

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/254056313>

Development of a volume rendering system using 3D texture compression techniques on general-purpose personal computers

Article · September 2011

DOI: 10.1109/ICAwST.2011.6163192

CITATION

1

READS

146

4 authors, including:



Akio Doi

Iwate Prefectural University

67 PUBLICATIONS 306 CITATIONS

[SEE PROFILE](#)



Taro Mawatari

112 PUBLICATIONS 1,913 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



CG of Nature [View project](#)



Both articles belong to our project. [View project](#)

Development of a volume rendering system using 3D texture compression techniques on general-purpose personal computers

Akio Doi, Hiroki Takahashi
Advanced Visualization Laboratory
Iwate Prefectural University
152-52 Takizawa-mura, Iwate-ken, Japan
doi{t-hiroki}@iwate-pu.ac.jp

Taro Mawatari
Faculty of Medicine
Kyushu University
6-10-1 Higashi-ku,
Fukuoka, Japan

Sachio Mega
Department of CG
JFP Corporation
2-26, Zaimoku-cho,
Morioka-shi, Japan

Abstract—In this paper, we present the development of a high-speed volume rendering system that combines 3D texture compression and parallel programming techniques for rendering multiple high-resolution 3D images obtained with medical or industrial CT. The 3D texture compression algorithm (DXT5) provides extremely high efficiency since it reduces the memory consumption to 1/4 of the original without having a negative impact on the image quality or display speed. By using this approach, it is possible to use personal computers with general-purpose graphics capabilities to display high-resolution 3D images or groups of multiple 3D images obtained with medical or industrial CT in real time.

Keywords: *Volume Rendering; 3D dimensional image; 3D Texture Compression; Parallel Processing, Medical Imaging;*

I. INTRODUCTION

In recent years, computer tomography (CT) has been extensively used in the medical field for internal measurements and examinations of the human body. CT uses radiation (X-rays) or changes in magnetic field to examine an object, and images of the internal parts of the object are constructed by using a computer. In a broad sense, CT encompasses examination methods such as magnetic resonance imaging (MRI), positron emission tomography and optical coherence tomography; in the present study, however, we use CT to refer to X-ray CT. In comparison to MRI, CT provides shorter examination times and higher spatial resolution, its market penetration is higher, and its price is lower.

CT devices can be divided into two main categories, namely, CT devices for medical use and those for industrial use. Since CT became commercially viable in the 1970s, the processing speed and resolution of medical CT devices has been greatly improved by employing helical and multi-slice CT and developing elemental technologies such as detector technologies, reconstruction technologies and dosage reduction technologies. In comparison, industrial CT devices deliver higher X-ray energy at the X-ray illumination site and utilize a fan-type beam (the intensity of X-rays in industrial CT devices is 4~5 times higher than that in medical devices). For this reason, images can be easily acquired in the form of cubic voxels with a resolution of 1024 pixels for each side. Examined objects include resins, engine blocks of automobiles

and electronic components, and due to the non-destructive nature of the examination, it can be applied to objects made of resin, wood, clay, stone and other materials. In addition, three-dimensional (3D) images obtained through CT or MRI are subjected to different types of processing, such as 1) slicing, shifting and deformation, 2) image processing such as subtraction, highlighting and segmentation, 3) merger of multiple MRI images into a single high-resolution MRI image (for example, generation of full-body MRI images), 4) display and registration of 3D images with different modality and 5) numerical simulation based on the raw volume data. In this regard, the need for simultaneous processing and simultaneous display of multiple high-resolution 3D images is constantly increasing.

In this paper, we present the development of a high-speed volume rendering system that combines 3D texture compression and parallel programming techniques for rendering multiple high-resolution 3D images obtained with medical or industrial CT. Although 3D texture compression is a technique developed in order to meet the strong demand for saving memory and improving the realism in the computer game and motion picture industries, it is also applicable to the display of volume rendering, which requires large amounts of memory. However, the time required for 3D texture compression for 3D images cannot be ignored since it is considerably longer than in the case of computer games. For this reason, we have improved the speed of 3D texture compression processing through the use of parallel processing based on multi-threaded programming. As a result, it has become possible to display multiple high-resolution 3D images simultaneously in real time by using general-purpose personal computers containing a relatively small amount of memory on the graphics processing unit (GPU).

II. PRINCIPLES AND METHODS OF DISPLAYING VOLUME RENDERING

The principles of volume rendering are as follows. Light beams (rays) are cast from a point of view into the 3D image (volume data), after which the transparency and color information is computed for voxels in the 3D image which are crossed by the rays, and the final color is determined by considering the attenuation of the rays. The mapping of pixel values from the 3D image to transparency and color

information is performed with the aid of a transfer function that defines transparency and color information with respect to a histogram of the 3D image, and this mapping is usually performed interactively by the user. This method is also referred to as the ray casting method since light beams (rays) are cast from the point of view and pass through the pixels on the screen, and integration over all rays is computed while sampling the internal parts of the voxels. The number of rays is the same as the number of pixels on the projection plane, and since the sampling must be performed in an interval that is shorter than the width of each voxel, it requires considerable time for computation.

GPUs perform operations such as geometric transformations and rasterization by using dedicated hardware and integrated circuits, and their processing speed for such specific tasks is considerably higher than that of general-purpose CPUs. In addition, the transfer speed between the GPU and the memory installed on the graphics adapter (referred to as "video memory" below in order to distinguish it from the main memory used by the CPU) is extremely high, as it is designed for real-time processing. The amount of video memory installed on a conventional graphics adapter is between several hundred megabytes and several gigabytes, and the video memory stores information such as displayed frames (frame buffer), an operation region (stencil buffer), 2D/3D texture images, geometric information (coordinates and polygon information) and optical attributes, thus providing an architecture that allows for computations to be executed independently from the CPU.

III. VOLUME RENDERING SYSTEM USING 3D TEXTURE COMPRESSION

The processing mentioned in Section 2.1, involving 1) sampling over rays, 2) transformation into color and transparency information for each sampling and 3) integration of color and transparency information over all rays, can be performed at high speed by using the texture mapping and α blending capabilities of the GPU. For a 3D image, each pixel value can be transformed into color information (R, G and B values) and transparency information (α value) by using a transfer function, and this information can be stored in the GPU memory as a texture image. The next step involves the preparation of multiple polygon surfaces that are perpendicular to the line of sight and that constitute cross-sectional surfaces of the 3D image, and texture mapping is performed on each polygon surface by using all texture images generated from the 3D image. If the line of sight changes, it is necessary to prepare new polygon surfaces that are perpendicular to the new line of sight. Please refer to [2] for information on the computation of nodes formed by crossing a 3D image with a plane.

In addition, there is a method in which arrays of polygons with texture mapping are first prepared for the X, Y and Z directions. If the point of view changes, the direction in which polygon arrays should be displayed is determined by using the

vector of the line of sight and a vector normal to the surface (the frontmost polygon array is used for the direction along the line of sight). The α blending function of the GPU, which adds the polygon surfaces, stacks semi-transparent polygons by using the transparency α of each pixel value, which is performed in hardware. As these operations are independent of the CPU, their execution speed is high.

A. Processing flow and 3D texture compression

When storing textures with color and opacity information (3D color images) in video memory, it is possible to utilize the video memory more effectively by applying image compression to the texture. This technique is standard in the computer game and motion picture industries and is presently supported in DirectX [3] and OpenGL [4, 5]. Storing textures in this manner allows for volume rendering of high-resolution 3D images on graphics adapters with a relatively small amount of video memory. The algorithm used for compressing texture images is S3TC [5-7] (made available in Microsoft DirectX 6.0 and also referred to as DXTC (DirectX Texture Compression)). As this compression algorithm utilizes single memory access and fixed-rate data compression (in the case of DXTC5, 16 32-bit pixels are converted into 128 bits of data, thus compressing the original image at a ratio of 4:1), the algorithm is suitable for compression implemented in hardware and has become widely adopted. There are five types of algorithms (DXT1 through DXT5) that differ in terms of compression ratio and image type, with DXT5 providing the highest quality; the specifications of these algorithms have also been implemented in OpenGL Ver. 3.0. Furthermore, as it was initially difficult to display textures such as marble, wood grain and mist, the concept of 3D texture that includes information about the internal parts of the rendered object has been devised [4, 5, 8]. When using OpenGL to perform 3D texture compression, it is implemented with an application program using the CPU, after which the resulting compressed image is transferred into the video memory, where it is decompressed and used at the texture mapping stage. In this process, the most time-consuming step is texture compression, which is performed by the CPU. For this reason, we have adopted parallel programming based on multi-threading in order to shorten the processing time.

B. Acceleration of 3D texture compression through multi-threaded programming

In CPUs with multiple processing units (cores), it is possible to implement parallelization based on threaded programming by executing 3D image compression on each core of the CPU in parallel. For example, the Core i series of Intel processors is subdivided into Core i7, Core i5 and Core i3 series, which contain two or four CPU cores. In addition, with the use of the Hyperthreading technology, multiple threaded processes can be executed on a single core, which is effective when performing multiple operations simultaneously. AMD has also developed similar technologies for multi-core computation. Fig. 1 shows an overview of the concepts of 3D texture compression and parallelization. Here, we assume that the

displayed 3D image is used to generate and display a 3D texture with the same resolution. The OpenGL function which transfers the compressed 3D texture into the video memory in Step 6 is "glCompressedTexImage3D()".

- Step 1.** Allocate operation Buffer 1 (4x4x4 pixels)
- Step 2.** Allocate Buffer 2, which is used for storing the results of compression (1/4 of the size of the 3D image)
- (Parallel processing start)**
- Step 3.** A 4x4x4 pixel block of the 3D image is stored in Buffer 1 by reversing the order of the pixel rows
- Step 4.** Image compression is performed for each 4x4x4 pixel block
- Step 5.** The result of compression is stored in Buffer 2 (If all blocks have been compressed, go to Step 6. If there are more blocks to process, go to Step 2 and process the next block.)
- (Parallel processing end)**
- Step 6.** After the compression is complete, perform 3D texture registration of Buffer 2 with the OpenGL function (transfer it into the video memory)

Figure 1. Processing steps for 3D texture compression

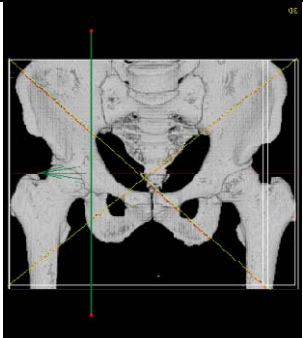
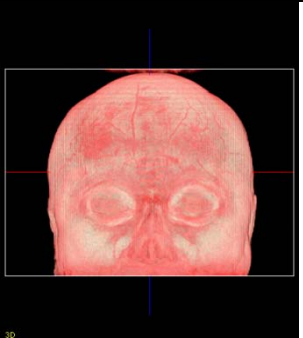
IV. EVALUATION

Table 1 presents the modality (CT, MRI), the image size and a display example of the sample images used for evaluation. For both images, the pixel pitch (pixel resolution) was 0.625 mm in the X and Y directions and 1.0 mm in the Z direction. The computer used for measurement of the computation time and for visual assessment was an HP PC (HPZ400 Workstation) with Windows 7 Professional (64-bit), Intel Xeon CPU W3680 (3.33 GHz), 12.00 GB RAM, 6 cores and an NVIDIA Quadro FX1800 graphics processor with 768 MB GDDR3 192-bit I/FSDRAM and a 400 MHz RAMDAC/pixel clock. The CPU time (in units of seconds) was obtained as an average of five measurements with a clock function (clock()).

Furthermore, when DXTC compression (S3TC compression) was applied to the sample image and the high-resolution texture, the amount of memory consumed by the used texture was 1/4 of that without image compression. In addition, aiming for practical use, the division patterns shown in Fig. 2 were prepared, and the sample image was sliced. Then the cube enclosing the 3D image (with each side perpendicular to the original image) was used to maintain and generate the texture. For this reason, in the case of division pattern "2 divisions-1" the amount of memory necessary for the texture was 2 times that for the original texture. In the case of "2 divisions-2", the necessary amount of memory was 1.5 times that for the original image.

Figs. 3 and 5 show the computation time for 3D texture compression in accordance with the number of threads for each division pattern for samples A and B, respectively. It is

clear that the computation time is decreased by increasing the number of threads. The computation time in Fig. 6 is longer than that in Fig. 3 due to the different distribution of the pixel values for the compressed 3D texture of sample B as compared to that of sample A. In particular, when there is a large number of empty areas (displayed in black), as in the case of sample A, part of the compression process is automatically skipped or simplified.

Table 1 Sample images for evaluation	
Sample A	Sample B
CT	MRI
512x512x256	512x512x248
	

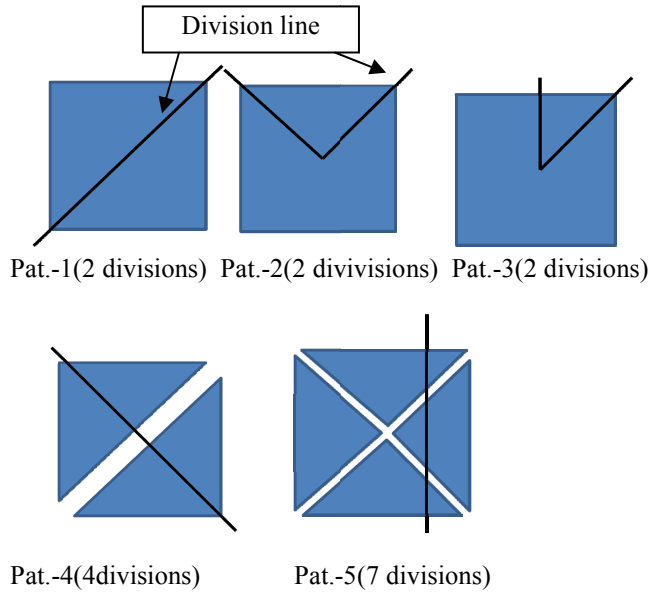


Figure 2. Division patterns (patterns 1–5)

Although the number of threads in threaded programming is also influenced by the number of CPU cores, we conducted the experiment with 1, 2, 4, 6, 8 and 10 threads. Fig. 4 presents the display results for sample A without and with 3D texture compression, respectively; Fig. 6 presents the display results for sample B without and with 3D texture compression, respectively. Although the amount of memory used was reduced to 1/4 of the original amount in the cases where 3D

texture compression was implemented, it can be seen that there are no notable changes in image quality in Figs. 4 and 6 respectively.

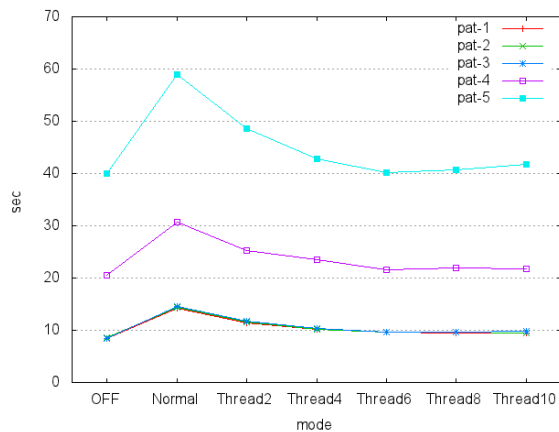


Figure 3. Computation time for sample image A

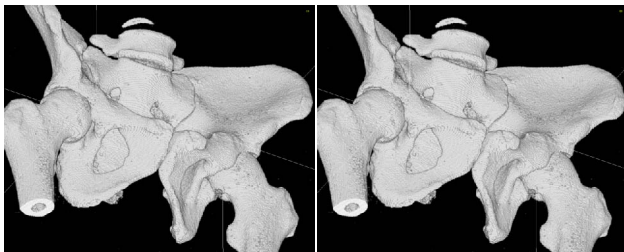


Figure 4. Volume rendering image for sample image A (right image: 3D texture compression)

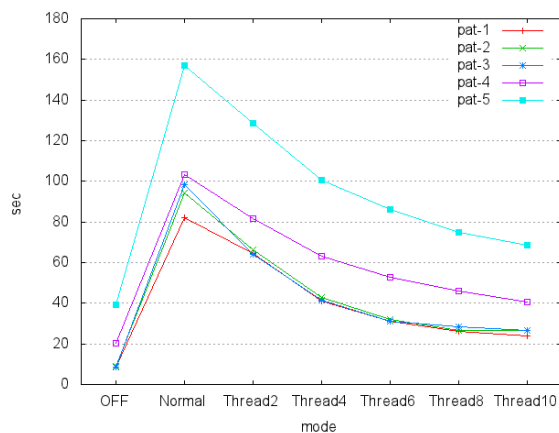


Figure 5. Computation time for sample image B

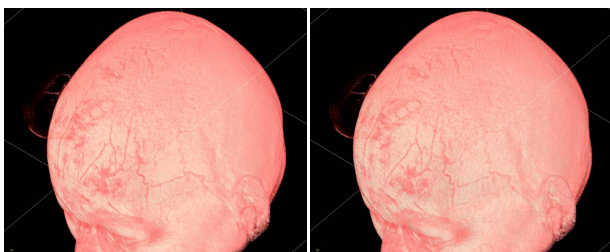


Figure 6. Volume rendering image for sample image A (left image: 3D texture compression)

V. CONCLUSION

In this paper, we presented a volume rendering system that combines 3D texture compression with parallelization programming techniques based on multi-threading. Although larger 3D textures in general entail longer processing times for texture processing, we achieved a several-fold increase in execution speed by parallelizing the algorithm, without sacrificing image quality. In particular, the effectiveness of parallelization was demonstrated to be sufficient as a result of the image areas compressed by the 3D texture compression algorithm being independent. Volume rendering in the present system is realized by utilizing texture subjected to 3D texture compression and overlapping the texture-mapped surface with the line of sight. For this reason, although the processing time for decompression of the 3D texture becomes an important factor, we did not observe any difference in display speed with respect to 3D textures with and without 3D texture compression. In this regard, the 3D texture compression algorithm (DXT5) provides extremely high efficiency since it reduces the memory consumption to 1/4 of the original without having a negative impact on the image quality or display speed. By using this approach, it is possible to use personal computers with general-purpose graphics capabilities to display high-resolution 3D images or groups of multiple 3D images obtained with medical or industrial CT in real time.

ACKNOWLEDGMENT

Part of this research was conducted with a grant-in-aid for Scientific Research (project number 20500425), the Project for the Promotion of Creative University Venture Seeds by the Japan Science and Technology Agency.

REFERENCES

- [1] I. Fujishiro, M. Okutomi, et al., "Visual Information Processing - Introduction of CG/Image Processing -, CG-ARTS Association, pp. 115-116, 2007.
- [2] G. Hoffmann, "Cube Plane Intersection", "http://www.fh-oerdingen.de/~hoffmann/cubeplane12112006.pdf", 2003.
- [3] Microsoft Corp., "DirectX Technical Information for Developer", "http://msdn.microsoft.com/ja-jp/directx/default.aspx", Microsoft DirectX Developer Center, 2010.
- [4] D. Shreiner, M. Woo, J. Neider, T. Davis, "Open GL Programming Guide Fifth Edition - The Official Guide to Learning OpenGL", Version 2, Addison-Wesley Pearson Education, 2006.
- [5] E. Lengyel, "The OpenGL EXTENSIONS GUIDE", pp. 21-86, Charles River Media, INC., 2003.
- [6] P. Brown, "EXT_texture_compression_s3tc", "http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt", 2009.
- [7] P. Brown, M. Agopian, "EXT_texture_compression_dxt1.txt", "http://www.opengl.org/registry/specs/EXT/texture_compression_dxt1.txt", 2008.
- [8] AMD Corp., "3D (Volume) texturing and Volume Texture compression", "http://developer.amd.com/gpu/radeon/archives/radeon_sdk_introduction", "ATI Radeon SDK Introduction Archive | AMD Developer Central", 2010.