Efficiency of RINGO Algorithm for Large Terrain Rendering

ALEKSANDAR DIMITRIJEVIĆ, DEJAN RANČIĆ
Department of Computer Science, Faculty of Electronic Engineering
University of Niš
Aleksandra Medvedeva 14, 18000 Niš
SERBIA

Abstract: – Nowadays, many time-critical applications require 3D rendering algorithms for the visualization of massive terrain datasets. For such applications it is essentially important to use CPU processing power as little as possible, because in real-life applications the CPU usually has more task to do than just a terrain drawing. There is a huge burden imposed by sensors polling, received information interpretation, communication with other devices and their management. This paper presents efficiency of simple out-of-core terrain-rendering algorithm, using OpenGL display lists organized into nested rings.

Key-Words: – Terrain rendering algorithm, out-of-core, level-of-detail, block based organization, efficiency

1 Introduction

A large volume of applications today requires some kind of terrain visualization technique in order to display realistic outdoor scenes. The most important trigger force for developing efficient algorithms for the visualization, which in the other hand should have a very pleasant outcome, are the interactive computer games. The gaming industry also boosts the graphics-accelerators hardware development, transforming plain PCs into very powerful graphics workstations. Many of the currently state-of-the-art algorithms are targeted to this marketplace.

Besides the entertainment marketplace, the terrain rendering algorithms find its appliance also in scientific applications, such as simulators, geographic information systems (GIS), and even in military mission planning applications. For such applications it is essentially important to use CPU processing power as little as possible, because in real-life applications the CPU usually has more task to do than just a terrain drawing. There is a huge burden imposed by sensors polling, received information interpretation, communication with other devices and their management.

This paper presents the efficiency of the RINGO, the algorithm for the out-of-core large terrain rendering. The solution is developed in the CG&GIS Laboratory, at Faculty of Electronic Engineering in Niš, for the implementation in the heavily loaded geographic information systems. The main requirements that govern the development of the algorithm are:

- Low CPU utilization,
- Highest possible rendering rate, and
- Portability.

The paper is organized as follows: the second section gives brief overview of terrain rendering algorithms, the third section explains the general principles of the RINGO algorithm, the fourth section shows some experimental results and the efficiency, and the conclusions are given in the fifth section.

2 Related Work

In the early days of the terrain rendering algorithms, predominant task was to maintain a minimal number of polygons in the scene. This led to various level-of-detail (LOD) approaches enabling visualization of massive terrain models in spite of the restriction imposed by the graphics hardware.

Some early works in terrain simplification dated from late 1970s [1], but the real explosion of the algorithms started in 1990s [2-4]. All those algorithms tried to optimize the number of graphics primitives by using the CPU power. Generally, there are two approaches in terrain simplification:

- Bottom-up – starting with highest-resolution mesh that has to be iteratively simplified, until desired resolution is achieved [2], and
- Top-down – starting with two or four triangles, which have to be progressively tessellated, until desired resolution is achieved [3,4].

The data structures used to represent the terrain can be: regular gridded height fields or triangulated irregular networks (TINs). TINs achieve required

accuracy with fewer polygons, but they are much harder for manipulation, comparing to regular grids. Furthermore, today graphics processing units (GPUs) easily handle large data sets with regular (uniform) structures, so many algorithms rather relay on regular gridded data. The optimal feeding of graphics pipeline is now more important than the fine tessellation tuning.

All algorithms developed in the last few years are based on the large triangular or rectangular patches at different resolutions. Some of the most famous are: Geometry Clipmaps [5,6], Spherical Clipmaps [7], Seamless Patches [8], BDAM [9], and its variations P-BDAM [10] and C-BDAM [11].

A great number of previously mentioned algorithms deals with terrain data that is already loaded into main memory. But, it is fairly desirable feature to enable out-of-core operation. This means that the algorithm should be able to browse terrain data set that exceeds the size of the available main memory. There are many proposed solutions for optimizing terrain data layout on the hard disk drive, and improvement of spatial coherence [12-14].

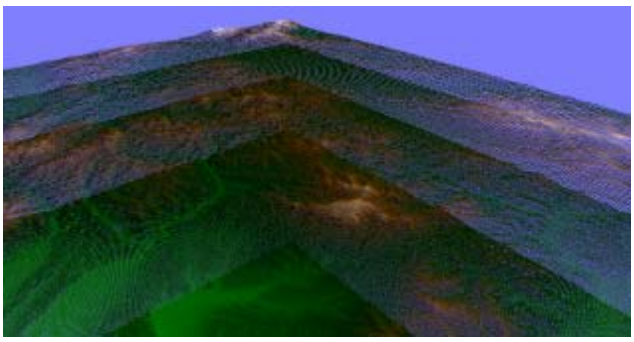

Fig.1 *RINGO approach - Concentric rings of terrain datasets*

In the next section we will present our algorithm, called RINGO, due to its data-organization into concentric rings (Fig.1). It is greatly influenced by Geometry Clipmaps, the algorithm that caches the terrain in a set of nested regular grids centered about the viewer. An important aspect of this approach is that the level of detail (LOD) is independent of data content, and therefore the terrain data does not require any precomputation of refinement criteria. The only drawback of the Geometry Clipmaps is their planar regular grid used for the terrain creation. As the main purpose of the RINGO is to enable fast terrain visualization in GIS applications, RINGO relies on regular grids, but in spherical coordinate system. That enables very precise Earth surface modeling, using WGS84 ellipsoid, geoid correction and various sources of data in standard file formats.

3 Algorithm Overview

RINGO algorithm [15,16] uses very simple scheme to maintain a large terrain dataset. The primary sources of data are files containing terrain height values in any of the supported file formats (e.g. BIL, HGT, etc.). According to viewer height above the geoid and chosen LOD scheme [15,16], a part of dataset is cached in the main memory.

The greatest possible frame-rate for the given graphics hardware can be achieved by using so called "retain mode". This assumes that the data is arranged into appropriate buffers, enabling fast DMA transfer to the graphics card memory. Like data, the manipulation commands, e.g. geometric transformations, can also be precompiled and stored in graphics memory.

To obtain cross-platform scalability, RINGO uses OpenGL for implementation of its rendering engine. Furthermore, only basic functionality is used to avoid compatibility problems with different hardware. Experimental results show that the display lists are the most efficient way to render a scene using OpenGL, especially when there is a bottleneck in the computer system [15].

Although very efficient, the display lists have some drawbacks, like:

- Additional time required for creation and destroying, and
- Disability of changing.

These drawbacks disable some advanced techniques, such as vertex morphing, but minimization of CPU load and maximization of speed justifies the use of display lists.

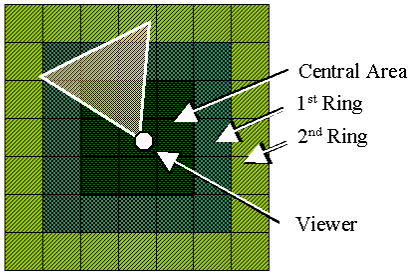

Fig.2 *The ring organization of terrain blocks*

In order to achieve better utilization of OpenGL display lists, as well as partially loading terrain data, the whole terrain that resides in main memory is divided into blocks. Each block has the same spatial size, but resolution depends upon the distance from the viewer. All terrain blocks are organized into one central area and the certain number of rings (Fig.2). The central area covers 3×3 blocks of the highest

resolution, and each ring consists of blocks with four times less triangles than their inner neighbors.

When viewer crosses the border of the central block, blocks reorganization starts. Some of the blocks have to be re-created (in Fig.3 blocks marked with 'R' and 'D'), while others just change their position in the terrain matrix (by pointers moving). Blocks re-creation is the only time-critical operation in our algorithm. The number and the resolution of re-created blocks directly determine the fluidness of the transition. The time for the blocks re-creation depends whether the blocks resides in front or behind the viewer. The blocks that are behind the viewer, marked with 'D', can reuse data from their neighbors with the higher resolution, while blocks that are in front of the viewer, marked with 'R', require reload and recalculation. Even these blocks, which have to be reloaded, can reuse data from the lower resolution neighbors. But since the lower resolution block contains just the quarter of required data, and certain calculation should be done even for the reused data, the speed gain is very poor. For the sake of simplicity, current implementation of our algorithm, completely reloads R-blocks.

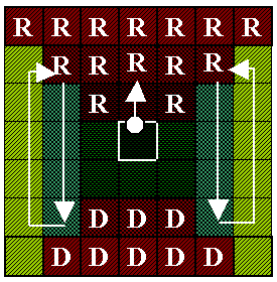

Fig.3 *Viewer crossing the border of the central block*

One of the greatest advantages of this algorithm is that it enables high level of parallelism. Namely, loading and preparing each block is totally independent of the others. Considering the case when viewer is moving to the north, each column of the terrain-blocks matrix can be reorganized in the separate thread (Fig.4).

Each thread examines blocks in its own column starting from the opposite side of the matrix in regard to viewer movement direction. If the current block has the same LOD level as its neighbor (in the direction of movement), working thread copies the pointer of the neighbor to the current block. If it is not true, current block have to be re-created by reloading or downsampling from the higher resolution neighbour in a separate thread (depicted as dashed lines).

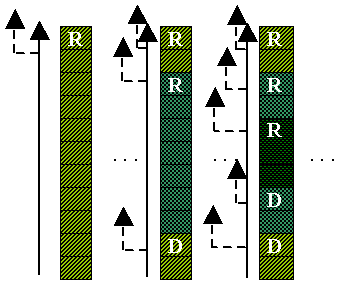

Fig.4 *Example of thread initialization for the moving-north motion*

The high parallelism in terrain reorganization already enables smooth block transitions on dual-core sistems, and it is very likely that new emerging technologies, such as quad-core procesors, will further boost the speed.

4 Results

The greatest advantages of RINGO algorithm are:

- Very simple implementation,
- High level of parallelism in the terrain reorganization, and
- Highest possible frame-rate while moving within the boundaries of the central block.

A set of experimets were cerried out, on verious platforms, to obtain measure of RING effectiveness. Table 1 summarizes used computer configurations, and table 2 displays their speed-test results. In this test we used terrain consists of 121 blocks organized in central area and 4 rings (11x11). The total number of triangles is 1.746.000.

Label	Graphics	CPU	OS
R1	X850 Pro	PentiumD@3.0GHz	XP
R2	X600	Athlon64 3000+ 1.81GHz	XP
R3	Nv6200	Pentium4@2.8GHz HT	XP
R4	Fx5500	Celeron2@800MHz	W2K
R5	GF7300GT	Pentium4@3.0GHz	W2K

Table 1. *Configurations used in the speed test*

Label	CPU Usage		FPS
	Inside block	Moving across	
R1	6 %	25 %	274
R2	13 %	50 %	280
R3	4 %	20 %	1200
R4	24 %	100 %	300
R5	3%	30 %	1062

Table 2. *Speed test results*

The first column in the table 2 contains references to appropriate configuration in table 1. The second colon shows CPU usage while viewer is moving through the central block. The usage ranges

from 3% to 24%, strictly depending on the CPU speed, because RINGO in every frame calculates:

- Distances to the near and far clipping plane of the viewing frustum and sets projection matrix, in order to accomodate variable viewing distance, from few meters to thousands of kilometers,
- Height above geoid,
- Collision with the terrain (and height above it),
- Movement parameters toward viewing course or paralell to a colliding surface, and
- Azimuth (measured clockwise from the north pole).

The third column in the table 2 shows CPU usage while viewer moves across the border of the central block. It is the only time-critical task in this algorithm, because it requires terrain reorganization. As it can be seen from the Fig.5, border-crossing increases the CPU load for the very short period of time.

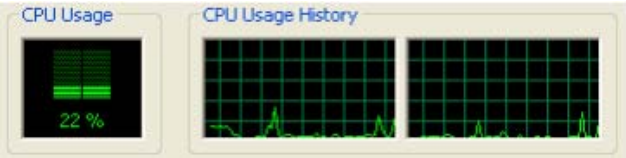

Fig.5 *CPU usage for the configuration R1 while viewer moves across the border of a block*

The last column in the table 2 contains "effective" frame-rate. It is not the real speed of the rendering, but the time while CPU is busy sending commands to a graphics card. This means, for example, that CPU in the configuration R1 spends about 3.65ms for each screen refreash. The standard input devices (e.g. keyboards) are used for actuators in all tests. If we artificially boost the screen refresh rate across certain limits the system will be saturated. For example, for the system R4 saturation arise if RINGO is forced to redraw screen every 20ms. This is the burden when CPU usage jumps to 100% and frame-rate drops to 47 fps.

The system saturation has not been observed on faster configurations (e.g. system R5) even when screen is redrawn up to 1000 times per second.

In all tests NVidia's graphics cards have achieved higher frame-rate indicating better optimization of drivers comparing to ATI.

5 Conclusion

The presented algorithm offers four main advantages: a simple implementation, maximal frame-rate that the graphics card can achieve, minimal CPU usage when the viewer remains in the central block and high parallelism in blocks re-creation procedure. The maximum frame rate and minimum CPU utilization can be achieved if and only if all terrain blocks can reside in the memory of the graphics card. In other case, for example when algorithm is executed on integrated video cards, performances are very poor, but still better than using other rendering techniques. The reason is very simple: the modern graphics card memory bandwidth is more than 20GB/s. It cannot be compared to even the fastest DMA to the main memory.

The speed of algorithm depends on spatial block size, number of blocks in the terrain matix and blocks arrangement (i.e. LOD scheme). Disccusion about the influence of all those factors exceeds the scope of this paper. RINGO is still under development, and certainly requires coarser or finer tuning in many aspects, but currently results promise successful usage in various applications.

References:

[1] R. J. Fowler and J. J. Little, Automatic Extraction of Irregular Network Digital Terrain Models, *Proceedings of SIGGRAPH,* 1979, pp. 199–207.

[2] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner, Real-time, continuous level of detail rendering of height fields, *ACM SIGGRAPH,* 1996, pp. 109-118.

[3] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, and M. Mineev-Weinstein, ROAMing terrain: Real-time optimally adapting meshes, *IEEE Visualization,* 1997, pp. 81-88.

[4] S. Röttger, W. Heidrich, P. Slussallek, and H-P Seidel, Real-Time Generation of Continuous Levels of Detail for Height Fields, *Proceedings of International Conference in Central Europe on Computer Graphics and Visualization*, 1998, pp. 315–322.

[5] F. Losasso and H. Hoppe, Geometry clipmaps: Terrain rendering using nested regular grids, *ACM SIGGRAPH 2004*, pp. 769-776.

[6] A. Asirvatham and H. Hoppe, Terrain rendering using GPU-based geometry clipmaps, *GPU Gems 2*, 2005, pp. 27–45.

[7] M. Clasen and H.C. Hege, Terrain rendering using Spherical Clipmaps, *Eurographics/IEEE-VGTC Symposium on Visualisation*, 2006.

[8] Y. Livny, Z. Korgan and J. El-Sana, Seamless Patchess for GPU-Based Terrain Rendering, *WSCG 2007, 15th International Conference in Central Europe on Computer Graphics,*

Visualization and Computer Vision, *Science Press*, Plzen, Chech Republic, 2007.

[9] P. Cignoni et al., BDAM – Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization, *EUROGRAPHICS, Volume 22, number 3*, 2003, pp. 505–514.

[10] P. Cignoni et al., Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM), IEEE Visualization 2003, pp. 147-154.

[11] P. Cignoni et al., C-BDAM - Compressed Batched Dynamic Adaptive Meshes for Terrain Rendering, *EUROGRAPHICS, Volume 25, number 3*, 2006.

[12] J. S. Falby, M. J. Zyda, D. R. Pratt, and R. L. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers and Graphics.* vol. 17(1), 1993, pp. 65–69.

[13] M. Reddy, Y. G. Leclerc, L. Iverson, and N. Bletter, TeraVision II: Visualizing Massive Terrain Databases in VRML. *IEEE Computer Graphics and Applications.* vol. 19(2), 1999, pp. 30–38.

[14] P. Lindstrom, and V. Pascucci, Visualization of Large Terrains Made Easy, *Proceedings of IEEE Visualization,* 2001, pp. 363–370, and 574.

[15] D. Rancic, A. Dimitrijevic, B. Predic, RINGO – Block Based Algorithm for Large Terrain Rendering, *Proc. of the 6th WSEAS Int. Conf. on Signal Processing, Computational Geometry & Artificial Vision*, Elounda, Greece, 2006, pp. 16-20.

[16] D. Rancic, A. Dimitrijevic, B. Predic, Spatial Coherency and Parallelism in Blocks Reorganization of RINGO Algorithm for Large Terrain Rendering, *WSEAS TRANSACTION on COMPUTERS*, Issue 12, Vol.5, 2006, pp. 3073-3079.