Interactive Exploded-View Generation for Medical Volume Data

Dongsoo Kang[1], Byeong-Seok Shin[2]

[1,2]Inha University, 256 Yonghyun-Dong, Nam-Gu, Inchon, KOREA

Abstract

Exploded-view generation is illustration technique to display the volumetric data for a selected region by cut-away specific parts. These parts are displaced to reveal hidden object in detail. Exploded-view is widely used in immersive surgical simulations or 3D anatomy atlas production. Previous methods change the direction of rays in ray traversal with pre-computed 3D transformation filters, or generate images in pixel shader by dividing the entire volume of data into several parts and applying rotation and translation transformation to individual blocks. Even though a renderer can generate the resulting image in real-time, the entire procedure may not be done in real-time since it takes a long time to classify interesting parts from the input volume. In addition, a 3D filter kernel must be created for deformation in the preprocessing step whenever the opacity transfer function changes. We show that divide the image after determining the interesting parts with a block-based structure, and to render a scene with the transformed segments. The entire process is performed in real-time with Computer Unified Device Architecture (CUDA). We present the volume ray-casting for exploded view which can interactive rendering.

Keyword : Exploded-view, GPU-based volume rendering, CUDA, Real-time rendering, Block-based re-rendering

---

[1] Dongsoo Kang
gagalchi@msn.com
Address. Dept. of Computer Science & Information Engineering, Inha Univ. 253 Yonghyun-Dong, Nam-Gu, Incheon, KOREA
Tel. +82-32-860-7452

[2] Byeong-Seok Shin
bsshin@inha.ac.kr
Address. Dept. of Computer Science & Information Engineering, Inha Univ. 253 Yonghyun-Dong, Nam-Gu, Incheon, KOREA
Tel. +82-32-860-7452

1. Introduction

In recent years, generating a cut-away display to observe specific regions of volume data has become an interesting research issue. There are two major types of methods: volume deformation and exploded-view generation.

Volume deformation creates a deformed image by modifying the direction of rays in the ray traversal step [1, 2]. A 3D deformation filter, which has the same size of the original volume data, should be made in the preprocessing stage to change the ray direction. Every voxel in the 3D filter contains a vector that deforms each ray in a specific direction. However, it is very hard to manipulate a large volume dataset with the limited memory of a graphic hardware, since it should create a 3D volume texture having the same size as the original volume data. Also, it is difficult to support various types of deformation in real-time, since it takes much time to make the 3D filter. If we make several deformation kernels in the pre-processing step, we must update a kernel texture whenever deformation condition is changed by user. It causes performance degradation.

Exploded view is an illustration technique in which an object is partitioned into several segments. It moves the segments in specific directions to specific position and renders those segments with GPU [3, 4]. However, it also takes a lot of time to segment and to place those segments as the number of segments increases. Therefore, it is hard to perform the entire procedure in real time. To solve this problem, we propose real-time volume decomposition and rendering method that can divide and replace volume segments and generate a cut-away view with rotation and translation transformation. We also exploit a block-based volume subdivision method using a fragment shader. Each sub-volume has an arbitrary non-rectilinear shape such as block. It improves rendering speed since it does not consider ray-casting for complicatedly arranged volumes. Our method can easily calculate the order of the incident for sub-volumes. Because, the exploded view is purpose to just separate interesting objects. It basically is not allows that volume subdivision is overlapped.

In Section 2, an overview of related researches is presented. We explain block-based interactive exploded-view generation in detail and show experimental results. Finally, we conclude our work.

2. Related Works

Cutting away some parts of the volume to reveal internal structures is quite common in volume visualization. Almost all volume renderer features simple clipping operation. Wang introduces volume sculpting as a flexible approach for exploring volume data [5]. Konrad-Verse uses a deformable cutting plane for virtual resection [6]. The work of Dietrich consists of clipping tools for the examination of medical volume data [7]. Owada presents a system for modeling and illustrating volumetric objects using artificial cutting textures based on surface models [8]. Chen introduces the concept of spatial transfer functions as a theoretical technique for modeling deformations in

volumetric data sets [9]. Islam extends this work using discontinuities to split volume data [10]. The interactive system presented by Singh allows manual editing of volume deformations based on a skeleton [11]. They extend Islam's work by introducing selective rendering of components for improved visualization. Exploded-views have been investigated in the context of architectural visualization by Niedauer [12]. McGuffin et al. are the first to thoroughly investigate the use of exploded-views for volume visualization [13]. Their approach features several widgets for the interactive browsing of volume data partitioned into several layers.

Most of the conventional CPU-based multi-volume rendering methods were an off-line rendering due to limited computing power [14]. However, it guarantees real-time rendering through cache-coherence, multi-threading etc [15, 16, 17]. Also, most of the recent volume rendering methods are developed for GPU since it supports branch instructions and powerful float point operation. Especially, the depth peeling method in multi-volume rendering can make a result as a complex scene implemented with a GPU-based method [18]. Recently multi-volume rendering has been combined with depth and dynamic shader generation into a very efficient framework [19].

3. Interactive Exploded-view generation

For real-time processing, we propose a block-based exploded-view generation method that segments interesting regions without preprocessing and performs cut-away operation. To implement fast GPU-based ray-casting algorithm for exploded view, we use CUDA with parallel mechanism [20]. It segments a volume data in real-time. CUDA is a parallel processing architecture using a GPU and it can process as many data sets as a user wants, using millions of concurrent threads. In order to generate an exploded view we need to specify interesting objects. So, we exploit a block-based rendering method for efficient volume cut-away. Each block has an arbitrary non-rectilinear shape and can be created and merged each other. Furthermore, we support empty space skipping and early ray termination. It assumes that we have geometry enclosing the visible volume under the current transfer function for interesting object or none.

The overall procedure of proposed method is shown in figure 1. It transfers volume data to a GPU's memory to perform ray-casting for segmented sub-volumes. Then it creates geometry of sub-volumes according to the size of each sub-volume and makes 3D textures for empty-space leaping and early ray termination. A pixel shader stores images of all sub-volumes to cast rays into the frame buffer. The order of ray-casting is determined by the depth values of the center of blocks bounding those sub-volumes. It produces images according to the ordering priority, and those images are written in a 2D texture back-to-front order. Our method provides the translation and rotation of sub-volumes and cut-away operations. Also, it is possible to perform real-time volume re-segmentation as much as a user wants.

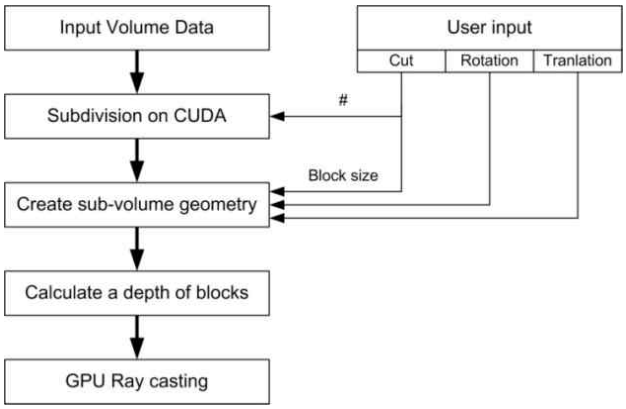

Fig. 1. Rendering procedure of our method

3.1 Volume cutting on CUDA

The CUDA hardware model has a set of SIMD multiprocessors. Each processing element has a small amount of local shared memory, a constant cache, a texture cache and a set of processors. At any given clock, every processor in the multiprocessor executes the same instruction. For example, the NVIDIA Geforce GTX260 is comprised of 24 multiprocessors. Each processing element has 8 streaming processors for a total of 128 processors. In CUDA, the GPU is a computing device that can execute a large number of threads concurrently. The batch of threads is organized as a grid of thread blocks. A thread block is a group of threads that synchronize and efficiently share data thorough local shared memory.

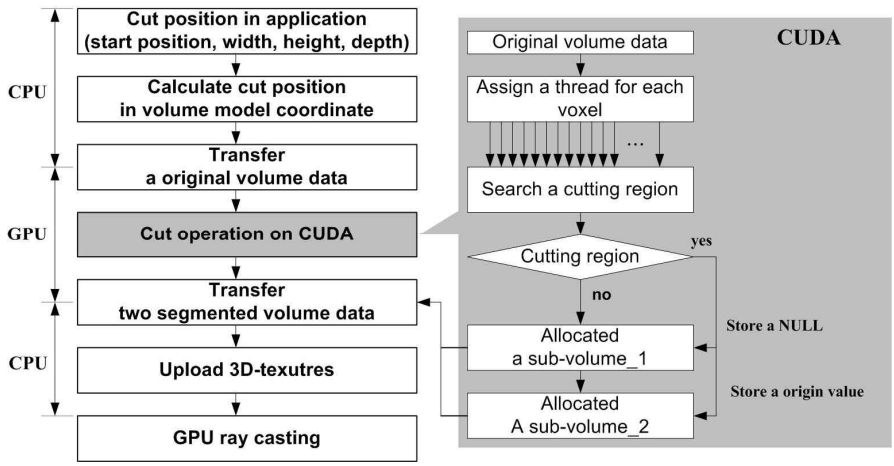

Fig. 2. Procedure of region cutting on CUDA

We implement all steps in CUDA and therefore have access to all intermediate results of the pipeline, which are kept in shared memory wherever possible for maximum performance. In a CUDA processor, input data from the CPU consists of four cutting parameters representing the cutting condition – start position such as width, height, depth to divide original volume data. The cut-away procedure is as

follows. Firstly, we input the start position of volume data in modeling coordinates. We get four cut-away parameters through a user interface. These are transferred to a GPU with original volume data. Next, we divide an original volume with CUDA (Fig.2). To store a segmented sub-volume, we have to allocate memory containing the interesting part. Then, we use as many threads as the number of voxels of all volume data for parallel processing, and each thread is executed when a specific thread is allocated in the voxel of the interesting part. To identify an interesting object, we apply a plane equation and the coverage mask for identifying a volume boundary. In an interesting part, none interesting parts set to '0'. It is performed on CUDA using simply the coverage make algorithm. That is, if a voxel is not included in an interesting part, it is stored as '0' in the relevant sub-volume data memory. The interesting part is actually visible part from the cropped area. Then, if specific voxel obtain a masking value of '0', this voxel value is changed to '0'. Accordingly, '0' voxels are not considered in rendering. In this paper, we perform cut-away operation for arbitrary cropped area which includes any straight line or plane.

3.2 Force-feedback configuration

Force-feedback arranges sub-volumes such as the nodes of a graph by translating the layout requirements specified by users into physical forces. A simple setup exploits repulsive forces between all sub-volumes and attractive forces between sub-volumes connected with an edge. The corresponding node positions constitute the layout.

We want to arrange three-dimensional objects in such a way that they reveal another object, but with as little displacement as possible. We want to achieve a steady state where the attractive forces and the repulsive forces are in equilibrium. For this reason we need to define a number of forces based on our requirements.

We generate a force that drives the specific part away from the selected object. Our method is to generate force field which describes the characteristics of our selected object. The selected object affects every other sub-volume with distance-based force. It is repulsive force value after it is divided by the value of the distance from center point of original node to center point of current node. We assume that each sub-volume is a node in octree hierarchy. Each node stores a center position. Each node generates direction vector from center position of upper level node to current node. After normalizing with the distance, we set the value to normal vector of each node. This vector is a criterion to move each sub-volume. It is a repulsive force. Also, we can control using coefficient C_r that the sub-volume move how far from the selected object. The C_r is weight value for direction vector.

$$\text{Repulsive force } (F_r) = \frac{C_r}{\| r \|} \times \frac{r}{\| r \|} \tag{1}$$

In interactive system, we must consider that the user want to rotate or zoom in/out. For this reason we

exploit view force which attempts to arrange sub-volumes. In general, previous illustration-based applications have a static view mostly. The application with static view does not need to consider the force-feedback. Although static view is simple to implement, it is not suitable to analyze complicated objects. Therefore, it is difficult to distribute the objects for detailed observation since preview applications engage the fixed view. To solve this problem, we implemented application that is appropriate to dynamic view. So, we consider an interactive force-feedback along observing direction since it is driven by user. This technique can solve the problem that specific objects occlude earlier object in the current view. For each sub-volume, we calculate a direction vector using distance from view plane and angle. This vector is used to calculate viewing force of sub-volumes. It is generated by center of sub-volume and view point in 3D coordinates. Then, it is normalized. Also, this normal vector is a criterion of coefficients (C_v) with distance. We can move the object to not occluded viewing position with the C_v. The smaller the C_v, the farther from initial location along viewing direction.

$$\text{Viewing force } (F_v) = \frac{C_v}{\| r \|} \times \frac{r}{\| r \|} \tag{2}$$

The scaling factor of repulsive force and viewing force are scaled with the degree-of force-feedback parameter. Additionally, the total force vector can be modified such as user's control or increase spacing.

3.2 Force-feedback configuration

The conventional multi-volume rendering method consists of three steps: (1) depth of the target object is calculated, (2) segmented volumes are assigned a drawing order and sorted, (3) ray-casting is performed in the sorted order immediately. Exploded-view generation does not consider the overlap of each volume since it aims to generate a detail unfolding view of the input volume. Therefore, we present a re-rendering method to visualize sub-volumes avoiding unnecessary calculations and accelerate rendering performance. Basically, our method uses a GPU ray-casting method [10]. The entire processor is as follows. First, we calculate several parameters in CUDA such as width, height, and depth after segmentation of the original volume data. We transfer these parameters to a vertex shader. These parameters are used to scale a unit-block of a single voxel. Through a scaling operation, each scaled block is used as proxy geometry to perform ray-casting. Generated proxy geometry is useful to perform empty space skipping and early ray termination in a fragment shader. It speeds up rendering performance, since we can control the number of compositing operations. Then the pixel shader applies ray-casting to store images of all sub-volumes into the frame buffer. The order of ray-casting is determined by the depth values and center position of blocks bounding sub-volumes. We calculate a distance from the center of the sub-volume to the camera position in real-time. This value is the priority number of several sub-volumes. We produce results for each sub-volume according to

the priority, and those images are written in a 2D texture with back-to-front order. Our method provides translation and rotation of sub-volumes and cut-away operations. Also, it is possible to re-segment the volume in real time as much as the user wants.

4. Experimental Results

The resulting method allows a user to easily configure a number of sub-volumes within a volume dataset. Our method is implemented on a NVIDIA GTX 260 with 512 MB video memory. We used DirectX 9.0 and CUDA 2.0. The viewport size is 1024×1024. We used the Big Head volume dataset, specific ultrasound datasets. We evaluated the speed of the cut away operation and rendering.
Table 1 shows the rendering speed of our CUDA-based exploded view generation method. We test for several cases that had different numbers of sub-volumes and resolutions (Table 1).

Table 1. Data type and comparison of rendering performance

Case #	Resolution of segmented volumes	Sub-volume #	original volume	Cutting + Data transfer	Rendering time
Case 1	128×128×128	4	O	71*ms*	44*fps*
Case 2	128×128×128 64×256×56	8	O	185*ms*	26*fps*
Case 3	64×256×256 128×256×256	3	X	50*ms*	68*fps*
Case 4	256×256×256	4	X	75*ms*	63*fps*

Figure 3 depict the resulting images of the proposed exploded-view generation method with the Big Head dataset. In case 1, the original volume is divided into four sub-volumes. In case 2, the original volume is divided into eight pieces. Cut away and time for data transfer to the GPU is 71*ms*. Overall rendering time is about 23*ms*. The experiment is classified into four cases. Two render sub-volumes with original volume data (case 1, 2) and two sub-volumes without original volume data (case 3, 4). We can show that the cutting operation and data transfer time is quite small (Table 2). It is a set of common public–domain data for testing medical imaging. And we measure the overhead of our exploded-view rendering. The results of this comparison are given in table 2. The data resolutions are 256×256×256. We can see that our approach have the frame rate drops linearly according to the number of sub-volumes and performance for a single volume object is almost identical.

Table 2. Performance comparison as number of sub-volume

number of sub-volume	frame/second	number of sub volumes	frame/ second
1	63	8	35
2	58	16	26
4	44	32	23

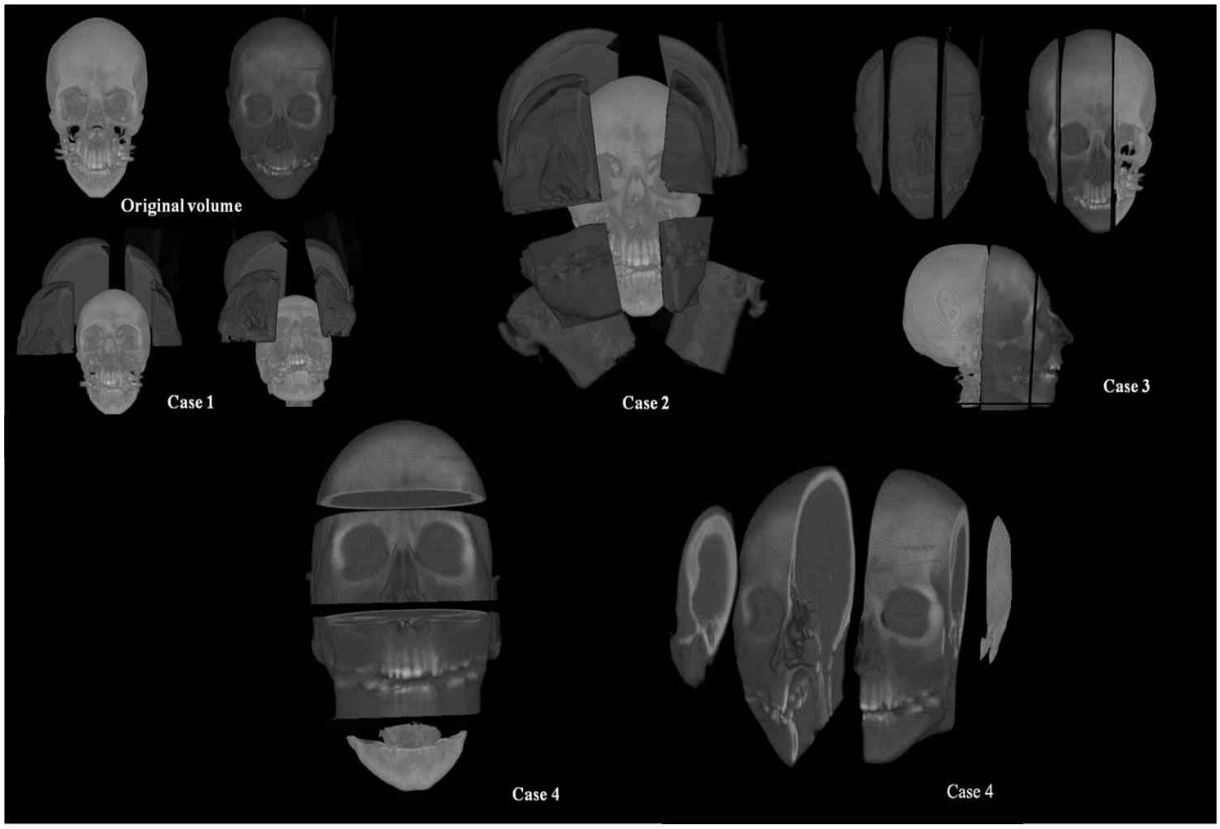

Fig. 3. Result image using our exploded -view; Case 1 is a result image produced by our method using 4 sub-volumes and original volume data. Case 2 is an image produced by our method using 8 sub-volumes and original volume data. Case 3 is resulting image of our method using just 3 sub-volumes. And Case 4 is an image produced by our method using just 4 sub-volumes.

5. Conclusions

We presented an exploded-view generation method that can be used to unfold the volume data in real-time using a block-based segmentation method that can operate on CUDA. Previous exploded-view generation methods cannot be performed in real-time due to view-dependent volume deformation and long processing time.

Our method supports real-time processing since it does not require preprocessing and provides high-quality images, because it exploits GPU-based volume ray-casting. At present, it cannot yet support bending and peeling types of volume deformation. In order to solve the problem, we will focus on volume rendering methods using a vertex shader. Finally, we will devise a hybrid method using both the currently proposed method and the volume rendering method operating on a vertex shader

Reference

[1] F. Schulze, K. Bühler, and M. Hadwiger, “Interactive deformation and visualization of large volume datasets,” *International Conference on Computer Graphics Theory and Applications*, vol.39, no.46, pp.39-46, 2007

[2] C. D. Correa, D. Silver, and M. Chen, “Volume deformation via scattered data interpolation,” *Eurographics/ IEEE VGTC Workshop on Volume Graphics*, pp.91–98, 2007

[3] J. Diepstraten, D. Weiskopf, and T. Ertl, “Interactive cutaway illustrations,” *Computer Graphics Forum*, vol.22, no.3, pp.523–532, 2003

[4] J. Krüger, R. Westermann, “Acceleration techniques for GPU-based volume rendering,” *In Proceedings of IEEE Visualization*, pp.287–292, 2003

[5] S. W. Wang, A. E. Kaufman, “Volume sculpting,” *In Proceedings of the Symposium on Interactive 3D Graphics*, pp.151–156, 1995

[6] O. Konrad-Verse, B. Preim, and A. Littmann, “Virtual resection with a deformable cutting plane,” *In Proceedings of Simulation und Visualisierung*, pp.203–214, 2004

[7] C. A. Dietrich, L. P. Nedel, S. D. Olabarriaga, J. L. D. Comba, D. J. Zanchet, A. M. M. da Silva, and E. F. de Souza Montero, “Real-time interactive visualization and manipulation of the volumetric data using GPU-based methods,” *In Proceedings of Medical Imaging*, pp.181–192, 2004

[8] S. Owada, F. Nielsen, K. Nakazawa, and T. Igarashi, “A sketching interface for modeling the internal structures of 3D shapes,” *In Proceedings of the International Symposium on Smart Graphics*, pp.49–57, 2003

[9] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea, “Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation,” *In Proceedings of the International Workshop on Volume Graphics*, pp.35–44, 2003

[10] S. Islam, S. Dipankar, D. Silver, and M. Chen, “Spatial and temporal splitting of scalar fields in volume graphics,” *In Proceedings of the IEEE Symposium on Volume Visualization and Graphics*, pp.87–94, 2004

[11] V. Singh, D. Silver, and N. Cornea, “Real-time volume manipulation,” *In Proceedings of the International Workshop on Volume Graphics*, pp.45–51, 2003

[12] C. Niederauer, M. Houston, M. Agrawala, and G. Humphreys, “Noninvasive interactive visualization of dynamic architectural environments,” *In Proceedings of the Symposium on Interactive 3D Graphics*, pp.55–58, 2003

[13] G. T. Herman, H. K. Liu, “Three-dimensional display of human organs from computed tomograms,” *Computer Graphics and Image Processing*, vol.9, no.1, pp.1–21, 1979

[14] W. Cai, “Data Intermixing and Multi-volume Rendering,” *Computer Graphics Forum*, vol.18, no.3, 2001

[15] S. Lim, D. Lee, B. S. Shin, “An image division approach for volume ray casting in multi-threading environment”, *Mutimedia Tools and Applications*, 2011

[16] W. Hsu, “Segmented ray casting for data parallel volume graphics,” *Proceeding of the Graphics Interface*, pp 70-77, 2003

[17] S. Lim, B. S. Shin, “An image-ordered parallel volume ray casting using frame coherence,” *Information Science and Application*, 2011

[18] S. Grimm, E. Groller, “Real-time mono- and multi-volume rendering of large medical dataset on standard PC hardware” Ph.D thesis, Vienna University, 2005

[19] D. Ruijters, A. Vilanova, “Optimizing GPU volume rendering”, *Journal of WSCG*, 2006

[20] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, v.2.0, NVIDIA, 2008