A Bandwidth Reduction Scheme for 3D Texture-Based Volume Rendering on Commodity Graphics Hardware

[1]Won-Jong Lee, [2]Woo-Chan Park, [3]Jung-Woo Kim, [1]Tack-Don Han, [1]Sung-Bong Yang, and [1]Francis Neelamkavil

[1]Media System Laboratory, Department of Computer Science, Yonsei University, Seoul 120-749 Korea,
{airtight, hantack}@kurene.yonsei.ac.kr
{yang, francis}@cs.yonsei.ac.kr
[2]School of Computer Engineering, Department of Internet Engineering, Sejong University, Seoul 143-747, Korea,
pwchan@sejong.ac.kr
[3]Digital Media R&D Center, Samsung Electronics, 416, Maetan-3Dong, Paldal-Gu, Suwon City 442-742, Korea,
jwoo.kim@samsung.com

Abstract. In this paper, we propose a bandwidth-effective volume rendering scheme which subdivides the volume into the sub-volumes and transmits them to the texture units in visibility order. Each sub-volume is rendered in the same manner as the original volume on the graphics hardware and the corresponding sub-image is blended in the alpha blending unit. The sub-volume oriented processing improves the cache efficiency and allows empty space skipping. Moreover, it is capable of rendering volume datasets that do not fit into the texture memory. Simulations show that the proposed scheme is effective for 3D texture-based volume rendering on commodity graphics hardware by reducing memory bandwidth up to 30 times when compared with the traditional method.

1 Introduction

Direct volume rendering is one of the popular methods to visualize volumetric data in various areas such as medicine, science, and engineering. Due to the rapid advances in graphics processing unit (GPU), 3D texture-based volume rendering on commodity graphics hardware receives great attention in these days [1, 2, 3]. Although using the commodity graphics hardware for volume rendering allows us to perform the 3D texture mapping with a low-cost, there are several disadvantages as mentioned in [4, 12, 16]. First, a large amount of data traffic causes bottlenecks in memory bandwidth both for 3D texture mapping and pixel processing. In 3D texture mapping, tri-linear interpolation may degrade the temporal locality of the texture memory accesses. Thus, we cannot achieve a satisfiable hit rate with the texture cache optimized only for 2D textures. In pixel processing, one depth test and one read/write operation must be done for each

pixel and accesses to the same screen location are separated by too many accesses to other pixels. As a result, using the commodity graphics hardware causes poor pixel cache utilization, which results in excessive frame buffer traffic in the blending operation. Second, empty space skipping is also problematic due to the slice-oriented processing order. Thus unnecessary computation for meaningless space cannot be avoided. Third, the size of a dataset that can be processed on the commodity graphics hardware is very limited. A realtime rendering of a large dataset (512^3 or larger) is infeasible on the current graphics hardware [19, 20] in general.

This paper proposes a bandwidth-effective rendering scheme by improving the cache utilization on the commodity graphics hardware. The proposed scheme subdivides the original volume into uniform sized sub-volumes logically. Each sub-volume is tested if it is an empty space or not, and then it is rendered on GPU separately. Finally each rendered sub-image is blended in the alpha blending unit for generating frame image (see Fig. 3). The advantages of the proposed rendering scheme are as follows. First, sub-volume ordered processing increases the locality of the memory access and hence improves cache efficiency, which results in a dramatic reduction of the memory bandwidth. Second, empty space testing for each sub-volume is possible in the preprocessing step so that we can avoid computation for meaningless space by skipping these empty sub-volumes. Third, a volume dataset that does not fit into texture memory can be rendered, because the original volume is subdivided into much smaller sub-volumes. A drawback of the proposed rendering scheme is that the subdivision operation generates additional vertices for each sub-volume, which causes overhead on a hardware T&L engine. However, according to the specification of a recent commodity graphic hardware [19, 20], the T&L engine has a capability of processing more than 350×10^6 vertices per second. On the other hand, in spite of the worst case of the proposed rendering scheme (the case of 16^3 sub-volume), as shown in the simulation results in Section 4, the required bandwidth for rendering at a rate of 30 frames per second is about 230×10^6 vertices per second. Thus, the additional vertices do not causes any problem in practice.

We have built a simulator to evaluate achievable performance. Memory access traces has been generated during the benchmark volume datasets of size 512^3 were rendered on this simulator. The performance has been evaluated with the trace-driven cache simulator, DineroIII [5]. Simulation results show that the proposed rendering scheme reduces memory bandwidth from 2 to 30 times compared with a non-subdivided method with the cache size varying from 16Kbytes to 128Kbytes. The rest of the paper is organized as follows. Section 2 reviews the related work, Section 3 describes the proposed sub-volume rendering scheme, Section 4 provides simulation results.

2 Related Work

The subdivision method has been adopted in a variety of researches for processing large scaled volume datasets. This method was used for load balancing

on parallel rendering machines [6] early in the research. Dedicated hardware for ray-casting [7, 8] partitioned the volume data suitable for their own memory organization. In particular, RaceEngine [7] proposed a new method to keep the visibility order of the sub-volumes. Recently, a networked cluster of PCs each of which is equipped with a fast graphics accelerator to render each sub-volume has been proposed [9]. However, parallel volume rendering is a quite expensive approach, because it requires dedicated hardware or a massive parallel rendering machine.

Nowadays the performance of commodity graphics hardware has been improved dramatically. High-level GPU-programming environment and relevant graphic APIs such as OpenGL and Direct3D are fully supported. Especially various 3D texture-based volume rendering methods using the commodity graphics hardware have been being proposed [2, 3, 4]. TriangleCaster [4] employed the extension units to divide the image plane and blend the corresponding sub-planes. However, the additional hardware units such as the composition buffer and the triangle generator should be implemented to extend the existing graphics hardware.

In order to render large scaled volume datasets on the commodity graphic hardware, the subdivision method can be applied to multi-resolution and compression. Multi-resolution volume rendering uses a spatial hierarchy to adapt the resolution to project onto the screen with hierarchical structures such as an octree. A variety of techniques have been applied to volume rendering [10, 11], since multi-resolution technique was first proposed in polygonal rendering. Moreover, volume data compression for reducing the size of such data as vector quantization [12], fractal compression [13], and wavelet transform [14, 15] utilizes hierarchical division method.

The goal of the proposed scheme is to resolve the memory bandwidth problem in 3D texture mapping by subdividing original volume into optimal sized sub-volumes to increase cache locality and by rendering in visibility order. The proposed scheme can support the rendering of the volume datasets that do not have fit into the texture memory. Also volume data compression in [15] can be easily adopted in the proposed scheme.

3 Bandwidth-Effective Sub-Volume Rendering

The proposed rendering scheme is composed of three steps; the preprocessing step for subdivision of the volume data, the rendering step including the classification and the 3D texture mapping, and the composition step for blending each corresponding sub-images. This section describes each step in detail.

3.1 The Preprocessing Step

In the preprocessing, we divide the original volume into uniform sized sub-volumes after determining the sub-volume size. For each sub-volume, vertices and texture coordinates of slices are generated with using the bound cube in [1,

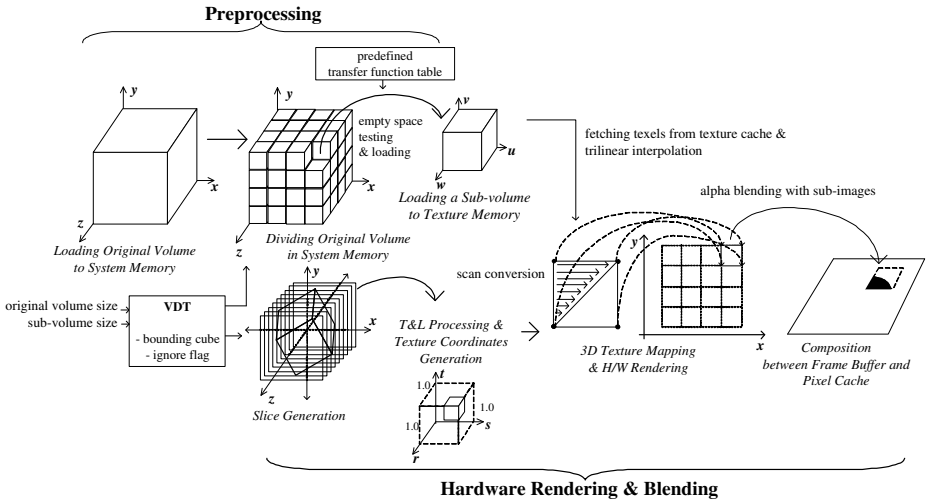

Fig. 1. Overall processing flow of the bandwidth-effective sub-volume rendering scheme

11]. We define a new data structure, called the *volume division table* (VDT) for managing the information of empty spaces and the geometry (min/max bounds of positions in the texture and the object space) of sub-volumes.

Fig. 1 illustrates the overall processing flow of the bandwidth-effective sub-volume rendering scheme. In the preprocessing step, the transfer function table is created for classification and the VDT is generated with the information about the sizes of the original volume and sub-volume. During loading the original volume to the system memory, each sub-volume can be tagged with either empty space or non-empty space after each voxel is compared with a user-defined threshold. If a sub-volume is tagged with empty, a single bit is stored into the corresponding position in the VDT. After the preprocessing step, the visibility order of each sub-volume is determined and each sub-volume is transmitted to GPU in this order which will be described in Section 3.2.

To increase cache locality, the size of a sub-volume is smaller than that of the texture cache, and the size of the slice to be mapped for this sub-volume is smaller than that of the pixel cache. This leads to minimize the bandwidth between the rendering processor and the graphics memory.

3.2 Sub-volume Rendering

In the sub-volume rendering step, each sub-volume is rendered in the same manner as the original volume is done and the corresponding sub-images are blended in the alpha blending unit to generate final frame image as shown in Fig. 3. The sub-volumes should be rendered according to the visibility order, because the volume data is not fully transparent in general and there are overlapped parts between adjacent sub-images. Whenever the viewpoint (for perspective projection) and the view direction (for parallel projection) are changed, this order

needs to be sorted again. We adopt the ordering method in [7]. In this method, the sub-volumes are ordered based on three types of classifications. First, viewpoint is parallel with a face of the volume in parallel projection. Second, looking at an edge or at a point of the volume in parallel projection, and the last type is perspective projection. In case of the parallel projection, the order is set either by height or by width based on the oriented position in the volume. In case of the perspective projection, the nearest sub-volume from the viewpoint precedes other sub-volumes.

One of the advantages of the sub-volume based rendering is that we can skip empty space easily. Because a significant portion of the volume data (more than 50% on the average in general) is empty space [18], empty space skipping is a common method to accelerate rendering for ray-casting based volume rendering. However, empty space skipping is difficult to be applied to the conventional 3D texture based volume rendering, because the whole voxels to be mapped onto slices should be fetched [16]. In contrast, the sub-volume based rendering [11, 17] has a capability of checking whether the current sub-volume is empty or not in the preprocessing step. As a result, rendering can be skipped for the empty sub-volume by referring the VDT. According to the simulation results of Section 4, the average empty space is more than 50% of the volume data. Such results can also be found in [7].

3.3 Texture and Pixel Cache Efficiency

Recent commodity graphics accelerators employ both the texture cache and pixel (color and depth) caches to reduce the bandwidth between the rendering processor and the graphics memory. But they are designed to optimize accesses only to process the polygonal data. Thus we cannot achieve satisfiable performance with them for volume rendering applications. We now describe the effective cache utilization issue of sub-volume rendering.

The tri-linear interpolation for 3D texture mapping requires accesses to the neighborhoods of voxels in the x-, y-, z-dimensions. For the volume stored in memory such as the neighboring voxels along the x-axis are adjacent, the neighboring voxels along the y-axis are one row of the x-axis voxels apart. The locality of the neighboring voxels along the z-axis is even worse. They are one row of the x-axis voxels times one column of the y-axis voxels apart. Thus a serious problem occurs in the texture cache as mentioned in [16, 18]. Due to this weak locality, access the neighboring voxel along the z-axis causes subsequent cache misses, which results in frequent replacement of cache blocks. Although some dedicated hardware accelerators for ray-casting [7, 8] are optimized for the three-dimensional data access to solve this problem, the texture cache of a commodity graphic accelerator is optimized only for 2D texture mapping. In this case, the caching is almost useless for volume rendering.

However, we can resolve this problem by subdividing and reorganizing the volume data, which make rendering of the sub-volumes more manageable. An 8bit-16^3 sub-volume consumes 4Kbytes (when 16bit-16^3 sub-volume, 8Kbytes) and a 8bit-32^3 sub-volume consumes 32Kbytes (when 16bit-32^3 sub-volume,

64Kbytes) of memory space. This chunk of data fits easily within the texture cache of a typical graphics accelerator available today. Thus the locality of memory accesses can be improved. Hence we can significantly increase cache hit rate.

There is a similar problem in the pixel cache. The size of each slice is the same as that of a slice in the view-plane in general. Each slice covers the entire screen. Moreover, the blending operation requires read/write operation for both the depth test and alpha-blending per each fragment of a slice. Thus the entire screen must be processed before a particular pixel is visited again. As a result, the locality of the frame buffer accesses is decreased and cache miss rate can be increased enormously. On the other hand, if the size of a sub-image of a sub-volume is smaller than the pixel cache size, cache hit can be guaranteed for all slices except the cache misses (compulsory miss) on the first slice. Therefore, we can improve cache utilization significantly.

4 Experimental Results

We have built a simulator to evaluate cache efficiency, memory traffic, and achievable performance. We model a standard graphics pipeline composed of the T&L engine, the Goraud shader, the 3D texture mapping unit, the blending unit, the framebuffer (color and depth buffer), and the texture memory. Memory access traces have been generated during the benchmark volume datasets are rendered on the simulator. The performance has been evaluated with the trace-driven cache simulator, DineroIII [5]. Because the pixel cache consists of the color cache and the depth cache in most of commodity graphics accelerators [19, 20], a separate simulation has been performed for each cache. All caches are configured with direct-mapped, a block size of 32Bytes. We have experimented varying the cache size from 16K to 128Kbytes.

We have used three datasets, the HeadMR, the Foot, and the Skull (see Fig. 4) from the volren web-page (http://www.volren.org). Each of these datasets has 8bit-intensity and the size of 256^3, but has been re-sampled to 512^3 (128Mbytes) to test the large volume data. Rendering is directed to a 512x512 viewport, with 100 slices, and the back-to-front blending method is used. 16^3 and 32^3 have been chosen as the size of sub-volume. Ten frames have been rendered for each case. For fair generation of memory access patterns, the volume is rotated by width and height for rendering. We describe the simulation results of the proposed rendering scheme in this section.

4.1 Texture and Pixel Cache Efficiency

Fig. 2 shows the comparison of each cache miss rate for our subdivided rendering scheme (SD) and the traditional non-subdivided rendering scheme (NSD). More than 50% of the miss rate of the texture cache with NSD was occurred due to frequent cache thrash, which means the cache could not perform its own duty. On the other hand, the miss rate was reduced sharply in the case of rendering with SD due to locality improvement. As shown in Fig. 2, all the datasets recorded

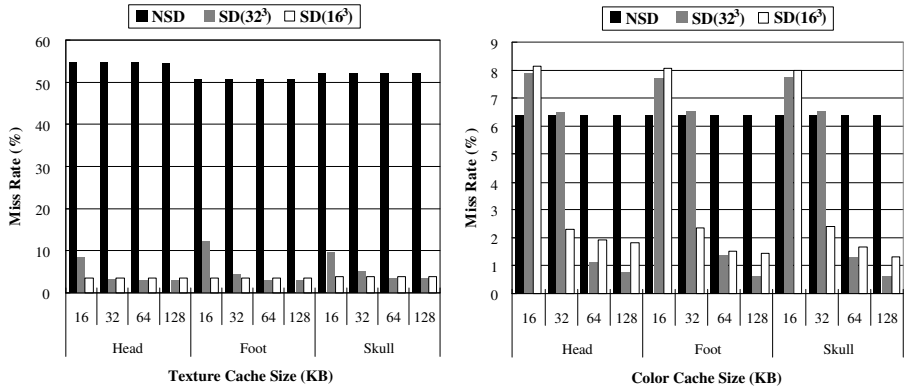

Fig. 2. Comparison of the cache miss rate for our subdivided (SD) rendering and the traditional non-subdivided (NSD) rendering

less than 5% of the miss rate of the texture cache when the cache size is larger than 32Kbytes. Even though the size is larger than 32Kbytes, the miss rate is hardly reduced for all the cases. This result shows that the miss rate had no relation with the cache size when the size is over a critical point as the case of 2D texture cache [21]. The miss rate of SD when the size of a sub-volume is 16^3 is little higher than that of SD when the size is 32^3 from the 32Kbytes for the HeadMR and the 64Kbytes for the Foot and the Skull, because more overlapped voxels are generated as the sub-volume size is decreased.

Similarly the miss rates in the pixel cache for SD have also been reduced due to high cache locality. For simplicity, we show the results for the color cache in the Fig. 2, because the number of accesses to the depth buffer for depth test and accesses to the color buffer for the blending are the same. In case of NSD, the miss rate is almost the same, over 6%, regardless of the cache size, because the size of a slice is larger than that of the pixel cache. In contrast, because the sub-image of a sub-volume is smaller than that of the pixel cache for SD, the locality is increased and the miss rate is reduced. The miss rate is sharply decreased from 64Kbytes for the 32^3 sub-volume and from 32Kbytes for the 16^3 sub-volume. When the size is 16Kbytes, on the contrary, each sub-image could not fit into the pixel cache, and hence the miss rate of SD was higher than that of NSD.

4.2 Total Bandwidth Comparison

Table 1 shows the comparison of the total bandwidth during 10 frames rendering. The total bandwidth is calculated by summing up the amount of fetched data from the texture memory due to texture cache misses, the amount of the read/write data from/to the frame buffer due to pixel cache misses, and the amount of vertices.

Table 1. The comparison of the required bandwidth between SD and NSD

Dataset	Rendering Method	Sub-Volumes	Empty-Space Skipped Sub-Volumes	Vertices	Texture Cache Size(KB)	Read Data from Texture Memory by Cache Miss(Bytes)	Pixel Cache Size(KB)	Read/Write Data from/to Framebuffer by Cache Miss(Bytes)	**Total Bandwidth needed for 30f/s(GB/s)**
HeadMR (Fig.4-a)	NSD	-	-	4,000	16	15,025,144,064	16	1,926,144,000	**47.37**
					32	15,024,721,472	32		**47.37**
					64	15,023,112,832	64		**47.36**
					128	15,016,700,416	128		**47.34**
	SD(32)	4,096	2,042	572,120	16	1,764,230,976	16	2,927,880,896	**13.16**
					32	680,485,568	32	2,418,177,920	**8.71**
					64	630,633,824	64	417,452,352	**2.98**
					128	630,477,568	128	289,171,136	**2.62**
	SD(16)	32,768	18,894	2,219,840	16	642,788,320	16	2,572,624,960	**9.19**
					32		32	726,369,856	**4.03**
					64		64	610,183,616	**3.71**
					128		128	575,279,040	**3.61**
Foot (Fig.4-b)	NSD	-	-	4,000	16	13,478,043,488	16	1,926,144,000	**43.04**
					32	13,477,901,216	32		**43.04**
					64	13,477,758,976	64		**43.04**
					128	13,477,473,600	128		**43.04**
	SD(32)	4,096	1,859	626,360	16	2,772,522,176	16	3,456,836,352	**17.47**
					32	969,392,416	32	2,935,609,984	**10.98**
					64	684,198,304	64	616,759,424	**3.7**
					128	684,198,304	128	215,540,160	**2.58**
	SD(16)	32,768	17,735	2,405,280	16	880,501,952	16	4,000,987,136	**13.86**
					32		32	1,155,852,416	**5.91**
					64		64	748,096,256	**4.78**
					128		128	710,950,016	**4.67**
Skull (Fig.4-c)	NSD	-	-	4,000	16	14,249,069,696	16	1,926,144,000	**45.2**
					32	14,249,069,696	32		**45.2**
					64	14,249,069,696	64		**45.2**
					128	14,249,069,696	128		**45.2**
	SD(32)	4,096	2,935	325,080	16	2,281,828,608	16	1,745,188,160	**11.29**
					32	578,412,608	32	1,465,006,080	**5.74**
					64	426,176,352	64	291,141,632	**2.04**
					128	426,176,352	128	143,732,992	**1.63**
	SD(16)	32,768	26,585	989,280	16	321,912,992	16	1,193,654,720	**4.33**
					32		32	358,563,008	**2.0**
					64		64	247,057,408	**1.68**
					128		128	193,965,568	**1.54**

In case of NSD, over 14Gbytes of data was transmitted from the texture memory on the average. If the frame buffer bandwidth is considered, the total required bandwidth for rendering 30 frames per second was more than 40Gbytes. Because the internal memory bandwidth of recent commodity graphics hardware [19, 20] is about 30Gbytes, it is difficult to render at a real-time frame rate.

When rendering the datasets with SD, the total required bandwidth for rendering 30 frames per second is only in the range between 1.5Gbytes and 17Gbytes. Such result was possible because SD could skip more than 50% of empty space on the average and SD improved cache hit rate on both the texture and the pixel caches. As we mentioned before, the miss rate was sharply decreased when the cache size is over the 64Kbytes for the 32^3 sub-volume and when the cache size is over the 32Kbytes for the 16^3 sub-volume. Thus, the required bandwidth for rendering 30 frames per second was from 2 to 3Gbytes in this case. The bandwidth was reduced up to 30 times, compared with NSD in the best case (when the cache size is 128Kbytes for Skull).

5 Conclusion

In this paper, we proposed a bandwidth-effective volume rendering scheme which divides the volume into uniform sized sub-volumes and transmits them to the texture mapping units in visibility order. Simulation results showed that the proposed scheme is effective for 3D texture-based volume rendering on commodity graphics hardware by reducing internal memory bandwidth substantially. Because this scheme manages the sub-volumes, it is expected to be applied to three dimensional volumetric effects such as volumetric lighting, fire, and clouds.

References

1. A.V.Gelder and K.Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In Proc. of ACM Symposium on Volume Visualization 1996, pages 23-30, 1996.
2. K.Engel, M.Kraus, and T.Ertl. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2001, pages 9-16, 2001.
3. J.Kniss, G.Kindelmann, and C.Hansen. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In Proc. of IEEE Visualization 2001, pages 255-262, 2001.
4. G.Knittel, TriangleCaster: extensions to 3D-texturing units for accelerated volume rendering. In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 1999, pages 25-34, 1999.
5. Dinero III, `http://www.cs.wisc.edu/ larus/warts.html`
6. C.T.Silva, A.E.Kaufman, and C.Pavlakos. PVR: High-Performance Volume Rendering. IEEE Computating in Science and Engineering, 3(4):18-28, 1996.
7. H.Ray and D.Silver. The Race II Engine for Real-Time Volume Rendering. In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2000, pages 129-136, 2000.
8. M.Meiner, U.Kanus, G.Wetekam, J.Hirche, A.Ehlert, W.Straer, M.Doggett, P.Forthmann, and R.Proksa. VIZARD II: A Reconfigurable Interactive Volume Rendering System. In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2002, pages 137-146, 2002.
9. J.Kniss, P.McCormick, A.McPherson, J.Ahrens, J.Painter, A.Keahey, and C.Hansen. TRex: Interactive Texture Based Volume Rendering for Extremely Large Datasets. IEEE Computer Graphics & Applications, 21(4):52-61, July 2001.
10. T.Ertl, R.Westermann, and R.Grosso. Multiresolution and Hierarchical Methods for the Visualization of Volume Data. Future Generation Computer Systems, 15(1), 31-42, 1999.
11. I.Boada, I.Navazo, and R.Scopigno. Multiresolution Volume Visualization with a Texture-Based Octree. The Visual Computer, 17(3), 185-197, 2001.
12. J. Schneider, and R. Westermann. Compression Domain Volume Rendering. In Proc. of IEEE Visualization 2003, pages 293-300, 2003.
13. W.O.Cochran, J.C.Hart, and P.J.Flynn. Fractal Volume Compression. IEEE Trans. Visualization and Computer Graphics, 313-322, December 1996.
14. K.G.Nguyen and D.Saupe. Rapid High Quality Compression of Volume Data for Visualization. Computer Graphics Forum, 20(3), 2001.

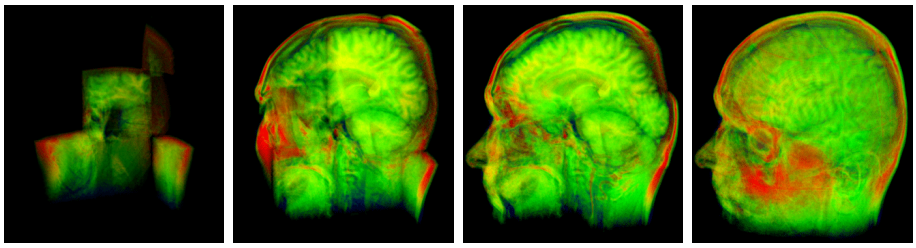

Fig. 3. Some snapshot of our rendering scheme

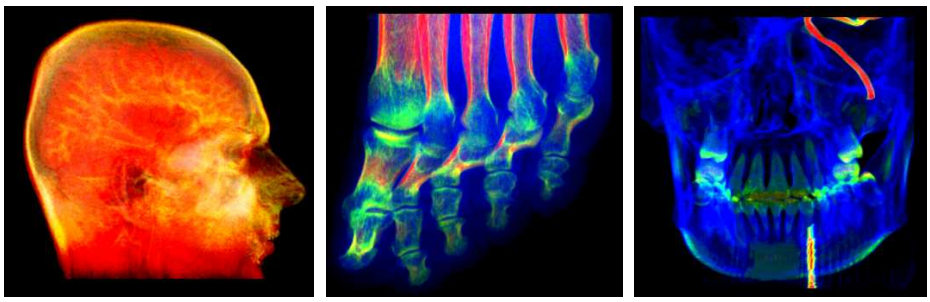

Fig. 4. Datasets rendered images with our simulator. HeadMR, Foot, Skull

15. S.Guthe, M.Wand, J.Gonser, and W.Straer. Interactive Rendering of Large Volume Data Sets. In Proc. of IEEE Visualization 2002, pages 53-60, 2002.
16. M. Hadwiger, J. M. Kniss, K. Engel, and C. Rezk-Salama. High-Quality Volume Graphics on Consumer PC Hardware. In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2002. Course Note.
17. W. Li, K. Mueller, and A. Kaufman, Empty Space Skipping and Occlusion Clipping for Texture-Based Volume Rendering, In Proc. of IEEE Visualization 2003, pages 317-324, 2003.
18. B. Lichtenbelt, R. Crane, and S. Naqvi, Introduction to Volume Rendering, Prentice Hall PTR, 1998.
19. GeForce FX 5900, `http://www.nvidia.com/page/fx_5900.html`
20. ATI's RADEON 9800 Pro, `http://mirror.ati.com/products/pc/radeon9800pro/index.html`
21. Z. S. Hakura and A. Goopta. The design and Analysis of a Cache Architecture for Texture Mapping, In Proc. of 24th International Symposium on Computer Architecture, pages 108-120, 1997.