CEIG - Spanish Computer Graphics Conference (2016)
Alejandro García-Alonso and Belen Masia (Editors)

Ordering Triangles in Triangulated Terrains Over Regular Grids

J. Alonso[1] and R. Joan-Arinyo[1,2]

[1]GIE Group, Universitat Politècnica de Catalunya, Barcelona,Catalonia
[2]MOVING Group, Universitat Politècnica de Catalunya, Barcelona, Catalonia

Abstract
In this work we report on a set of rules to visit triangles in triangulated height fields defined over regular grids in a back-to-front order with respect to an arbitrary viewpoint. With the viewpoint, we associate an axis-alligned local reference framework. Projections on the XY plane of the local axis and the bisector of the first and third quadrants define six sectors. Specific visiting rules for collections of triangles that project on each sector are then defined. The experiments conducted show that the implementation of a simple algorithm based on the set of visiting rules defined allows real time interaction when the viewing position moves along an arbitrary 3D path.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Efficient visualization of Digital Terrain Models (DTM) is important in many applications such as computer graphics, resource management, earth and environmental sciences, civil and military engineering, surveying and photogrammetry, and interactive 3D games programming.

In this work we report on a minimal, complete and correct set of visiting rules which define a back-to-front ordering of triangles in a coherently triangulated DTM with respect to an arbitrary viewpoint. With the viewpoint, we associate an axis-alligned local reference framework. Projections on the XY plane of the local axis and the bisector of the first and third quadrants define six sectors. Specific visiting rules for collections of triangles that project on each sector are then defined. The set of rules includes rules for uniformly triangulated DTMs and additional rules for fans of triangles that seamless stitch triangles belonging to different levels of detail in level of detail-based renderings.

As far as we know, the only technique to visit triangles in a regular triangulation based on predefined configurations is described in [BN08]. In this work, configurations define a back-to-front ordering of quad cells. The set of configurations given suffers from some drawbacks. For example, configurations for some quadrants are redundant and no specific configurations are given to render quads overlapping more than one sector. In these conditions, the approach can lead to quads which are wrongly rendered and to holes in the surface thus turning the approach useless for realistically rendering terrains.

As a proof of concept, we have implemented an algorithm based on the set of visiting rules defined. The algorithm is simple and only requires graphics boards featuring basic capabilities. The algorithm allows real time interaction when the viewing position moves along an arbitrary 3D path. The algorithm does not suffer from cracking or popping effects.

2. Terrain Representation

We consider terrains represented as DTMs defined over regular grids along both the X and Y axis. Every pair of neighbor heights in a DTM along a sampling axis defines a DTM edge. A loop of four edges defines a DTM cell. Each DTM cell is subdivided into two surface triangles. There are two possible different ways of subdividing DTM cells as shown in Figure 1 where triangle vertices are labeled with grid coordinates. In the sequel, we consider the cells subdivided into triangles as shown in Figure 1a. There is nothing essential in the choice but it has an effect on the resulting set of cell configurations needed to properly render the terrain.

To allow fast processing, compact representations and easy rendering, [DBPN06, Wag04] we organize the terrain into squared blocks each of which consists on a number of *tiles*. In the current implementation, blocks are of size 512×512 and tiles of size 64×64. A block is the basic unit our algorithm considers. The block under consideration is updated as the viewer position changes. Then tiles in the block are rendered.

DOI: 10.2312/ceig.20161313

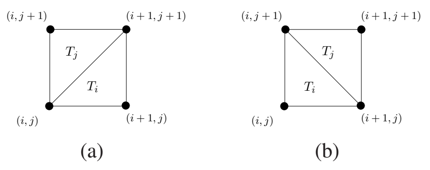


Figure 1: *Projections on the XY plane of two different possible triangulations associated to a DTM cell.*

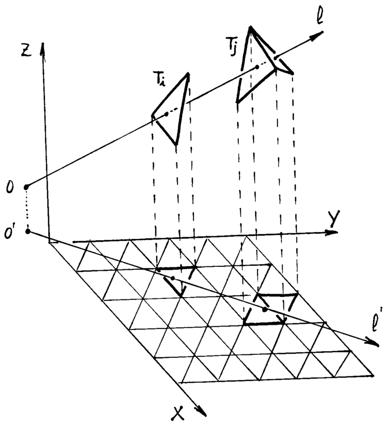


Figure 2: *Projections on the XY plane of: the DTM triangulation, the point of view, the line of sight and the 3D X and Y axis.*

3. Boiling the 3D Problem Down to a 2D Problem

Rendering triangulated terrains over a regular grid takes advantage of the fact that height fields do not allow terrain overhangs and that a triangle is a convex shape. In these conditions, the 3D hidden surface elimination problem can be solved as a 2D problem considering the projection onto the XY plane of the DTM surface triangulation, the viewing position and the line of sight.

Let T_i and T_j be two different triangles in the DTM surface triangulation. Clearly, triangles T_i and T_j share at most one common edge. See Figure 2. Let O be the point of view and l the line of sight as illustrated in Figure 2. Let p_i and p_j be the points where the line of sight l intersects triangles T_i and T_j, respectively. Let T_i', T_j', O', l', p_i' and p_j' denote the parallel projections onto the XY plane of the corresponding geometric elements in the 3D space. Clearly T_i' and T_j' are convex. Since projections preserve incidence, p_i' both is on l' and belongs to T_i' and p_j' is on l' and belongs to T_j'.

Assume that p_i is closer to O than p_j and that the line of sight l through p_i and p_j is not parallel to the Z axis. Then the relationship p_i' is closer to O' than p_j' trivially holds. As considered, DTM triangulations do not allow terrain overhangs. Taken into account that DTM projected triangles are convex and do not overlap, triangles in a DTM triangulation over a regular grid can be sorted according to distances to the viewing point just by considering the projection of the 3D geometry onto the XY plane.

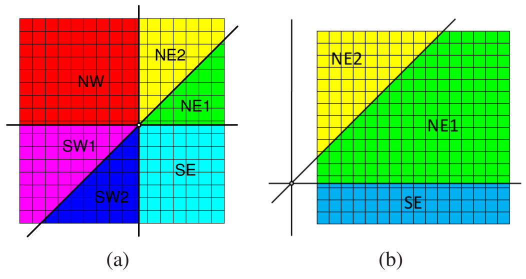


Figure 3: *Sectors defined by a viewing position in a DTM. a) Viewing position projection is within the DTM. b) Viewing position projection is outside the DTM.*

4. Back-to-Front Ordering

A back-to-front ordering of the triangles in the DTM is at the heart of the algorithms that explore triangulations over regular grids for fast processing to solve the visibility problem.

To define a complete and correct set of orderings, we split the projection of the DTM triangulation onto the XY plane as follows. Let $O = (x, y)$ denote the projected viewing position which is not necessarily a grid point. We define a set of local orthogonal axis, X and Y, with origin at the projected viewing position O and aligned with the terrain sampling directions. Now let B be the bisector of the first and third quadrants defined by axis X and Y. The triple $\{X, Y, B\}$ partitions the terrain into different regions that we call *sectors*. When the viewing point projects within the projected triangulation, there are six sectors that we label NE1, NE2, NW, SW1, SW2 and SE as shown in Figure 3a. When the viewing point projects outside the projected triangulation, the number of sector is at most three as depicted in Figure 3b.

Next we describe the set of rules to visit triangles in a tile to guarantee back-to-front ordering. We consider first the case where tiles project within one sector and then tiles the projection of which straddle over different sectors.

4.1. Ordering Triangles in Tiles Within One Sector

With each sector we associate a unique and particular rule that defines the path in which DTM cells must be visited to guarantee a back-to-front ordering of triangles. Consider first a set of terrain tiles placed within the NW sector with respect to the viewing position and the projection of the viewing frustum as depicted in Figure 4a. Visiting the DTM cells following the red arrows from top to bottom and from left to right guarantees a back-to-front ordering for any line of sight l starting at the viewing position and running through the frustum. When the set of tiles to be displayed is within the SE sector all what we need to do is to follow the path defined for the NW sector but in a bottom-up and right-left order, Figure 4b.

Now consider terrain tiles within the NE quadrant. For the DTM cell triangulation we have chosen, see Figure 1a, the relationship *closer than* applied to DTM triangles within a DTM cell, as discussed in Section 3, depends on the line of sight slope.

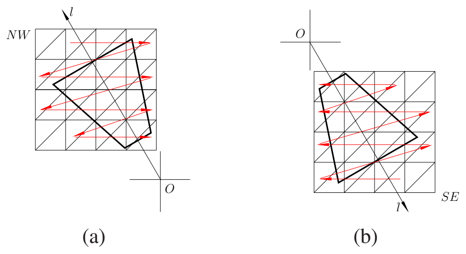


Figure 4: *Back-to-front orderings for terrain tiles within sectors. (a) NW sector. (b) SE sector.*

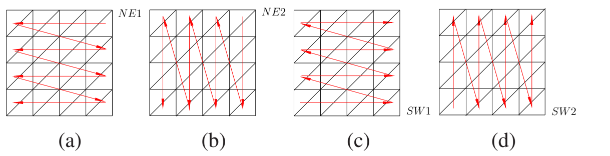


Figure 5: *Sorting triangles within the NE and SW quadrants. a) NE1 sector. b) NE2 sector. c) SW1 sector. d) SW2 sector.*

An analysis of the angle between the line of sight and the bisector leads to the set of visiting rules shown in Figure 5.

When the viewing point projects outside the projected triangulation, some of the sectors discussed above do not appear on the projection plane. See for example Figure 3b. However, the rules for visiting triangles in tiles within sectors described above apply.

4.2. Ordering Triangles in Tiles Straddling Over Sectors

In general, frustum angles are smaller than 90°. Thus the projection of the frustum onto the XY plane straddles at most over three different sectors. For a field of view on the X and Y axis of 60°, Figure 6 shows the projected frustum as a triangle in dashed lines when the viewing position projection falls within the terrain projection.

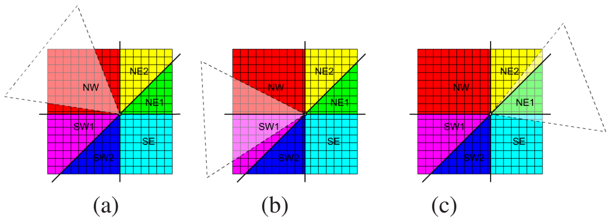


Figure 6: *Number of sectors overlapped by the field of view. a) One sector. b) Two sectors. c) Three sectors.*

When triangles in a DTM tile straddle over two or more sectors, an approach to solve the triangles ordering would consist in two steps. First, one could compute the set of triangles in the tile within each terrain sector overlapping the field of vision. Then to each set of triangles within a sector, we could apply the corresponding visiting rule defined in Section 4.1. However taking the terrain tile as the unit to be rendered leads to a simpler approach.

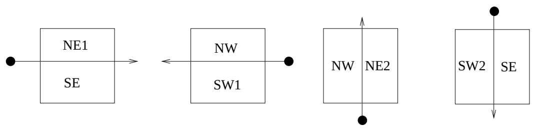


Figure 7: *Sectors involved in the ordering when tiles straddle over X and Y axis. The viewing point projection is outside the tile.*

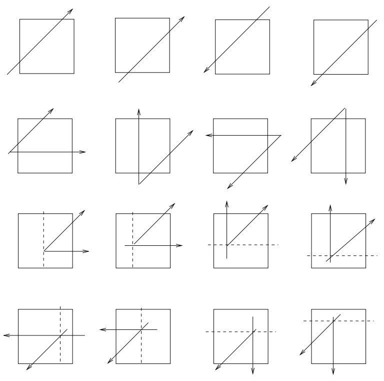


Figure 8: *Sectors involved in the ordering when tiles straddle over bisector B. Top two rows: the point of view is projected outside the tile. Bottom two rows: the point of view is projected within the tile.*

When triangles in a DTM tile straddle over two or more sectors, we first classify tiles according to the tile configuration. Then we define specific rules to visit triangles within each region in the configuration. We distinguish two situations depending on whether the projected viewing position is outside or inside the tile. Then, within each family we consider different configurations depending on the geometry of the regions. Terrain tiles intersected by just the local X or Y axis result in four possible types of regions shown in Figure 7. These regions include two subregions each belonging to just one sector. Thus, visiting rules already defined apply.

When the bisector B intersects the terrain tile there are sixteen different possible sector configurations shown in Figure 8. The two rows at the top correspond to configurations where the point of view is projected outside the tile. The two bottom rows include configurations where the viewpoint is projected within the tile. An analysis of the sets of triangles within each subsector induced by the bisector B leads to four new visiting rules. These new rules are illustrated in Figure 9 where triangles in the strip labeled $k+1$ are closer to the point of view than those in the strip labeled k.

In conclusion, our approach includes a total of ten different back-to-front rules to visit triangles in a triangulated DTM. Six rules correspond to tiles that are projected within a single terrain sector. Four rules are associated with tiles whose projections straddle over more than one terrain sector. On the one hand, we have considered all the possible tile-terrain sector combinations, thus the set of visiting rules is complete. On the other hand, no rule in the set can be reduced to a combination of other orderings in this set, therefore the set is minimal. Since the set of rules always sort the terrain triangles correctly, the resulting set of rules is correct.

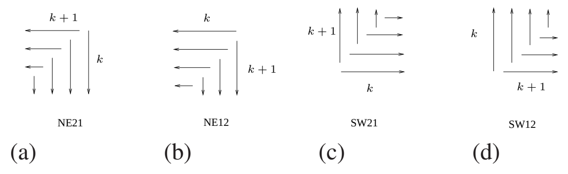


Figure 9: *Sorting rules for subsectors intersected by the bisector B. Sequences for alternatively traversing row and column triangle strips.*

5. Case Study and Discussion

First, as a proof of concept, we have implemented an experimental algorithm based on the set of rules defined in Section 4 to realistically render triangulated DTMs. Pseudo-code for the algorithm is listed in Algorithm 1. We assume that the algorithm is fed with the set of terrain tiles to be rendered which have been properly selected in the DTM model, the point of view *O*, the line of sight *L* and the cullig quadtree depth *D*. The output is the rendered terrain.

Algorithm 1 DTM-painter

```
Input: DTM, a block of terrain tiles
       O, viewing point
       L, line of sight
       D, quadtree depth
Output: A render of the DTM

Q = quadtree(DTM, D)
CT = cullTiles(Q, O, L)
for each tile T in CT do
    if L does not cross T then
        r = identifyRule(T, O, L)
        renderTile(T, r)
    else
        R = splitTileInRegions(T, O, L)
        for each tile fragment F in R do
            r = identifyFragmentRule(F)
            renderFragment(F, r)
        end for
    end if
end for
```

Then, to assess the performance of the algorithm implemented and therefore of the back-to-front ordering technique described, we have implemented two extra algorithms. One extra algorithm was a DTM rendering algorithm using the standard z-buffer provided by the graphics card featured by our computer. The other extra algorithm just renders triangles in tiles using always the NE rule. Clearly, this algorithm does not solve the hidden-surface problem but yields the highest rendering frame rate and is used as a reference. We shall refer to this algorithm as the *naive* algorithm.

The experiments have been conducted on a laptop Pentium Intel Core i7 at 2.20 GHz, with 8GB RAM, featuring an AMD Radeon HD6750M graphics board with 1GB running Visual Studio 2010 under Windows 7. The graphics API used was OpenGL and the GLUT library was used for events and window management.

The benchmark consisted in three different terrains shown in Figure 10 represented as digital elevation models of height fields sampled on a regular grid alligned with the *X* and *Y* terrain axis. The terrain in Figure 10a is a synthetic terrain. Figure 10b is a section of the Grand Canyon carved by the Colorado River in Arizona (USA), [Sur]. Figure 10c shows Mount Ruapehu and Mount Ngauruhoe in New Zealand, [Koo].

For each terrain in the benchmark, we considered two different series of experiments. In one series, the point of view was static, in the other series the point of view moved along an arbitrary 3D path. For each case, we tested three different terrain resolutions with respectively 512×512, 1024×1024 and 2048×2048 uniformly distributed grid points. For the lowest and middle resolutions, eight different quadtree subdivision depths were considered. Due to the limited available storage space, the maximum quadtree depth tested for the highest resolution case was seven.

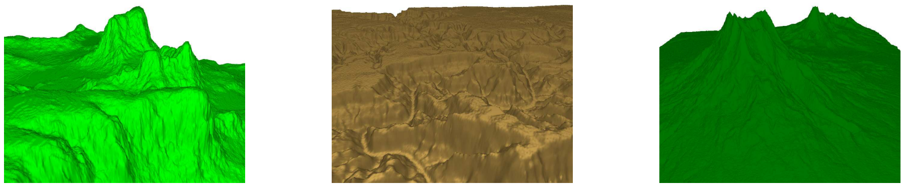
5.1. Fixed Point of View

For the static point of view and 512×512 and 1024×1024 terrain precisions, plots of frame rates follow the same pattern. See Figure 11. For small quadtree depths, curves show a plateau where the number of frames per second rendered is almost constant. Then the frame rate drops off sharply. As expected, the naive algorithm always performed better than both the graphics card z-buffer and our algorithm. In general, our algorithm performs as well as the graphics card z-buffer.

5.2. Moving Point of View

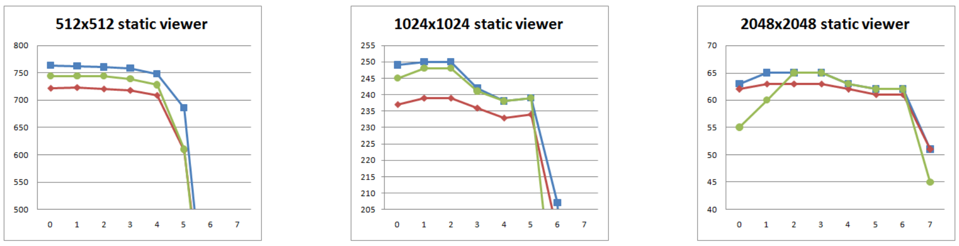
When the viewing point moved along an arbitrary 3D path, the number of frames per second rendered for the terrain precisions considered show patterns consistent with those yielded for the static point of view. See Figure 12. The drop off also appears for quadtree depths of about six and the rationale given for the static point of view also applies.

For small quadtree depths, our approach always performs worse that the graphics card z-buffer. However, performance of our approach steadily increases with the quadtree depth. When the quadtree depth reaches a value of four, the frame rate reaches the plateau where the number of frames per second rendered by our algorithm is equal to that yielded by the graphics card z-buffer.



(a) (b) (c)

Figure 10: *Benchmark terrain models. a) Synthetic landscape. b) Grand Canyon, Colorado River (USA). c) Mount Ruapehu and Mount Ngauruhoe (New Zealand).*



(a) (b) (c)

Figure 11: *Frame rate for static point of view versus quadtree depth. Height points grid of a)* 512 × 512*. b)* 1024 × 1024*. c)* 2048 × 2048*. (■) naive algorithm, (◆) the graphics card z-buffer algorithm, and (●) our approach.*

6. DTMs with Level of Detail

Interactive rendering of large DTMs with high resolution is a challenging problem. Consequently, a number of algorithms have been developed that render simplified representations of terrains, [FTD04]. Among them, multi-resolution terrain models provide efficient mechanisms to represent and manipulate DTM by optimizing the tradeoff between complexity and accuracy of representation.

To adopt the multi-resolution level of detail approach reported in [Boe] also applied, for example, in [ARJ06], we extended the set of rules already defined. We limit the difference between the levels of detail of two neighboring terrain tiles to one. In these conditions, Figure 13 shows the four possible cases of connectivity change that arise in neighboring tiles.

There are four possible tile neighborhoods and each of them can be found within each terrain sector, therefore we need to consider 24 different cases. As an example, Figure 14 shows the rules to visit triangles when the stitching fan is located within viewing sector NE2. Labels in triangles define the visiting sequence. Similar rules have been defined for stitching fans located in NW, SW1 and SW2 terrain sectors. Notice that coincidence of cases by tile rotation does not apply because tile rotation results in an underlying triangulation different from the one chosen. See Section 2.

7. Summary

In this paper we provide a complete, correct and minimal set of visiting rules that define a back-to-front that correctly solve the hidden surface elimination in DTMs over regular grids. In addition, we provide rules to visit fans of triangles that seamless stitch different levels of triangulations.

An experimental implementation of an algorithm based on the set of rules defined for realistically rendering DTMs over regular grids shows that the rules are robust and support real time interaction without the need of exploiting cutting edge graphics cards technology. The approach does not suffer from popping or cracking effects.

The visiting rules described were defined using the specific DTM triangulation chosen in Section 2 and illustrated in Figure 1a. If the triangulation of interest is the one shown in Figure 1b, the path to visit triangles in tile raws or columns in each rule should be reversed.

The approach described can also be applied to visit triangles in a front-to-back order. All what is needed is to reverse the whole path described by each rule.

Acknowledgments

The material in this paper is based upon work supported by the Spanish *Ministerio de Economia y Competitividad* and by FEDER funds under Grant No. TIN2014-52211-C2-1-R.

References

[ARJ06] AGRAWAL A., RADHAKRISHNA M., JOSHI R.: Geometry-based mapping and rendering of vector data over LOD phototextured 3D terrain models. In *WSCG 2006* (Plzen, Czech Republic, January 30-February 3 2006), UNION Agency, Science Press. 5

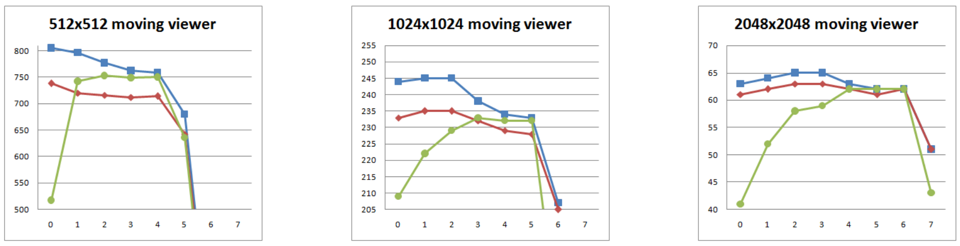


Figure 12: *Moving point of view. Frame rate versus quadtree depth. Height points grid of a)* 512 × 512*. b)* 1024 × 1024*. c)* 2048 × 2048*. (■) naive algorithm, (◆) the graphics card z-buffer algorithm, and (●) our approach.*

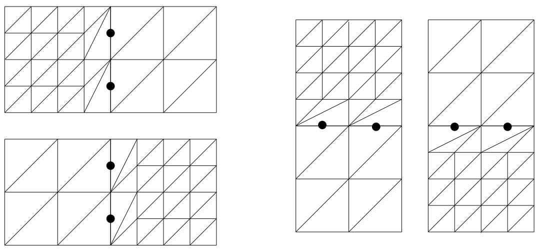


Figure 13: *Neighboring tiles with different level of detail and stitching triangle fans.*

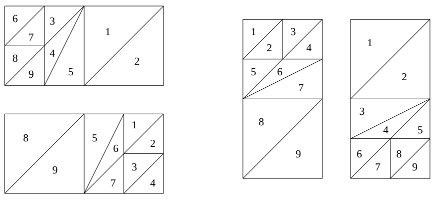


Figure 14: *Visiting sequences for triangles in a stitching fan located within viewing sector NE2.*

[BN08] BHATTACHARJEE S., NARAYANAN P.: Real-time painterly rendering of terrains. In *Sixth Indian Conference on Computer Vision, Graphics and Image Processing* (Bhubaneswar, India, December 16-19 2008), IEEE Computer Society Press, pp. 568–575. 1

[Boe] BOER W. D.: Fast terrain rendering using geometrical mipmapping. E-mersion Project. 5

[DBPN06] DEB S., BHATTACHARJEE S., PATIDAR S., NARAYANAN P.: Real-time streaming and rendering terrains. In *ICVGIP'06*, LNCS 4338. Springer-Verlag, 2006, pp. 276–288. 1

[FTD04] FAN M., TANG M., DONG J.: A review of real-time terrain rendering techniques. In *Computer Supported Cooperative Work in Design: 8th International Conference* (Xiamen, China, 26-28 May 2004), pp. 685–691. 5

[Koo] KOORDINATES: Mount Ruapehu and Mount Ngauruhoe, New Zealand. https://koordinates.com. 4

[Sam90] SAMET H.: *The Design and Analysis of Spatial Data Structures*. Addison Wesley Publ., Reading, MA, 1990.

[Sur] SURVEY U. G.: Grand Canyon, USA. http://www.usgs.gov. 4

[Wag04] WAGNER D.: *ShaderX2: Shader Programming Tips & Tricks with DirectX 9*. Wordware Publishing Inc., 2004, ch. Terrain Geomorphing in the Vertex Shader. 1