomes larger, on the order of millions of points, as the ones available in SRTM Database (2007). Only a parallel horizon algorithm could face the significant computational demands involved.

There exist several algorithms that simplify the horizon calculation, but only for one single processor. Atallah (1983) used a divide-and-conquer algorithm to compute the upper envelope of N segments in the plane with a time complexity of $O(N\alpha(N)logN))$, where α is the inverse Ackermann function, a slow-growing function, nearly constant. This algorithm recursively divides the line segments into halves, and pairwise merges the resultant upper envelopes through a sweep line technique. Hershberger (1989) improves Atallah's algorithm, reaching an optimal complexity of $O(NlogN)$. De Floriani and Magillo (1994, 1995) use a probabilistic algorithm that computes the horizon for a single point at different resolutions on triangulated terrains in time $O(NlogN)$. The approximative method of Cabral *et al.* (1987) divides the horizon into S sectors, and in each sector it determines the elevation of the horizon, considering solely the points of the terrain that are in the central line of the sector with a computational cost $O(S(N^{1.5}))$. Finally, Stewart (1998) proposed a more precise algorithm that computes the horizon in the whole sector with a total cost $O(SN(\log^2 N + S))$ and space $O(N(\log N))$. The last algorithm is faster and more precise than the former ones, but it can only be applied to small terrains.

This work aims to develop a faster, more accurate, and parallel algorithm especially suitable for very large high-resolution DEMs. Thanks to its accuracy over the existing serial algorithm, Stewart's method has been used as the core of our implementation. The main contributions of this work have come to improve the accuracy of a preliminary implementation of the horizon algorithm that we used for the irradiation model described in Tabik *et al.* (2007). This preliminary code (1) uses a two-grid partition method, (2) applies a two-level halo to eliminate the edge-effect, (3) accelerates the core of the algorithm by applying several algorithmic optimizations, and (4) includes parallelism but only at the (computational-) node level. The work presented in this article substantially improves all the aspects introduced in the previous version by (1) using a four-grid partition method to allow better accuracy (2) extending the two-level halo to multilevel halo, (3) readapting the arithmetic optimizations to the core of the algorithm, and (4) including parallelism at the sector level too (inside each single node). To summarize, this article makes the following contributions:

- The classification of the horizon into three components: the ground, near, and far horizons. The resulting horizon is computed as the maximum of the three components (Section 4).
- A multilevel halo method, which is critical for high-precision DEMs (less than 1×1 m^2), allows us to include the far-horizon component with a low computational cost (Section 4.1).
- A new four-overlapping grid tiling partition method has been applied to compute the near-end horizon, which usually represents the most costly component. This method not only accelerates the algorithm but also reduces the memory requirements (Section 4.3).
- The application of several algorithmic optimizations substantially reduces the runtime of the core of the algorithm (Section 5).
- The obtained hybrid parallel implementation can run on a large range of parallel architectures, from shared to distributed memory architectures including hybrid platforms such as clusters of multi-core nodes (i.e., multi-core computers interconnected via the network) (Section 6).

- The proposed algorithm has been tested on the DEM of Andalusia, south of Spain, of 52,000 × 32,000 point grid, each point of resolution equaling 10 × 10 m^2 (Section 5). In addition, the evaluation of the computational (Section 7) and numerical results (Section 8) demonstrates that our algorithm is able to provide more accuracy with less computational requirements.

2. Review of Stewart's algorithm

The DEM of a given terrain consists of a set of N points $p_i(i = 0, ..., N)$ of coordinates (x_i, y_i, z_i), where z_i is the height of the point. As illustrated in Figure 1, Stewart's algorithm (1998) divides the space around a point p_i of the DEM into S sectors σ_s of azimuth angle $\frac{2\pi}{S}$ radians each, and computes in each single sector $s\,(s = 1, ..., S)$ the horizon of all the N sample points. Upon completion, the complete horizon is obtained for all points in all the S sectors.

Let a and b be two horizontal vectors that delimit the sector σ_s, consider Figure 1. Stewart's algorithms uses a data structure called Hull Tree to store during the s^{th} iteration all the candidate points. In particular, the algorithm processes the points in a previously determined order in such a way that each point finds its horizon in the Hull Tree and simultaneously it is incorporated as a candidate point for the points that will be processed in future iterations.

In summary, the main stages, *Sort*, *Find horizons*, *Add to Hull*, and *Reorder* of Stewart's algorithm, are applied as follows:

- For each sector s in S
  - *Sort*: The new coordinates of all the points of the DEM are computed in a new coordinate system $(a^{\perp}, b^{\perp}, z)$, where $a^{\perp}$ and $b^{\perp}$ are two vectors perpendicular to a and b, respectively. All the points are then sorted by $a^{\perp}$ and $b^{\perp}$ and the new coordinates j and k are determined for each point.

– For each point $p_{j,k}$ in order of increasing index j :

  - *Find horizons*: The points in the Hull Tree are used to calculate the horizon of $p_{j,k}$. The structure and processing order of the Hull Tree ensure a highly efficient

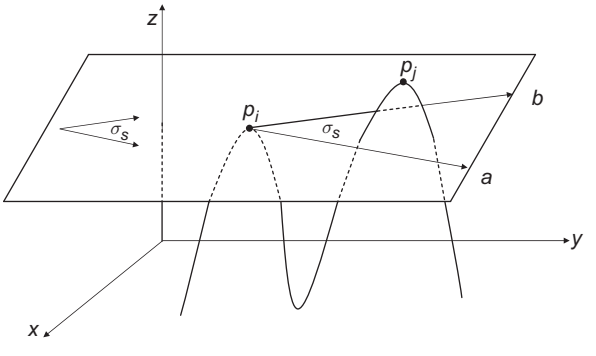


Figure 1. A 3D representation of two points of the terrain, p_i and p_j. Sector σ_s is delimited by the vectors a and b; p_j is visible for p_i in σ_s.

computation of the horizon because the maximum elevation is always found among the $O(logN)$ leaf-to-root nodes in the tree.

- *Add to Hull*: The Hull Tree structure is updated by inserting the heights of the new candidate point $p_{j,k}$. Index k is used to insert the point in the structure.

– *Reorder*: In this stage, the points are reordered from their indices j to recover the original ordering.

3. Parallel computing for Stewart's algorithm

Parallel computing can be introduced in Stewart's algorithm by distributing the sectors among processors, because calculations are independent between sectors. Nevertheless, inside each sector, where the expensive calculation resides, parallelism can hardly be performed due to the strong sequentiality of operations. In addition, for a large DEM, the size of the problem in each sector can still be too high for a single processor, although small parts of the algorithm, such as the *Sort* stage, could be efficiently parallelized.

Nevertheless, there exists an important property of the horizon computation that leads us to consider parallel computing from the next perspective. In general, the horizon calculation for each single point requires only information from the closest surrounding elevations. The distance between the considered point and the closest horizon elevations can vary from a few meters to a few hundred kilometers in an extreme situation depending on the precision of the terrain. Thus, the absence of any relationship between geographically separated regions allows the application of a data decomposition technique based on a block partition of the DEM. That is, each node (e.g., multi-core computer) can compute the horizon for a local area of the DEM. To avoid the undesirable edge-effect due to the lack of information in the boundaries, each node must incorporate additional information from neighboring areas, that is, the points in the outer part of the area assigned to the node. This area of influence, commonly referred to as 'halo,' should consider any elevation that, within the considered margin of error, can belong to the horizon of any point of the local area. To achieve a high efficiency and accurate results, and to avoid redundant calculations, the size and resolution of the area of interest and halos assigned to each node should be carefully dimensioned as explained in the next section.

4. Three-horizon composition method

In this work, we propose a three-composition method that considers the total horizon of a given point as the maximum elevation of three components: (1) an exact ground horizon related to the shadow produced by the tangent plane to the ground at the considered point itself; (2) a high-precision near-end horizon due to the closest points and computed using the full-resolution DEM; and (3) a low-precision far-end horizon that takes into account the elevations placed behind a certain limit (consider Figure 2). This limit depends on the desired precision and the computing system properties as explained in Section 4.1. Only regular DEMs have been considered here; however, the next discussions can be easily generalized to other irregular DEMs.

4.1. Multi-resolution halo technique

The error committed in the horizon computation can be separated into vertical and horizontal components. The vertical error depends only on the accuracy of the elevation data of the

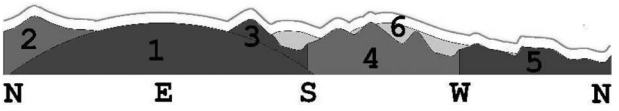


Figure 2. A 2D representation of the total horizon at a given point, represented by the coarse gray line that sums the three horizons: (i) the ground horizon labeled as number 1; (ii) the near-end horizon labeled as numbers 2, 3, 4, and 5 computed in the northeast, southeast, southwest, and northwest, respectively; and (iii) the far-end horizon labeled as number 6.

DEM, whereas the horizontal part strongly depends on the situation of the maximum heights with respect to the observer. Without algorithmic restrictions, the horizontal error in horizon calculations can be expressed as the angle ϵ between the azimuthal position of a maximum elevation in the real terrain and its digitized location. For large distances, this angle becomes very small and can be approximated by one of its upper-bound values, that is, its tangent function. Details about the mathematical formulation of ϵ are presented in Romero and Tabik (2008). For regular DEMs, the root mean square (RMS) of $\tan(\epsilon)$ in a given cell approximates $0.288/d$ and it decreases as the distance d between the observer and the digitized elevation increases; see Figure 3a.

In the field of terrain rendering, most Level-of-Detail (LOD) algorithms use multi-resolution grids to improve the visualization continuity and runtime functionalities. One noteworthy LOD algorithm is provided by Blow (2000). It introduced a hierarchy of nested spheres that envelope all the possible positions of the viewpoint. The size, density, and resolution of each sphere are determined according to the roughness of the area it bounds. This makes the algorithm extremely irregular and difficult to optimize.

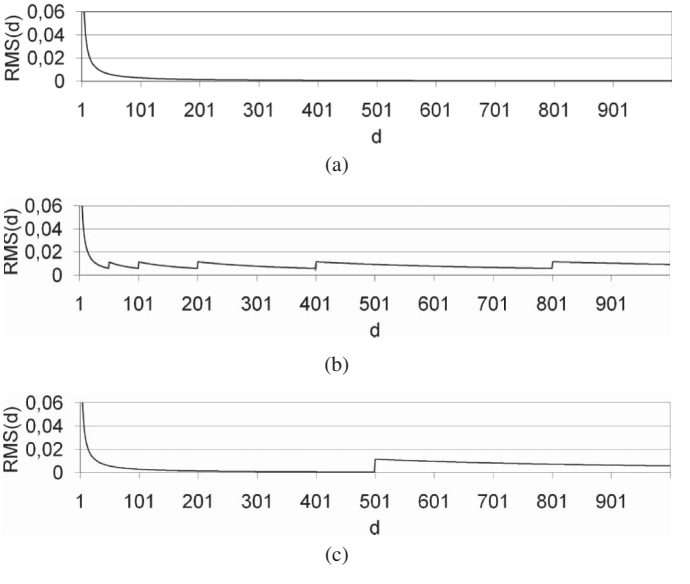


Figure 3. (a) The variation of RMS(d) in terms of the distance d between the observer point and neighboring points. (b) The variation of RMS(d) imposed by Losasso and Hoppe's model (2004). (c) The variation of RMS(d) we adopted for the horizon computation, that is, from the third level on, we coarsen the resolution of the next halo levels without surpassing the threshold error, which is equalling 0.01 in this case.

Losasso and Hoppe (2004) proposed a simpler algorithm. They treated the terrain as a 2D grid and considered a hierarchy of nested squares centered about the inner $n \times n$-points square of interest. The resolution from the inner to the outer levels decreases by a fixed factor 2. To maintain the number of points constant ($= n \times n$) in each one of the multi-resolution levels, the size of each level is also multiplied by that factor. The horizontal error in the area of interest decreases until it reaches a threshold error in the perimeter. When passing to the next coarser level (i.e., the next square), it is multiplied by 2 and then decreases again as shown in Figure 3b. The main limitation of this model is that the size of the area of interest is imposed by the previously fixed threshold error.

We adapt this technique for the horizon computation to distinguish between near and far horizons. In particular, we determine the size of the area of interest (i.e., first level) only in terms of the computer memory hierarchy size, then we keep the same resolution in the first halo (i.e., second level). This allows us to coarsen the resolution of the next level (i.e., third level) by a factor greater than 2, for instance 20, without surpassing the threshold error as shown in Figure 3c. Therefore, we reduce both the number of multi-resolution levels and the complexity of the algorithm.

4.2. Ground horizon

If the curvature of the Earth is not considered, horizontal surfaces, modeled by a point in the DEM, see one half of the sky dome at most, because the ground hides the other half. On inclined surfaces, there is an additional fraction of the sky dome that is not visible from the points in the surface, due to self-hiding, which is commonly called ground shadowing. The ground horizon of a surface, represented by a point p_j, in a sector s can be calculated as follows:

$$h_g(j,s) = \max(0, \alpha_{j,s} - 90^{\circ})$$

where $\alpha_{j,s}$ is the angle between the normal to the surface $\vec{N}_j$ and the search direction $\vec{u}_s$.

Usually, the point of the observer is placed at a certain altitude above the surface. In most cases, this elevation is negligible compared with the DEM precision, but when high-resolution models are considered, the influence of the observer's elevation must be taken into account. As an illustrative example, an observer can see more than 5 km of the sea if his eyes are approximately 1.5 m above sea level, but the visibility will be null if his eyes are exactly at sea level. In solar energy utilization, solar panels are usually placed several meters above the ground, which is of the same order of magnitude as the DEM resolution. In this case, ground shadowing must be reformulated to consider the panel height and inclination, its position with respect to the boundaries of the DEM cells, and several other factors related to the observer's degrees of freedom. The geometrical shape and position of the ground horizon depends on the tilt and head of the considered point and the observer's elevation above the surface.

However, at the scale considered in this work, the study of the horizon should exclude the ground shadowing as a potentially dynamic component (e.g., rotating panels), and consider only horizons that can be classified as static. It should also be mentioned that the ground elevation calculation has a very low computational cost with respect to the total cost.

4.3. Near-end horizon

The selection of an appropriate data partition strategy is crucial for an effective high-performance algorithm. Herein, we introduce a new partition method that not only allows the parallel computation of the near-end horizon but also provides a remarkable reduction of the memory usage.

4.3.1. Four-overlapping grid tiling partition

After partitioning the DEM into small blocks that fit in the computer memory, one could assign a block and its eight surrounding neighbors (consider Figure 4a) to each node and compute the near-end horizon of that block by applying Stewart's algorithm using all the points of the full-resolution halo composed of these eight blocks. This solution implies a large amount of unnecessary calculations and memory usage because too many useless points are included in the Hull Tree to compute the near-end horizon in each single sector (e.g., the points r and t in Figure 4a are used to compute the horizon of the point p in the sector in light gray). Alternatively, a large amount of unnecessary computations can be avoided by assigning only the data involved in the horizon computation of the block of interest, but this implies an irregular partition that will extremely complicate the algorithm. However, an easy reuse of data in memory can be allowed by assigning one sub-grid composed of four neighbor blocks (consider Figure 4b) to each node and computing the near-end horizon of the northeast, northwest, southeast, and southwest blocks in the sectors that belong to the directions southwest, southeast, northwest, and northeast, respectively. For the example used in this work, we fix the size of the blocks n to 1000×1000 points.

To complete the horizon computation in all directions for each single block, we introduce the following partition technique that we call four-overlapping grid tiling partition. We first generate four overlapped grids, the original DEM and three copies of fit shifted by an offset of 1000 points in x, y, and x–y axes, respectively, as shown in Figure 5. We call these grids A, B, C, and D, respectively. Each obtained grid is then partitioned into a number a, b, c, and d of 2000×2000 point sub-grids, $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$ respectively, where $0 \leq i \leq \frac{X_{\mathrm{DEM}}}{2000}$ and $0 \leq j \leq \frac{Y_{\mathrm{DEM}}}{2000}$ (X_{DEM} and Y_{DEM} are the DEM size in x- and y-axes, respectively). The

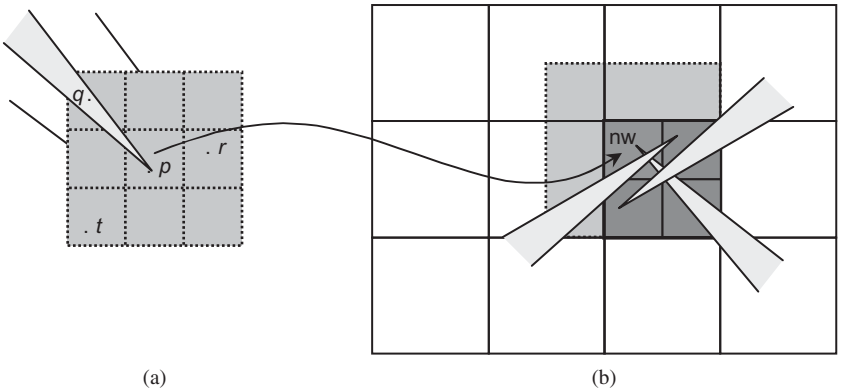


Figure 4. The terrain is partitioned into small blocks of size $n \times n$ points. (a) Case 1: each node is assigned a block (dark gray) together with its full-resolution halo composed of eight blocks (light gray). (b) Case 2: each node is assigned a sub-grid (dark gray) composed of four neighbor blocks, that is, the northeast (ne), northwest (nw), southeast (se), and southwest (sw) blocks; the node then computes the horizons of the ne, nw, se, and sw blocks in the sw, se, nw, and ne quadrants, respectively.

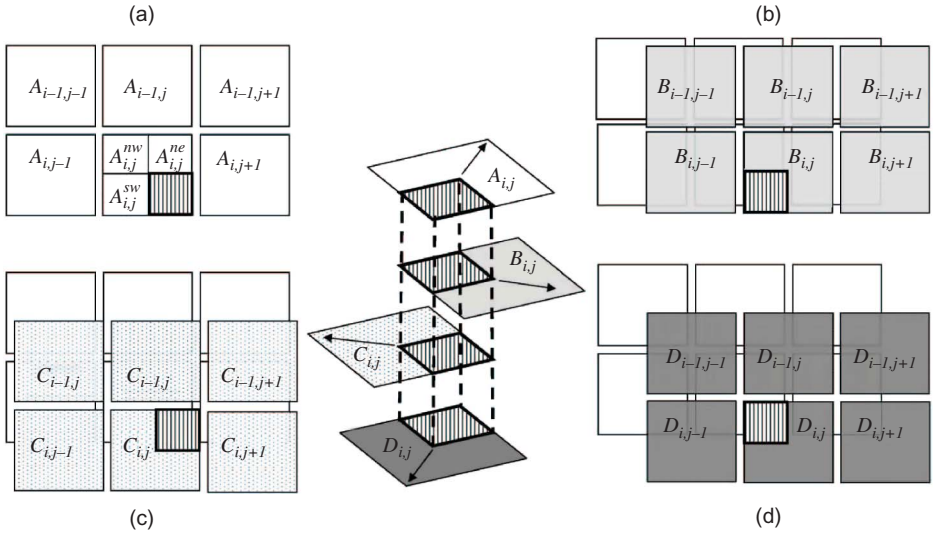


Figure 5. Four overlapped grids are created from the DEM, (a) grid A (white), (b) grid B (light gray), shifted by an offset of n points in x axis, (c) grid C (dotted), shifted by an offset of n points in y axis, and (d) grid D (dark gray), shifted by an offset of n points in both x and y axis. Each grid is then divided into sub-grids: $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$ of size $2n \times 2n$ (with $0 \leq i \leq \frac{X_{\mathrm{DEM}}}{2n}$ and $0 \leq j \leq \frac{Y_{\mathrm{DEM}}}{2n}$, where X_{DEM} and Y_{DEM} are the DEM size in x and y axis respectively) and finally, each single sub-grid is further divided into four blocks, for instance, $A_{i,j}$ is divided into $A^{nw}_{i,j}$, $A^{ne}_{i,j}$, $A^{sw}_{i,j}$, and $A^{se}_{i,j}$. Notice that the block filled by vertical lines belongs simultaneously to the sub-grids: $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$. The arrows show the direction where to compute the horizon in each sub-grid.

resulting sub-grids $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$, of size 2000×2000 points grid, are in fact composed of four blocks, of size 1000×1000 point grid, where,

$$A_{i,j} = \bigcup (A^{nw}_{i,j}, A^{ne}_{i,j}, A^{sw}_{i,j}, A^{se}_{i,j})$$
$$B_{i,j} = \bigcup (B^{nw}_{i,j}, B^{ne}_{i,j}, B^{sw}_{i,j}, B^{se}_{i,j})$$
$$C_{i,j} = \bigcup (C^{nw}_{i,j}, C^{ne}_{i,j}, C^{sw}_{i,j}, C^{se}_{i,j})$$

and

$$D_{i,j} = \bigcup (D^{nw}_{i,j}, D^{ne}_{i,j}, D^{sw}_{i,j}, D^{se}_{i,j})$$

Notice that each 1000×1000 point block simultaneously belongs to four 2000×2000 sub-grids, as shown in Figure 5, that is, $A^{nw}_{i,j} = B^{ne}_{i,j-1} = C^{sw}_{i-1,j} = D^{se}_{i-1,j-1}$, $A^{ne}_{i,j} = B^{nw}_{i,j} = C^{se}_{i-1,j} = D^{sw}_{i-1,j}$, $A^{se}_{i,j} = B^{sw}_{i,j} = C^{ne}_{i,j} = D^{nw}_{ij}$, and $A^{sw}_{i,j} = B^{se}_{i,j-1} = C^{nw}_{i,j} = D^{ne}_{i,j-1}$.

Applying the above-described partition to the example of Andalusia DEM, we obtain $a = 16 \times 26$ of sub-grids $A_{i,j}$, $b = 16 \times 25$ of sub-grids $B_{i,j}$, $c = 15 \times 26$ of sub-grids $C_{i,j}$, and $d = 15 \times 25$ of sub-grids $D_{i,j}$. The horizon of each block is calculated using the appropriate blocks in the sub-grids $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$, respectively. For instance, the horizon of the area of interest filled by vertical lines in Figure 5 is calculated using the blocks $A^{nw}_{i,j}$, $B^{ne}_{i,j}$, $C^{sw}_{i,j}$, and $D^{se}_{i,j}$.

In summary, the resulting parallel algorithm distributes the sub-grids, $A_{i,j}$, $B_{i,j}$, $C_{i,j}$, and $D_{i,j}$ among the available nodes, such as multi-core nodes. Each node then computes, in

parallel by sector, the near-end horizon in exactly four directions, northeast, northwest, southeast, and southwest.

4.3.2. *Space reduction*

The total memory usage of Stewart's algorithm is due to the data structures needed for (in decreasing order of memory usage) (1) constructing the Hull Tree, (2) sorting and reordering all the points of the terrain, (3) computing the horizon in one sector, and (4) storing the DEM. This can be expressed as follows:

$$\text{Memo.Req.(Stewart's Algorithm)} = 50n^2 + 12X^2 \tag{1}$$

where n^2 is the size of the DEM and X^2 is the number of nodes in the Hull Tree. Analytically, $2n^2 \lesssim X^2 \lesssim 6n^2$ for flat terrains and $n^2 \lesssim X^2 \lesssim 2n^2$ for rough terrains; this means that the computational cost is higher for flat terrains. The memory usage of the DEM data structure is only $4n^2$, less than 10% of the total memory usage. To obtain horizons (using Stewart's algorithm) with exactly the same precision as our algorithm on a terrain of million points, we must include a halo of 1000 points. This multiplies the memory requirements, formulated in Equation (1), by at least a factor of 9 (this value increases when the roughness of the terrain decreases). However, for the same terrain, computing the horizon using the four-overlapping grid tiling partition increases the DEM data by only a factor of 16/9 with respect to Stewart's implementation, but reduces the Hull Tree size as well as the size of all the rest of the data structures including the DEM needed during the horizon calculation in a specific sector by a factor of 4/9. In consequence, computing the horizons using the proposed partition method implies less memory usage than Stewart's implementation.

4.4. *Far-end horizon*

Beyond the near-end horizon limits, a coarse grid is used to calculate the far-end heights placed far away from the area of interest. More levels of the far-end horizon could be used depending on the required precision of the data.

The use of low-resolution DEM to compute far-end horizon has two main disadvantages. First, it produces a loss of precision because far-end horizon is computed using information *from* a reduced set of points. Second, far-end horizon is computed only *for* a reduced set of points in the area of interest. This problem can be solved by interpolation techniques or using the information from the closest low-precision data to compute far-end horizon for the rest of the points. In both cases, the induced error depends on the differences in precision and the distance between the considered point and its far-end horizon limits. The difference in precision between near- and far-end limits that achieve the desired accuracy for the overall model can be easily calculated. That is, the precision in far distances, for an area of interest of 1000 × 1000 point grid and a halo of 1000 points width, is lower than 0.000288 radians (about 0.05°). Such high precision is not usually required in typical uses of horizon models because the largest error is always due to the near-end horizon, up to 45° ($= arctan \frac{\Delta x}{\Delta x}$, being Δx, the distance between two neighbor points in regular grids). Therefore, it is reasonable to fix the horizontal far-end horizon error in 1° which corresponds to coarsening the third level of the DEM by a factor of 20.

As the low-resolution representation of Andalusia DEM is too small from a storage point of view, that is, of size 2600 × 1600 point grid and resolution equaling 200 m, it could be computed on a single processor computer. Also, due to Earth's curvature, there are no

horizons behind a distance of 230 km (1150 points), and so neither further recursion nor parallel processing is required in this case. However, if the size of the low-resolution DEM is very large, the same parallel strategy used for computing near-end horizon could be applied. Additional lower resolution halos are only required in very isolated situations.

The core of Stewart's algorithm is applied here in a different way. Each point computes the horizon using only the elevations that are stored in the Hull Tree and placed far away from the near-end limit, because the Hull Tree also includes low-resolution data of the local area and the closest halo that have been already used. These considerations achieve a reduction of 10% in CPU time.

5. Arithmetic optimizations

Dividing Andalusia DEM into a 16 × 26 point grid of size 2000 × 2000 and considering 64 sectors, the horizon computation using Stewart's algorithm takes about 16 days and obviously, no information about the halos is included, whereas considering a halo data of 10 km, that is, 1000 points width, to eliminate the edge-effect, the runtime becomes 10 times larger. Several optimizations can be applied to Stewart's method:

(1) The first optimization consists in reusing the ordering by $b^{\perp}$ in sector i to reorder by $a^{\perp}$ in sector $i + 1$. This reduces the time step 1 (in Stewart's algorithm) to approximately the half. This optimization can be applied to all sectors except to the first one. Note that Stewart's algorithm cannot use this simplification because the coordinates of two points in $(a^{\perp}, b^{\perp}, z)$ can be equal if they are aligned with respect to the sweep line (a or b), and each order will use the opposite criteria for these cases. It can be easily shown that this coincidence cannot occur in a regular grid for a properly selected number of sectors and for an irrational value of the initial angle (measured in radians). Although this coincidence can be true for finite precision arithmetics, in practice, it has been tested that it does not happen for a regular grid of size 2000 × 2000 and for the number of sectors employed in this work. In fact, the alignment of two points is extremely infrequent if angles are not multiples of $\frac{\pi}{6}$ or $\frac{\pi}{4}$. Irregular grids can also benefit from this optimization by carefully selecting an appropriate number of sectors and initial angle.
(2) In a regular grid, there exists an easier way to sort the points by $a^{\perp}$ and $b^{\perp}$ without the need to compute its coordinates in the coordinate reference. The sort in this case can be done by just counting the number of points behind the sweep line using very simple trigonometric operations. As the position number is directly computed, it is not necessary to use any sorting algorithm. In practice, the time spent in the optimized phase 1 is negligible.
(3) The use of the four-overlapping grid tiling method described in Section 4.4 reduces considerably the CPU time of Stewart's algorithm, because the new algorithm performs the second stage, *Find horizons*, for only the quarter part of the sectors; the horizon in the other three parts is obtained from other overlapped grids.

6. Parallel implementation

Taking into account the three-horizon composition, the multi-resolution halo technique, and the four-overlapping grid tiling partition methods described in Section 4, and including the arithmetic optimizations described in Section 5, the implementation of our algorithm could

be summarized as follows. First, the size of the local area and its closest halo are computed in terms of the memory capacity per computing node, then the number of local areas and the limit distance between near and far heights are determined.

The ratio between low-and high-resolution models is then computed in terms of the size of the local area, its closest halos, and the desired precision. The far-end horizon is computed for each point using the low-resolution model. In this case, the computed horizons include only elevations behind the far-end limit. It should be noted that both the core of the proposed algorithm and the original Stewart's algorithm core could be applied recursively, until reaching the required precision. The full high-resolution DEM is partitioned into local areas using the four-overlapping grid tiling method.

Finally, each one of the available nodes starts computing near-end horizons in one of the unprocessed high-resolution sub-grids and then updates the corresponding sectors of the four blocks. At the end of this stage, low-resolution and ground horizons are used to complete the horizon formation of each block.

Works are assigned dynamically to idle nodes using task-farming paradigm (Silva and Buyya 1999) that we have implemented using the message passing interface (MPI) standard (Gropp *et al.* 1999). Inside each node, each MPI process creates as threads as the number of cores in the node using OpenMP standard (OpenMP 2008). Each thread computes the near-end horizon for its local number of sectors on the shared sub-grid. This parallel code has been written in C++ combined with MPI and OpenMp libraries and is available via email from the corresponding author.

7. Analysis of the runtime

Two experiments are described in this section, the first quantifies the improvement in runtime when applying each one of the optimizations, described in Section 5, to the core of Stewart's algorithm. The second consists of a comparison, in terms of runtime, between Stewart's algorithm with and without edge-effect and the proposed algorithm. Both experiments have been carried out on a HP XW4300 Pentium D 960, dual core, of 3600 MHz and 2 GB of RAM.

7.1. Performance improvement using optimizations 1, 2, and 3

Figure 6 shows the mean runtime per sector (in seconds) of Stewart's algorithm without including any optimization (column 1), and applying optimizations 1 (column 2), optimizations 1 and 2 (column 3), and optimizations 1, 2, and 3 (column 4). In fact, column 4 reports the runtime of our algorithm's core. As can be seen from this figure, applying optimization 1 results in a runtime improvement of 16%; applying both optimizations 1 and 2 results in a runtime improvement of 30%; and finally applying the three optimizations results in a runtime improvement of the core of Stewart's algorithm to 50%.

7.2. Stewart versus our algorithm

The differences (i.e., in the used techniques, accuracy of the results and the size of the needed data) between our method and Stewart's method make the comparison in terms of the computational performance very difficult. That is, to obtain results of the same accuracy with both methods, in a 1000×1000 point grid without edge-effect, we must use a grid of 3000×3000 point grid for Stewart's algorithm and four quadrants of 2000×2000 point grid for our algorithm. However, an analysis of the runtime when applying each one of the

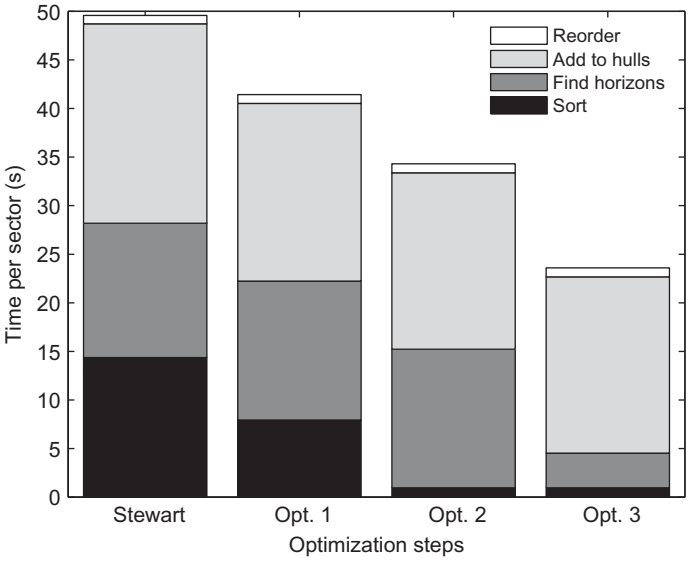


Figure 6. Runtimes of Stewart's algorithm (column 1), Stewart's algorithm including optimization 1 (column 2); optimizations 1 and 2 (column 3); and optimizations 1, 2, and 3 (column 4) for a 2000 × 2000-point grid.

approaches described in this work to Stewart's algorithm is possible and could be worthwhile. Four cases have been considered.

(1) Computing the horizon in an area of size 1000 × 1000 point grid, in 64 sectors, 32 sectors in each processing core, using Stewart's algorithm with edge-effect takes 5.86 minutes.
(2) Computing the full horizon in a 1000 × 1000 point grid, without edge-effect, that is, considering a halo of width 1000 points, using Stewart's algorithm, in 64 sectors, 32 sectors in each processing core, takes 94.10 minutes. A comparison between this time and the time obtained in analysis 1 shows a concordance with $O(N(\log N))$ stated in Section 1.
(3) Applying Stewart's algorithm without edge-effect, including optimizations 1, 2, and 3, to an area of size 1000 × 1000 point grid, with halo data of width 1000, in 64 sectors, 32 sectors in each processing core, takes 44.7 minutes. In this case all the 9,000,000 (3000 × 3000) points are considered as possible horizons and have been included in the Hull Tree; however subroutine *Find horizons* is only applied to the 1000 × 1000 central points (i.e., optimization 3).
(4) Applying Stewart's algorithm without edge-effect, including optimizations 1, 2, and 3, and using the techniques described in Sections 5.2 and 5.3 (i.e., our algorithm), to four quadrants of size 1000 × 1000 takes 13.6 minutes, that is, for each quadrant, the calculations of eight sectors per core take 3.4 minutes.

Comparing analyses 3 and 4, which are the ones that include the same arithmetic optimizations and give identical results with identical accuracy, shows that our algorithm is more than thrice faster than Stewart's algorithm on a Pentium D, dual core.

8. Accuracy of the numerical results

The proposed high-performance algorithm can be used to accelerate several applications from visibility, shading to solar irradiation applications. Two experiments are presented in this section to show the accuracy of our algorithm in both visibility and solar irradiation applications.

8.1. Real versus computed horizons in visibility applications

A comparison of real horizon and computed horizon is shown in Figure 7, where the first figure is a panoramic photography of the horizons from the northeast to southeast at a point of coordinates 36.72294N, 4.356194W in the city of Malaga, south of Spain. We extracted the horizon from this photography by applying Sobel's method for edges detection as displayed in the second image. The third figure represents, for the same terrain, the horizons computed using our algorithm; the far-end horizon shown in this figure is calculated at the closest point to 36.72294N, 4.356194W in the low-resolution DEM (reference ellipsoid WGS84); the horizon line is determined by far- and near-end heights from the left to the right part of the image. It should be noted that the far-end horizon accuracy in this experiment is very high, equaling 0.1°, because the calculated horizon values belong to the interval [0°, 17°], which are considerably small and can fit in one byte per data. Moreover, the size of the matrices is 400 times smaller in the far-end horizon case and more precision can be given for validation purposes. As can be seen from these figures, the calculated and the real horizons are almost identical with an error smaller than 1°, that is, smaller than the desired precision.

Reproducing the horizon shown in Figure 7c using Stewart's algorithm, that is, including the far-end horizon placed at 20 km (represented by the solid line in Figure 7c), needs a DEM of size 5000 × 5000 points, each point of resolution 10 × 10 m^2. This implies a total memory usage of at least 26 GB, which is practically impossible on a single processor computer.

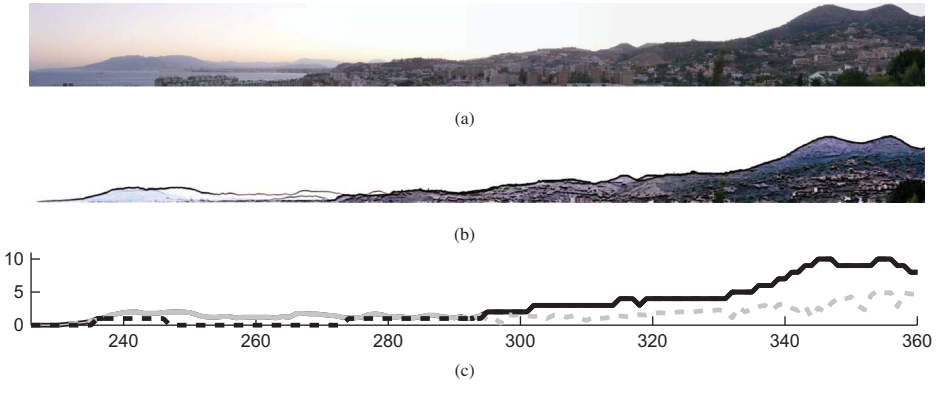


Figure 7. (a) A panoramic photography of the horizon from the northeast to southeast at the point of coordinates: 36.72294N, 4.356194W, in the city of Malaga, south of Spain. (b) This shows the total horizon (the coarse line) extracted from part (a) using Sobel's method for edges detection. (c) This represents the near-end horizon (the black line) and the far-end horizon (the light gray line) calculated by our new algorithm on the same terrain as in part (a).

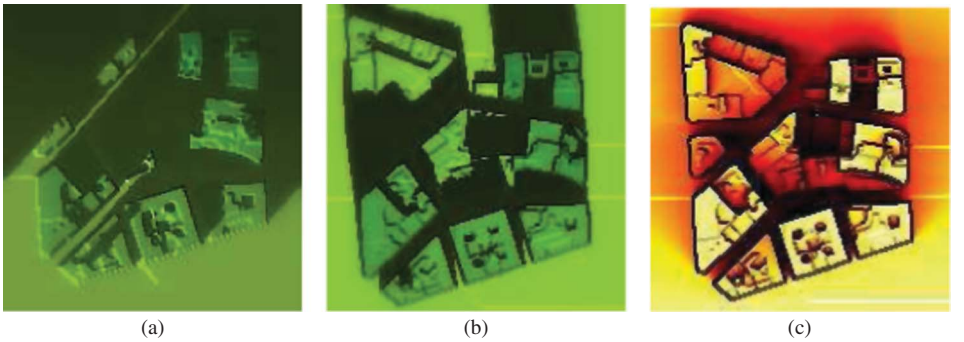
8.2. *Renderings of solar irradiation maps*

The horizons, which need to be computed only once, can be used to determine the global solar irradiation in each single point of the terrain. In this experiment, two low-resolution halos have been used to compute the far-end horizon. For a very accurate rendering, we have used the solar irradiation model developed by Romero *et al.* (2008). To emphasize the shadow effect, we have considered the beam irradiance instead of the global irradiance in the two first experiments. Figure 8a and b shows two renderings of the (instantaneous) beam irradiance (W m^{-2}) on a terrain of 1000 × 1000 points and resolution 1 × 1 m^2, of the city center of Málaga, whereas Figure 8c shows the global solar irradiation (W h m^{-2}) that reaches each point of the considered terrain in one year. The three figures demonstrate a high accuracy on high-resolution DEMs. In particular, Figure 8a shows the important effect of the shadow produced by the mountains that are at 2 km in the west when the sun is at 26° of altitude. The absence of elevated terrains in the southeast of the city center is also shown in Figure 8b. Computing the same horizon with Stewart's implementation requires a very high memory usage, about 20 GB, which is impossible on a single processor computer.

9. Conclusions

This article has proposed a fast, accurate, and parallel horizon algorithm appropriate for large high-resolution DEMs, useful for shading and visibility applications and especially for solar irradiation models. Stewart's algorithm, one of the most accurate horizon algorithms, has been used as the core of our implementation. A three-composition horizon approach along with a multilevel halo have been used to eliminate the edge-effect associated with Stewart's algorithm. In addition, a four-overlapping grid tiling partition has been used to make parallel computing straightforward. The resulting algorithm is substantially more accurate and more than thrice faster than the original Stewart's algorithm. Moreover, our implementation has shown a good scalability on parallel architectures.

The model described in this article represents one of the powerful tools necessary to compute the solar irradiation with a high accuracy, especially for areas with stable climatic



(a) (b) (c)

Figure 8. (a) Rendering of the beam solar irradiance (W.m^{-2}) map on the city center of Malaga, on 27th June at 18:15 hours (sun altitude = 26°). (b) Rendering of the beam solar irradiance (W m^{-2}) map, for the same terrain, on 27 March at 13:06 hours (sun altitude = 55°). Dark and light colors in parts (a) and (b) show low and high values of irradiance, respectively. (c) Rendering of the annual global solar irradiation (W h m^{-2}) map for the same terrain. Dark and light colors in part (c) show low and high values of global solar irradiation, respectively.

conditions. In fact, this tool can considerably improve the accuracy of the solar irradiation model described in Romero *et al.* (2008).

References

Atallah, M., 1983. Dynamic computational geometry. *Proceeding of the 24th IEEE Symposium on the Foundations of Computer science*, 92–99.

Blow, J., 2000. Terrain rendering at high levels of detail. *In*: Game developers' Conference 2000 CMP, 8–12 March, San Jose, CA.

Cabral, B., Max N., and Springmeyer, R., 1987. Bidirectional reflection functions from surface bump maps. *ACM SIGGRAPH Computer Graphics*, 21, 273–281.

CGIAR-Consortium for Spatial Information, 2007. Shuttle Radar Topography Mission (SRTM) Database [online]. Available from: http://srtm.csi.cgiar.org/. [Accessed 2 February 2008]

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

De Floriani, L., and Magillo, P., 1995. Horizon computation on a hierarchical triangulated terrain model. *The Visual Computer*, 11 (3), 134–149.

Dubayah, R., and Rich, P.M., 1995. Topographic solar radiation models for GIS. *International Journal of Geographic Information Systems*, 9 (4), 405–413.

Fu, P., and Rich, P.M., 2000. The solar analyst 1.0 user manual [online]. Available from: http://www.hemisoft.com. [Accessed 12 December 2007].

Gropp, W., Lusk, E., and Skjellum, A., 1999 Using MPI. Portable parallel programming with the message passing interface, Cambridge MA: MIT Press.

Hershberger, J., 1989. Finding the upper envelope of n line segments in O(n log n) time. *Information Processing Letters*, 33 (4), 169–174.

Losasso, F., and Hoppe, H., 2004. Geometry clipmaps: terrain rendering using nested regular grids, ACM transactions on graphics. *Proceedings of SIGGRAPH*, 23 (4), 769–776.

Marcel, S., and Jaroslav, H., 2004. A new GIS-based solar radiation model and its application to photovoltaic assessments. *Transaction in GIS*, 8 (2), 175–190.

Romero, L.F., and Tabik S., 2008. Horizontal error in horizon computation [online]. Available from: http://www.ac.uma.es/∼siham/epsilon.pdf.

Romero, L.F., et al., 2008. Fast clear-sky solar irradiation computation for very large digital elevation models. *Computer Physics Communications*, 178 (7), 800–808 (accessed 18 November 2009).

Schamer, K., and Greif, J., eds., 2000. Paris: Les Presses de L'ecole des Mines.

Silva, L., and Buyya, R., 1999. *High performance cluster computing programming and applications*. Upper Saddle River, NJ: Prentice Hall.

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

Tabik, S., et al., 2007. Fast insolation computation in large territories. *In*: *International conference on computational science*, 27–30 May 2007 Beijing. China. Berlin: Springer, 54–61.

The OpenMP API specification for parallel programming, 2008 [online]. Available from: http://www.openmp.org/ (accessed 2 December 2008).