High-Speed Volume Rendering in CAVEs

Yuki YAMAURA, Youhei MASADA and Akira KAGEYAMA

Graduate School of System Informatics, Kobe University, Kobe 657-8501, Japan

(Received 26 July 2013 / Accepted 12 August 2013)

A high-speed rendering method for three-dimensional animated volume rendering in a CAVE visualization environment is developed. The proposed method accelerates the standard three-dimensional texture-slicing approach to volume rendering by making use of asynchronous data transfer with the pixel buffer object of graphics processors. The method enables stereoscopic animation of volume rendering at five frames per second for scalar data of 512^3 grid points in a four-screen CAVE system.

Keywords: data visualization, virtual reality system, volume rendering

DOI: 10.1585/pfr.8.1201135

Volume rendering [1] is one of the most important visualization methods for scalar field data in the CAVE-type virtual reality (VR) environment. High frame rate is critically important in CAVE visualizations to maintain quick response to the viewer's head motion. Among several algorithms proposed for volume rendering, the three-dimensional (3D) texture-slicing method is commonly adopted because it provides the fastest rendering speed [2, 3]. In this method, a volume of data of 3D texture is generated from the target scalar data to be visualized, and 2D semitransparent textures are sliced from the volume. The slice planes are taken in real time in such a way that they are always perpendicular to the viewer's line of sight. The procedure can be summarized as follows: (i) conversion of 3D scalar data into 3D texture data with a transfer function using a prescribed RGBA color code, (ii) tansfer of the texture data from CPU RAM to GPU RAM, (iii) generation of slice planes perpendicular to the viewer's line of sight, and (iv) mapping of the textures onto the planes and drawing them in back-to-front order.

Because VR visualizations are actively applied in plasma-fusion studies [4, 5], it is important that the high frame rate of VR volume rendering be maintained as the target data size grows larger than the present typical value of $O(128^3)$–$O(256^3)$.

The purpose of this *Rapid Communications* is to report a new acceleration technique for the 3D texture slicing method that makes the best use of the current computer graphics technology.

Technical challenges for high-speed volume rendering in the CAVE environment can be divided into two categories; (1) the volume rendering process itself, and (2) the data transfer from a hard disk drive to GPU graphics memory.

As a reference, we use the VSVR (Very Simple Volume Rendering) library [6] to implement the above mentioned standard volume rendering. A clipping technique is used in VSVR to accelerate the rendering speed. In this technique, a slice plane of scalar data is always rectanglular, in spite of the fact that a straightforward slicing generates other polygons in general. We have found that volume rendering by VSVR in a CAVE application program leads to a reasonably high frame rate when the scalar data is relatively small [$\leq O(128^3)$]. Our purpose is to accelerate this volume rendering by new techniques described here.

To realize high-speed 3D animation of volume rendering in a CAVE, one has to process 3D scalar data at each time step in a sequential manner. The data conversion to 3D RGBA texture is executed by a function called Texture3D in VSVR. Texture3D is also used to transfer the data from CPU to GPU. By analyzing the processing time of a VSVR application in our CAVE system in detail, we found that the data transfer step is the hot spot with regard to time consumption.

We therefore take an alternative approach to the data transfer by making use of the Pixel Buffer Object (PBO) provided by OpenGL API. PBO enables high-speed data transfers to and from a graphic card through direct memory access (DMA). In this PBO approach, we apply the transfer function to the whole set of 4D scalar data to generate 4D texture data in the preprocessing step. We then transfer the 3D texture data at each time step for animation. The PBO approach is the first trick in our new volume rendering method. The speed is further accelerated by a type of parallel processing described in the following.

For the PBO data transfer, we generate a new thread from each display thread in the CAVE application. This separation enables us to perform asynchronous data transfer within the drawing time. In the original procedure, both data transfer and rendering steps are sequentially executed in GPU at each time step of animation. In contrast, data transfer by DMA enables us to perform asynchronous data transfer and rendering steps occurring concurrently instead

author's e-mail: kage@cs.kobe-u.ac.jp

Table 1 Frames per second in the original and proposed methods.

	256^3	512^3
Original	15.23	2.05
PBO	30.0	4.25
PBO +Async	30.0	5.00

of sequentially [7].

In CAVE applications, the OpenGL context and the animation timing must be shared by all display threads and transfer threads in this asynchronous data transfer method. Because DMA maps OpenGL memory onto virtual memory and gives a virtual address, we can access OpenGL memory through this virtual address in the transfer thread. We use the well-known C++ library `Boost.Thread` and `Boost.ASIO` to handle threads for data transfer. `Boost.ASIO` is used to synchronize threads to display the same graphic snapshot.

The effects of our new volume rendering are measured with sample data sets from a geodynamo MHD simulation [8] with 256^3 and 512^3 grid points with 12 or more temporal sequences. The typical number of texture slices is 64 and 256, respectively. Other slice numbers have been tried, but they do not affect the frame rate. We have also confirmed that the frame rate does not decrease with the number of temporal sequences for the animation. The π-CAVE system [8] is used for this experiment.

The π-CAVE is a four-screen system with a rectangular box geometry of 3 m (height) $\times$ 3 m (depth) $\times$ 7.8 m (width). It can be controlled by two different computer systems. One is a Windows-based cluster system [7 nodes $\times$ Intel XeonW3680 (12 cores), Quadro 5000, and 24 GB RAM] and the other is a Linux-based shared memory system [QuadroPlex D2 2200 $\times$ 3 and 192 GB RAM]. CAVELib is configured for 12 display threads. A wand and 10 VICON cameras are used in the tracking system. Our volume rendering program runs on both Linux and Windows systems.

The results are summarized in Table 1. Note that the frame rate for the 512^3 grid data in the original method is approximately two frames per second (FPS), which is intolerably small for the CAVE visualization environment.

In case 256^3, both methods "PBO" (with only PBO transfer implemented) and "PBO +Async" (with asynchronous transfer added) lead to an effectively maximum value of 30 FPS. Our π-CAVE system adopts the time division switching method for stereoscopic viewing, which in practice imposes a maximum on the frame rate for images. In our environment, the maximum value is set to 30 FPS.

In case 512^3, "PBO +Async" achieves to 5 FPS, a sizable increase over the 2 FPS in the original method.

We developed a high-speed volume rendering method for 3D animated visualization of scalar data in CAVEs. The key is to combine 3D data transfer by PBO with an asynchronous data transfer method. The frame rate of our new 4D volume rendering program realizes 30 FPS for 256^3 grid point data. Even for 512^3 grid point data, it attains 5 FPS, which is an acceptable value for visualizations in a CAVE. The 4D volume rendering method reported in this *Rapid Communication* is implemented as a library and has already been applied to several CAVE applications in our π-CAVE [8] system.

Acknowledgments

This work was supported by JSPS KAKENHI Grant numbers 24740125 and 23340128.

[1] T. Möller, E. Haines and N. Hoffman, *Real-Time Rendering*, A.K. Peters (Wellesley, Mass., 2008).
[2] J.P. Schulze and A.S. Forsberg, Brown University, Department of Computer Science Report, 2005.
[3] N. Ohno and A. Kageyama, Phys. Earth Planet. Inter. **163**(1-4), 305 (2007).
[4] H. Ohtani and R. Horiuchi, Plasma Fusion Res. **3**, 054 (2008).
[5] N. Ohno, H. Ohtani, D. Matsuoka and R. Horiuchi, Plasma Fusion Res. **7**, 1401001 (2012).
[6] T. Lewiner, Preprint No. MAT. 16/06, Department of Mathematics, PUC - Rio de Janeiro, Brazil, 2006.
[7] S. Venkataraman, In GPU Technology Conference 2009, volume 1102, 2009.
[8] A. Kageyama and Y. Masada, In IOP Journal of Physics: Conference Series **454**, 012077 (2013).