A COMPARATIVE STUDY OF THE MARCHING CUBES TECHNIQUES FOR 3D REAL-TIME RENDERING OF SCENES IN VIDEOCONFERENCING SYSTEMS

Maria Ángeles Calabuig, Moisés Ferrer, Mariano Alcañiz, Juan José Fuertes
Instituto Interuniversitario de Investigación en Bioingeniería y Tecnología Orientada al Ser Humano
Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

Mariano Alcañiz
Ciber, Fisiopatología Obesidad y Nutrición, CB06/03 Instituto de Salud Carlos III, Madrid, Spain

Keywords: 3D Immersive videoconferencing, Scene reconstruction, Model rendering, Marching Cubes.

Abstract: This paper presents a comparative study of the Marching Cubes techniques and describes their optimised implementations for a 3D videoconferencing system with the aim of obtaining maximum efficiency and flexibility. Different approaches in the CPU and in the GPU are compared in order to analyze the influence of the rendering process in the reconstruction algorithm. To do this, several 3D scenes were captured, codified and sent to a real-time rendering system that reconstructs 3D models at high speed.

1 INTRODUCTION

Videoconferencing systems are becaming highly popular communication tools that are used in different contexts (Kauff and Schreer, 2002; Regenbrecht et al., 2003; Chu et al., 2009); however, they do not use the same rendering methods (Pierleoni et al., 2006; Weik et al., 1998; Riegel and Kaup, 1997). Rendering, an important part of these systems, is the process of generating an image from a 3D model, a scene or a volume of voxels, by means of computer programs.

One of the most popular techniques for rendering tasks of 3D real-time reconstruction techniques is the Marching Cubes algorithm (Newman and Yi, 2006), that it has been used and improved for quite a while, such as the GPU version created by Christopher (Dyken et al., 2008) which is known as Histopyramids. Marching Cubes implementations have been used in biomedicine, processing, and natural phenomena rendering. However, to the authors' knowledge, there is no work about computer graphics in multiuser videoconferencing systems.

The aim of this paper is to present a comparative study of different algorithms for 3D real-time reconstruction that are used in a high immersive videoconferencing system developed in the VISION project. In section 2, an overview of the system is presented, including the reconstruction module. Section 3 describes the different implementations of the Marching Cubes algorithm with special emphasis on innovative contributions. In section 4, we present a comparative study of the experimental results about the influence of the rendering process on the different implementations inside a real videoconferencing system. Finally, section 5 summarizes the conclusions of the work.

2 SYSTEM OVERVIEW

The architecture of the system is composed of several modules (capture, coding, communication, reconstruction, and presentation), from the user who is captured at one end of the system to the end user who receives a visualization of the first user after being reconstructed in real time (see Fig. 1).

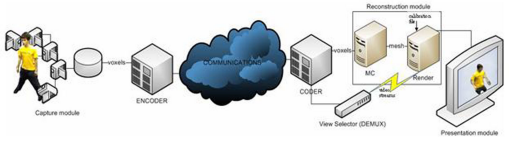

Figure 1: The system block diagram.

2.1 Inputs of 3D Reconstruction

Multi-camera technology is used to capture 3D scenes. In a smart room, eighteen cameras are syn-

Ángeles Calabuig M., Ferrer M., Alcañiz M. and José Fuertes J..
A COMPARATIVE STUDY OF THE MARCHING CUBES TECHNIQUES FOR 3D REAL-TIME RENDERING OF SCENES IN VIDEOCONFERENCING SYSTEMS.
DOI: 10.5220/0003320102030206
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2011), pages 203-206
ISBN: 978-989-8425-45-4

chronized and calibrated spatially to register the 3D sensor data in the same world coordinate frame. The capture module uses the cameras' data to obtain the volume of voxels which is the input to the Marching Cubes algorithm. The data acquired by the cameras are also the texture inputs to the reconstruction system that comprise the Marching Cubes algorithms for mesh generation and the render block.

2.2 Render Block

The render block receives meshes, which are the result of the Marching Cubes algorithm. It also receives the calibration file that is sent from the capture module to find correspondences among the cameras of the capture module and the reconstruction module. Finally, it receives the textures from the demultiplexer device that selects the textures for each frame.

The eighteen cameras are available in the render block but 4 of them are only used in texture mapping. The best 4 views are updated and chosen in each frame by the demultiplexer among the 18 available views.

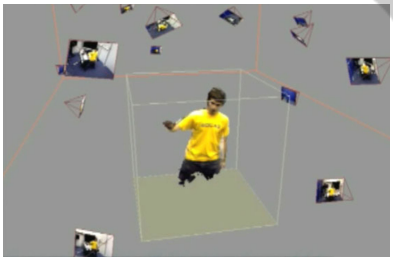

Figure 2: Volume texturization from the best 4 views.

3 MARCHING CUBES

Marching Cubes (MC) is a simple method for extracting isosurfaces (Lorensen and Cline, 1987). For every position in the voxel volume, neighbouring voxels are checked to determine whether they are inside the volume. Then an identifier is acquired, a lookup table is checked, and the extracted triangles are obtained. The chosen look-up table does not introduce ambiguities (which would yield to topology errors in the mesh) at the expenses of generating 5 triangles at most per voxel (Magnor, 2005).

The extraction of triangles is always performed locally. By running over the entire volume, all the triangles are obtained and combined to form the final mesh. This locality is exploited to boost performance by parallel programming techniques. In this work, several versions for the reconstruction algorithm have been implemented, by the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) (see Fig. 3).

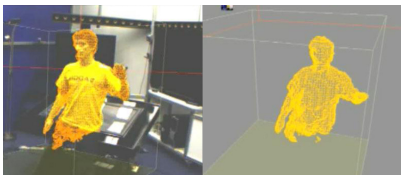

Figure 3: Output of the Marching Cubes algorithm.

A 3D real-time reconstruction based on different MC implementations in the CPU or the GPU is performed to adapt to the requirements of the system. Initially, each version has to allocate memory for the buffers that are used in the application. Then new voxels are obtained, and a new mesh is calculated in each iteration by the MC method.

3.1 CPU-based Implementations

For comparison purposes, a monothread version was implemented. Several memory usage tests were performed to provide an efficient solution, and a large block of contiguous memory was used.

3.1.1 Multithreading

Two different multithreading versions were implemented in this system. In order to deal properly with the data the volume was divided into layers that were grouped into clusters. The cluster size was defined as the number of contiguous layers to process per thread.

In the first multithread version the number of threads is the same as the number of clusters; therefore the number of threads depends on the cluster size. The higher the number of threads is, the higher the cost to keep them. Therefore an optimum value must be determined by experimentation.

In the second multithread version, the number of threads coincides with the number of processor cores at most. Each thread requests a new cluster to process if it is free, until the whole volume is processed. This request for new clusters is centralized and synchronized to ensure that only one thread requests a cluster at the same time. This implementation not only performs better than the before one, but also doesn't need any tweaking.

3.2 GPU-based Implementations

Different implementations have been carried out; one on Compute Unified Device Architecture (CUDA), based on Nvidia's version (Nvidia, 2008; Harris

et al., 2007); and other in OpenGL Shading Language (GLSL), based on Christopher's version (Dyken et al., 2008) which is known as Histopyramids. In both cases, lookup-tables are processed as textures, and their main operations (compression/expansion) can be parallelized and optimized for GPU.

In the 2D Histopyramids version, first the input data are compressed to generate histopyramids (or pyramids of data structures). With each compression operation, a new level of the pyramid is created by accumulating four elements of the previous level in just one. Finally, the top level will hold the number of output elements. Then the output data are expanded to obtain the resulting elements.

The CUDA implementation classifies the voxels with the MC identifier, and the number of generated triangles is obtained. Then, the information is compressed using the scan technique, and the triangles are generated in an expansion operation.

4 COMPARATIVE RESULTS

Several time measurements were carried out to obtain a precise comparison of the different MC implementations. It was not difficult to compare the CPU versions because the times of the MC algorithm can be calculated isolatedly. Since the MC algorithm cannot be measured isolatedly in one of the GPU versions (histopyramids) we calculated the Marching Cubes algorithm together with the normals and the data sending to the GPU in order to compare equivalent processes in the CPU and the GPU. These measurements were obtained by averaging the time of 50 iterations to obtain more accurate estimates. We also calculated the algorithms on a normal run of the pipeline (i.e. there were no iterations, just the processes followed by the rendering tasks as occurs in the usual process of the videoconference system).

4.1 Multithread Versions

The comparison of 2 multithread versions was performed. As Fig. 4 shows, version 1 was more unstable than version 2 and required finer tuning because it was more static.

Version 1 is slower because it requires more threads, changes the context frequently, and the work is not distributed equally (static option). Thus, if a thread ends soon, it cannot help others and the optimum time results are not reached. Version 2, a dynamic version, charges the system less than a static one because the same work requires a lower number of threads to be processed.

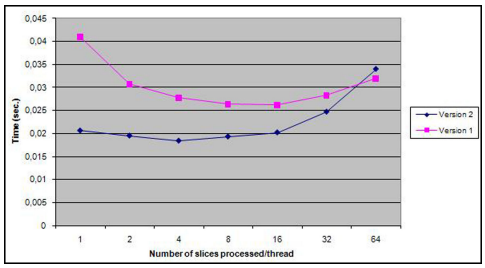

Figure 4: Multithread versions.

When comparing the most efficient multithread and monothread versions using four cores, the multithread version performs 2.8 times faster as Fig. 5 shows.

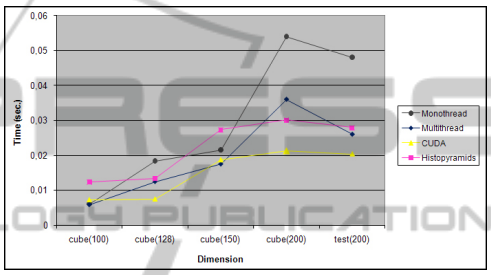

Figure 5: Multithread vs GPU versions.

4.2 Multithread vs. GPU Versions

In this section, the most efficient multithread version is compared with the 2 versions implemented in GPU.

Fig. 5 presents a breakdown of the two GPU versions in power of 2 values. This is because the GPU algorithms operate on textures or data structures that are power of 2. Therefore, the processing of data, whose size is "N", is similar to the processing of "M", where "M" is the power of 2 immediately above "N". The computational cost of "N" is shown in the expression (1) (assuming dimensions of similar size), where "N" is the number of elements of the maximum dimension and "x" is an integer.

$$O\left\{(N^3)\right\} \approx O\left\{(M^3)\right\}/2^{x-1} < N \leq M, M = 2^x \quad (1)$$

This expression explains why the GPU versions are more stable than the multithread version. The GPU computational cost is similar between a power of 2 and the following power of 2 and the multithread version increases cubically.

4.3 Real Pipeline Profiling and the Influence of the Rendering Process

The run time of the most efficient multithread version and the GPU versions on a normal run of the pipeline was calculated, and the results are shown in Fig. 5.

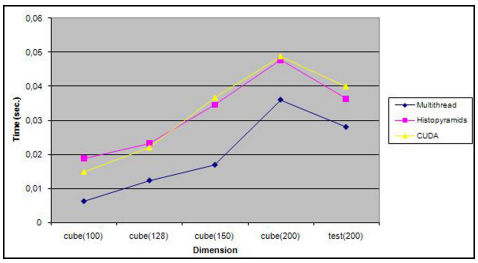

Figure 6: Multithread vs GPU versions in real pipeline.

As Fig. 6 shows, the times of the 3 versions in the normal running of the pipeline are different from the time average. Also, there is a shift in the times, which is sharper in the GPU versions. The reason is that the GPU is already saturated with the calculations of the MC and the normals and the rendering of the scene and calculation of occlusion textures. However the CPU has the same work load in the whole pipeline than in the isolated one, so there is no significant time shift in the multithread version. The most effective algorithm in the current system is the CPU based one.

The rendering system was calculated to adapt it to the load of the videoconferencing system and the available resources. In this way, an acceptable frame rate for a videoconference (always over 15 frames) was guaranteed. Exchangeable implementations were developed (using CPU-GPU) for that purpose.

4.4 Hardware Setup

The desktop where all these time results were calculated had the following characteristics: a PC with a NVIDIA Geforce 280 GTX graphics card; an Intel Core 2 Quad Q9550 CPU clocked at 2.66 GHz; and 3.25 Gb RAM of a 4.028 Mb system were usable because of the operative system used.

5 CONCLUSIONS

This comparative study about MC implementations shows how the rendering process influences the videoconferencing system. Moreover, since the algorithms implemented in a CPU are also performed on a GPU, the final results show how the algorithms are different in terms of efficiency. The performance of each option depends on the system loads, and, specifically, whether or not the CPU or the GPU is more or less saturated. To avoid a significant decline in the real-time frame rate and in the user's sense of immersion in the virtual environment, it is important not to forget that the GPU is used to represent virtual environments and the consumption of resources depends on the detail needed, as we have shown in this work.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Administration agency CDTI under the VISION project of the CENIT programme and by Ministerio de Educación y Ciencia of Spain. The authors are very grateful to all the partners of this project, especially to Telefonica I+D for coordinating the project.

REFERENCES

Chu, R., Tenedorio, D., Schulze, J., Kuwabara, S., Nakazawa, A., Takemura, H., and Lin, F. (2009). Optimized Rendering for a Three-Dimensional Videoconferencing System. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 540–546. IEEE.

Dyken, C., Ziegler, G., Theobalt, C., and Seidel, H.-P. (2008). High-speed marching cubes using histopyramids. *Comput. Graph. Forum*, 27(8):2028–2039.

Harris, M., Sengupta, S., and Owens, J. (2007). Parallel prefix sum (scan) with CUDA. *GPU Gems*, 3(39):851–876.

Kauff, P. and Schreer, O. (2002). An immersive 3D videoconferencing system using shared virtual team user environments. In Broll, W., Greenhalgh, C., and Churchill, E. F., editors, *CVE*, pages 105–112. ACM.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA. ACM.

Magnor, M. A. (2005). *Video-Based Rendering*. Ed. Sales and Customer Service Office.

Newman, T. and Yi, H. (2006). A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879.

Nvidia (2008). *NVIDIA CUDA Compute Unified Device Architecture: Programming Guide*, version 2.0 edition.

Pierleoni, P., Fioretti, F., Cancellieri, G., Di Biase, T., Pasqualini, S., and Nicolini, F. (2006). Audio Rendering System For Multimedia Applications. *Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements*, page 61.

Regenbrecht, H., Ott, C., Wagner, M., Lum, T., Kohler, P., Wilke, W., and Mueller, E. (2003). An augmented virtuality approach to 3D videoconferencing. In *ISMAR*, pages 290–291. IEEE Computer Society.

Riegel, T. and Kaup, A. (1997). Shape initialization of 3D objects in videoconference scenes. In *Proceedings of SPIE*, volume 3012, page 116.

Weik, S., Wingbermuhle, J., and Niem, W. (1998). Automatic creation of flexible antropomorphic models for 3Dvideoconferencing. In *Computer Graphics International, 1998. Proceedings*, pages 520–527.