Simple Real-Time Flight Over Arbitrary-Sized Terrains

EDUARDO POYART [1], PAULA FREDERICK[1], ROBERTO DE BEAUCLAIR SEIXAS[1,2] , MARCELO GATTASS [1]

[1] Tecgraf - Computer Science Department, PUC-Rio,
Rua Marquês de São Vicente 225, 22450-900 Rio de Janeiro, RJ, Brazil
{poyart, paula, tron, gattass}@tecgraf.puc-rio.br
[2] IMPA– Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina 110, 22460-320 Rio de Janeiro, RJ, Brazil
tron@impa.br

Abstract. Terrain visualization is an area of active research that has yielded many results in the last decade. Despite all the literature generated, there is still need for a simple and effective solution that takes advantage of graphics boards available in current consumer PCs. This paper describes an implementation of a modified version of the Geometrical Mipmaps proposed by de Boer [1]. Our version allows arbitrary sized terrains and is not restricted to square terrains with a side measuring $2^k + 1$ samples. Furthermore it is simpler and requires no preprocessing stage, thus allowing its direct insertion in an existing open GIS such as TerraLib. Results are shown to validate the ideas presented here and to suggest future works. The source code and some test examples are available in the internet to promote discussion.

Keywords. Terrain visualization, Geometrical Mipmaps, flight over terrains.

1 Introduction

There are many applications that require a simulated flight over a terrain. These applications vary from computer games to computer-assisted installations of well equipment on the ocean bed.

The literature on algorithms and data structures to support this kind of visualization is vast. The performance issues are mainly two: how to quickly discard the portion of the terrain that is outside the frustum (culling) and how to produce an optimal mesh to render the rest of the terrain. "Optimum" here means a mesh with the smallest number of triangles that has little perceptual difference to the full mesh.

Two main strategies seem to have prevailed over the others. The first is based on the papers co-authored by Hugues Hoppe [2,3,4]. Hugues created a very effective multi-resolution scheme for arbitrary surfaces is his first work [2], but later, when he introduced view-dependent simplification, the algorithm became very complex.

The second strategy is based on the papers co-authored by Peter Lindstrom [5,6,7,8]. Lindstrom's work seeks simple algorithms for terrain visualization and has achieved a great deal in that direction. A further approach in this view was proposed by de Boer [1].

Despite all these contributions, there is still need for a simple and effective solution that takes advantage of graphics boards available in current consumer PCs. The present paper describes an implementation of a modified version of the Geometrical Mipmaps proposed by de Boer [1]. Our version allows arbitrary sized terrains and is not restricted to square terrains with a side measuring $2^k + 1$ samples. Furthermore it is simpler and requires no preprocessing stage, thus allowing its direct insertion in an existing open GIS such as TerraLib. Results will be shown to validate the ideas presented here and to suggest future works. The source code and some test examples are available in the internet to promote discussion.

2 Geometrical Mipmapping

The Geometrical Mipmapping (Geo-mipmapping) technique works as follows. Each terrain block has a maximum resolution level, in which all data from the height map is used. From the maximum-resolution block, several lower-resolution blocks are generated. These lower-resolution blocks have dimensions that are half the original, a quarter of the original and so on. When a terrain block falls under a certain distance d to the camera (or to the projection plane), this block is rendered with its full resolution.

When a terrain block is far enough from the viewer, it can be rendered at a lower resolution. The Geo-mipmapping technique is analog to the mipmapping technique for textures. A block of level 0 uses all the corresponding samples from the terrain. A block of level 1 has half the width and height of the original block, and so on. A function of d can be used to determine the level of mipmap to be rendered at each time. The lower-resolution blocks can be pre-calculated using some kind of filtering.

When two blocks with different resolution have to be rendered together, they must be stitched in order to eliminate visible gaps. De Boer describes a convenient way to do this stitching: triangular fans are used to connect a lower-resolution block to a higher-resolution block.

De Boer describes a way to implement this technique when the terrain blocks have dimension $2^k + 1$, where k is an integer ($2^k + 1$ samples imply 2^k segments of terrain, a segment of the terrain being the space between two samples). As it is usually necessary to structure the terrain in a conventional quadtree, this implies that the whole terrain should be a square with sides of the dimension of a power of 2 plus one.

3 Geometrical Mipmapping on Arbitrary-Sized Terrains

Our implementation of the terrain renderer has two major differences from de Boer's Geometrical Mipmapping. In the first place, it can be used with arbitrary-sized terrains, not only square terrains with a side measuring $2^k + 1$ samples. Secondly, it is a simpler version, being easier to implement and not requiring preprocessing or special data structures to hold mipmap levels. Details about the implementation of the algorithm, which are not presented by de Boer, will be shown in this work.

3.1 Quadtree Generation

To build a conventional quadtree on a terrain, the data should be divided exactly in two for the height and width of the terrain, so that the four children of a node have exactly the same dimension. This division must continue until a node reaches a minimum size.

Our quadtree relaxes the exactness of this division. The terrain can have any size, which will lead to variable-sized leaf nodes.

The quadtree-generation method is recursive and is implemented in the constructor of class Quadtree. It is given a minimum size for the leaf nodes. When of the dimensions of the current area is smaller than that minimum size, the division ends. Otherwise, the width and height is divided in two and rounded down if the size is odd. The procedure is then called recursively for each of the four children.

The data held by each node of the quadtree are simply the minimum x and y and maximum x and y coordinates of the terrain represented by this node. The root node of the quadtree stores the index of the minimum and maximum coordinates of the whole terrain. Each leaf node has the indices of the block it represents. The actual samples remain in an array that is created when the height map is loaded.

3.2 Terrain Renderer

When the terrain is rendered, the variable-sized leaf nodes must be treated by a special procedure: drawArea, in class Terrain. Its responsibility is to correctly render a node with any size at any resolution level, stitching it to the terrain block at its east and south. To simplify the multiresolution method, we have opted for simply skipping samples in order to build a lower-resolution version of a block. Therefore, lower-resolution blocks do not have to be pre-generated. We will now examine the drawArea method by means of a top-down approach.

The drawArea method has to draw the main area of the terrain and its east and south connections if necessary. It computes a step, which is a function of the mipmap level calculated for the block. When the mipmap level is 0, the step is 1, which means every sample is used. When the mipmap level is 1, the step is 2: the samples are picked in steps of 2. When the mipmap level is 2, the step is 4, and so on. Therefore, step = 2^m, where m is the mipmap level.

To draw the main area of the terrain, drawArea calls the method drawHStrip, passing to it the step to be used. This method draws one horizontal strip of triangles. It must take care when the step doesn't divide the strip by an exact number of samples. If this happens, a thin square is drawn at the end of the strip. The method drawArea calls drawHStrip as many times as necessary to draw the whole main area. Figure 1 shows the main and connection areas of drawArea, and Figure 2 shows the general case of drawHStrip's output, with a thin square at the end. The black dots are used samples and the white dots are unused samples at this resolution level. In this case, the step is 2.

With all the main area of a block completely drawn, it is time to draw the horizontal and vertical stitching. If the resolution of the east block is the same as the current one, no horizontal stitching is needed and drawArea will already have handled it by extending the horizontal strips up yo the end of the block. If the resolution of the south block is the same as the current one, drawArea handles it by drawing one more horizontal strip to complete the block.

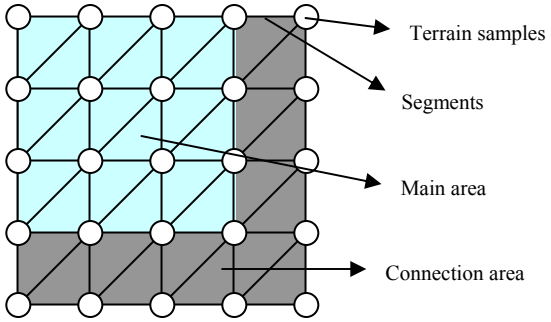


Figure 1 – A terrain block

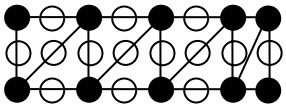
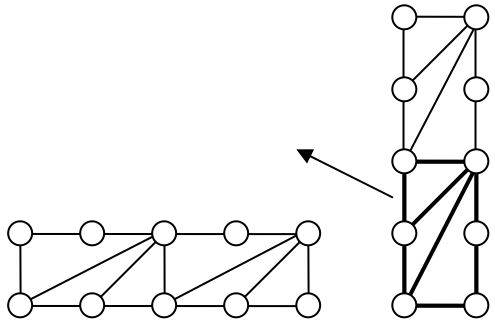


Figure 2 – Output of the drawHStrip method.

If either the east or the south block has a different resolution from the current block, either a horizontal or a vertical connection (or both) must be drawn. This is handled by the methods drawEastConnect and drawSouthConnect. The output of these methods is shown in Figure 3. Each of them draws a series of fans to connect the blocks. Each fan is drawn by a flexible fan-drawing method, drawFan, which draws the fans connecting any given resolution with any other resolution, in any orientation (horizontal or vertical, with the lower resolution being above or below, to the left or to the right). In other words, the responsibility of correctly drawing a fan is held by drawFan, and the methods drawEastConnect and drawSouthConnect use drawFan by calling it with the correct parameters in order to obtain the intended results.



It is important to notice that drawEastConnect and drawSouthConnect must be flexible enough to draw connections with any thickness. The thickness of the connections can vary from one single segment to the value of the step used in the block. This implies that drawFan must have this same flexibility.

The reader can refer to [9] to examine the source code of our implementation. Some screen outputs are shown below, on figure 4.

4 Conclusion

Our terrain renderer was a lot simpler to implement than the version described by de Boer. It also has the advantage of being able to render terrains of any size without the need to increase the size to a power of 2 plus one.

The simplifications introduced two visual differences from the original idea. First, the use of samples without filtering for lower resolutions. Filtering can considered better, but both techniques introduce visual differences when a lower-resolution version of a block is used. Filtering does not eliminate this problem completely, so we are not introducing a new problem, and there is the advantage of eliminating the need to preprocess the terrain, which can be useful for some applications.

Second, thin strips of terrain can be a source of visual artifacts, but, when rendering a texturized terrain using a lightmap technique for illumination, the difference is unnoticeable. Thin terrain strips can only be noticed when viewing a wireframe version of the terrain, and in some specific situations when vertex lighting is used.

Therefore, development speed and integration with other pieces of software can be increased by simplifying the Geo-mipmapping technique. Some compromise in the visualization exists, but the result is still very acceptable.

This technique can be expanded in the same ways as the Geomipmapping technique, such as performing morphing in different resolution levels in order to reduce popping, and choosing the level of mipmapping based on screen pixel error, not only camera distance. Particularly the second one is supported by the drawFan, drawEastConnect and drawSouthConnect implementation. These methods are designed to work correctly even when rendering two blocks with a difference in resolution level greater than 1. For example, they can correctly connect a block of level 0 and a block of level 2, although this is not done on the current implementation.

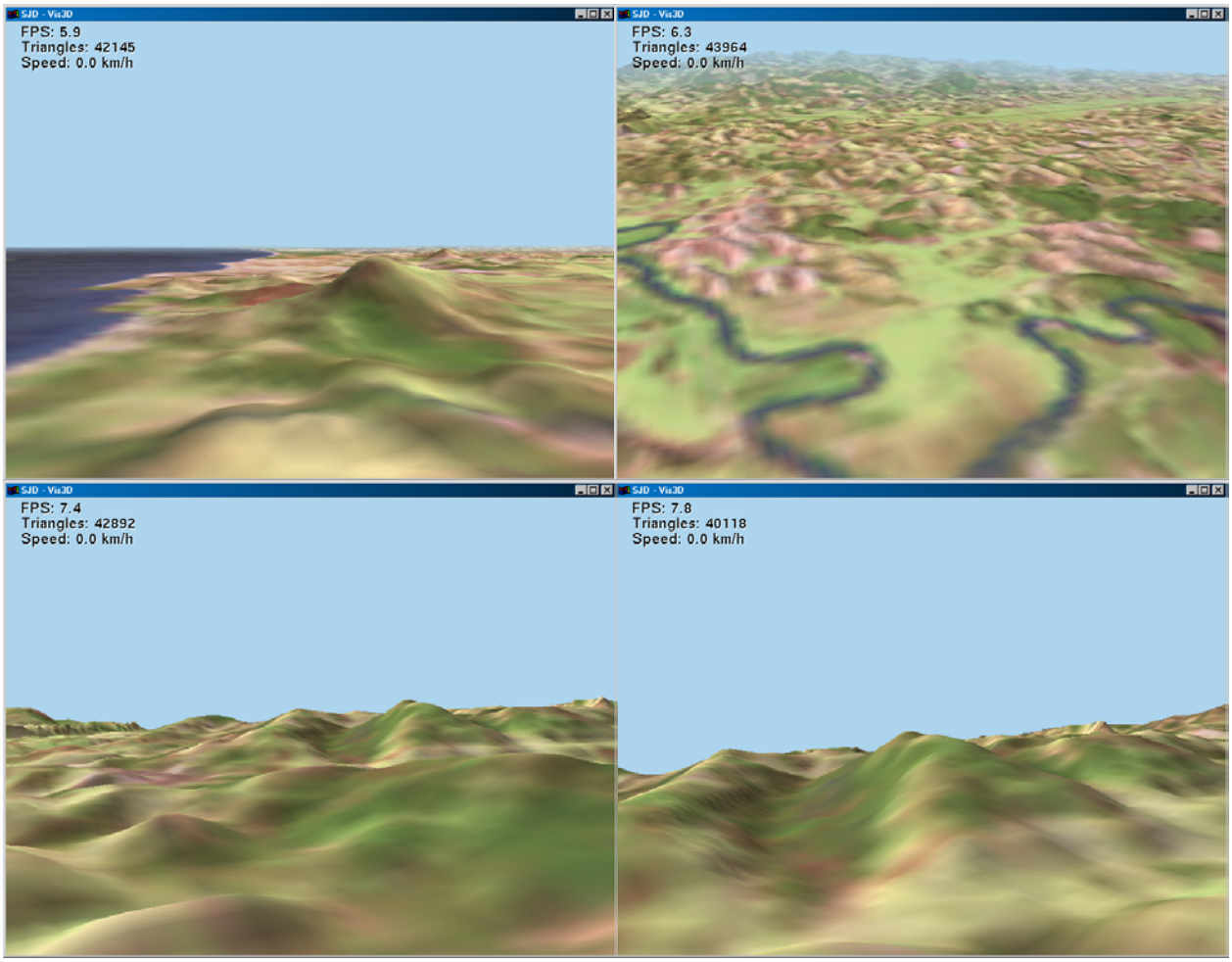


Figure 4 - Images of the terrain renderer.

Acknowledgements

Tecgraf/PUC-Rio is a laboratory mainly funded by PETROBRAS and by the Brazilian Marines Navy. The visualization group from CENPES and IMPA provided valuable suggestions.

References

[1] de Boer, *Fast terrain rendering using geometrical mipmaps*, http://www.flipcode.com/tutorials/geomipmaps.pdf, October 2000.

[2] Hugues Hoppe, *Progressive Meshes,* ACM SIGGRAPH 1996 proceedings, pages 99-108.

[3] Huges Hoppe, *View-dependent refinement of progressive meshes*, ACM SIGGRAPH 1997, pages 189-198.

[4] Hugues Hoppe, *Smooth view-dependent level-of-control and its application to terrain rendering*, IEEE Visualization 1998, pages 35-42.

[5] Kooler, Lindstrom, Ribarsky, Hodges, Faust and Tuner, *Virtual GIS: a real-time 3d geographical information system*, IEEE Visualization 1995, pages 94-100.

[6] Lindstrom et all., *Real-time, continuos level of detail rendering of height fields*, ACM SIGRRAPH 1996, pages 109-118.

[7] Peter Lindstrom and Valerio Pascucci, Visualization of large terrains made easy, IEEE Visualization 2001, pages 363-370.

[8] Peter Lindstrom and Valerio Pascucci, Terrain Simplification Simplified: A General Framework for view-depandent out-of-core visualization, IEEE Transactions on Visualization and Computer Graphics, 8(3), pages 239-254, 2002.

[9] Código fonte do renderizador de terrenos na Web