CUDA Based Interactive Volume Rendering of 3D Medical Data

Piyush Kumar and Anupam Agrawal

Information Technology Department
Indian Institute of Information Technology, Allahabad
{rs109,anupam}@iiita.ac.in

Abstract. Improving the image quality and the rendering speed have always been a challenge to the programmers involved in large scale volume rendering especially in the field of medical image processing. The paper aims to perform volume rendering using the GPU, in which, with its massively parallel capability has the potential to revolutionize this field. The final results would allow the doctors to diagnose and analyze the 2D CT-scan data using three dimensional visualization techniques. The system is used in two types of data, one is human abdomen (45 MB) and colon_phantom8 (300MB) volume data. Further, the use of CUDA framework, a low learning curve technology, for such purpose would greatly reduce the cost involved in CT scan analysis; hence bring it to the common masses. The volume rendering has been done on Nvidia Tesla C1060 card and its performance has also been benchmarked.

1 Introduction

Since ancient times vision has been an important part of how a human perceives the environment around him. Vision is responsible for providing inputs upon which necessary action is performed. Then came the advent of television when the world around was projected on a screen having only two dimensions. However, long before that various other forms of projection on two dimensions have been in use. A point in this case is the painting by the ancient artists which have depth in them. Soon, it was realized that if the data by other senses can also be projected in a form which is visible, then, new information can be extracted out of that.

The term visualization means the construction of a visual image in the mind. Scientific visualization is an important part of 3D computer graphics. Volume data is used for visualizing purpose. A typical 3D volumetric data set is a group of 2D slice images acquired by CT, MRI machines or 3D scanners. 3D MRI or CT data reconstruction is a complicated and challenging problem with high computing density and also a time consuming process. Volume rendering is a method which is used to visualize this type of dataset. This is also called direct volume rendering. It is a set of techniques used to display a 2D projection of a 3D discretely sampled data set.

The complicated and challenging problems are now easier due to the fast development of the parallelism technique in parallel computing. Especially thankful to multi-core CPU and CUDA on GPU making fast 3D reconstruction practically possible [1]. The 3D volume data would be reconstructed and rendered in just a few

A. Agrawal et al. (Eds.): IITM 2013, CCIS 276, pp. 123–132, 2013.

seconds after scanning by the scanner. This is very helpful in the field of medical operations/ surgeries and online inspection. Here, it can be easier to run complex filtering and segmentation based techniques in real-time. With the development of APIs just like CUDA and OpenCL, the flexibility for scientific visualization programming has achieved new heights [2].

This paper will go through some literature review of the volume rendering in scientific visualization in next section. The third section explains the proposed methodology and its implementation in CUDA on GPU. A few interactive result snapshots and performance analysis results are summarized in fourth section. Finally we have concluded these entire things with the future work in the end of the fifth section.

2 Literature Review

Ray casting algorithm for volume rendering was first introduced by Kajiya [3]. The ray casting algorithm is an important approach for volume visualization. This has mainly come through two phase process. One is CPU-based and the other is GPU-based. Here, it would explore the method with the help of graphical model and the CUDA model [1].

During rendering, optical properties are accumulated along each viewing ray to form an image of the data. Here, an optical model was used to map data values to optical properties. The role of optical model is to describe that how particles in the volume interact with light. Optical parameters are specified by the data values directly, or they are computed from applying one or more transfer functions to the data [4]. The transfer functions can either be applied before the interpolation from the surrounding scalar values (pre-classification) and interpolating the resulting RGBA values, or after the interpolation of post-classification [4]. The most commonly used algorithms are summarized here which were: splatting, shear-warp, texture mapping, and ray casting under Direct Volume Rendering. But the ray casting is the most popular method for volume rendering [4].

Generally the same process is used to do this for all rendering techniques as [4]:

- **Splatting method:** It is a technique, where every volume element is splatted on the projection plane in a systematically back to front order. These splats are rendered with various attributes depending on the volume density and the transfer function.
- **Shear-Warp method:** It is a factorization technique, where the viewing transformation is used. The faces of the volume become axis aligned with image plane and voxels (volume element) to pixels scale is fixed. If all slices have been rendered, the buffer is warped into the desired orientation.
- **Texture mapping:** This approach is based on blending of textured slices. Volume is stored on the GPU in three sets of 2D textures. One set of 2D texture for each dimension. These textures are rendered using alpha blending. The second possibility is to store volume in one 3D texture and then render polygons using alpha blending.

- **Volume Ray Casting:** It is a basic technique for volume visualization. In our research work, initially we will concentrate on this technique of direct volume rendering for a large scale dataset. The GPU-based volume ray-casting technique provides high-quality result at interactive frame rates [3].

2.1 Volume Ray Casting Approach

General ray casting is based on the idea of shooting rays, which originate in the user's eye, through an object, thus computing the colours of the pixels passed by the rays. For ray casting through volumetric data, each ray is traversed from the location of the eye until it leaves the dataset [4] [13].

Direct volume rendering methods are used to generate 3D volumetric data visualization without extracting the surface geometry from the data [12]. The basic idea is to accumulate the optical properties such as color and opacity as we travel along the ray emanating from each pixel of the screen.

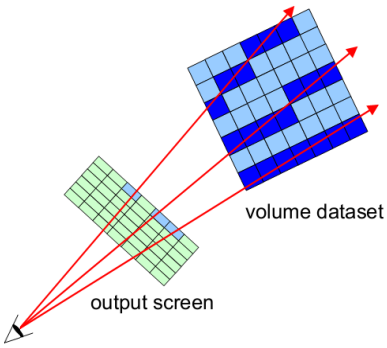

Fig. 1. Casting Rays through an Object [4]

Fig.1. illustrates how the rays are cast from the eye through the screen and the object: dark blue voxels in the volume object are the voxels that are traversed by the algorithm. The light navy blue pixels on the output screen represent the pixels that are involved. Images are created by sampling the volume along all viewing rays and accumulating the resulting optical properties [6].

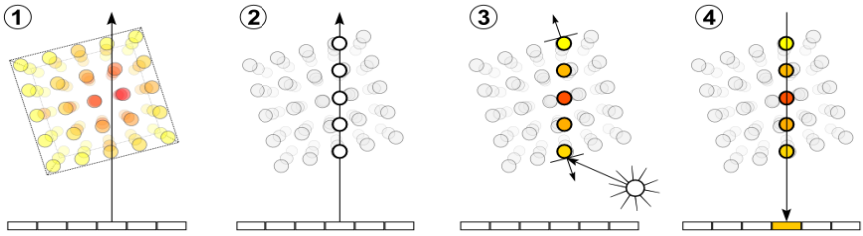

Fig. 2. Four basic steps of volume ray casting: (1) Ray Casting (2) Sampling (3) Shading (4) Compositing Rays [5]

The basic four steps are used in the method of volume ray casting algorithm which are shown in above Fig.2. which are: ray casting , sampling, shading, and compositing rays.

2.2 CUDA Based Architecture

The hardware has been designed to support lightweight driver and runtime layers, resulting in high performance. The structure of the CUDA (Compute Unified Device Architecture) device has described in the form of threads, blocks, and grid [9]. This is shown in Fig.3.

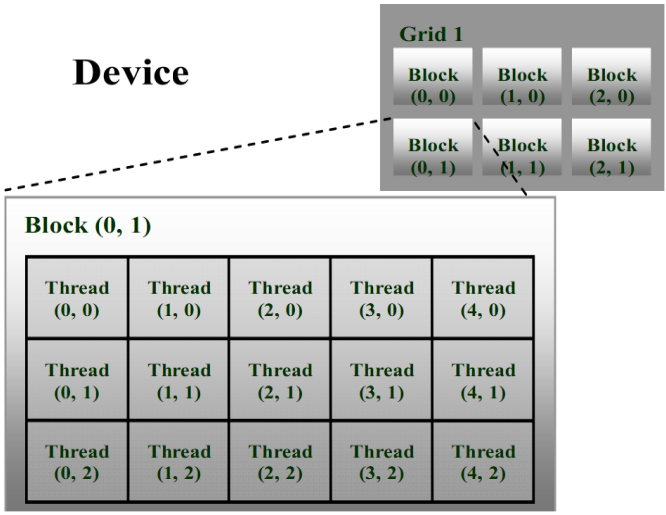

Fig. 3. Showing the distribution threads in CUDA [9] [14]

In the above figure the minimal execution unit is thread. The GPU is a device which computes several numbers of threads at a time. The CUDA API defines each thread according to its thread id and the batch of threads is organized as a block. Each block has its own block id. The batch of blocks is then organized as a grid [14]. One thread block can synchronize and efficiently shared through shared memory.

Systematically Processing flow on CUDA, as shown in Fig.4., has following steps:

1. Copy data from main memory to GPU memory.
2. CPU instructs the process to GPU.
3. GPU execute parallel in each core.
4. Copy the result from GPU memory to main memory.

The implementation of the ray casting architectures of volume visualization [4] [8] and ray casting method using CUDA can be observed from [7] [10] [11]. Changgong et. al [4] proposed a volume ray casting method which performs sampling within a ray segment using B-spline.

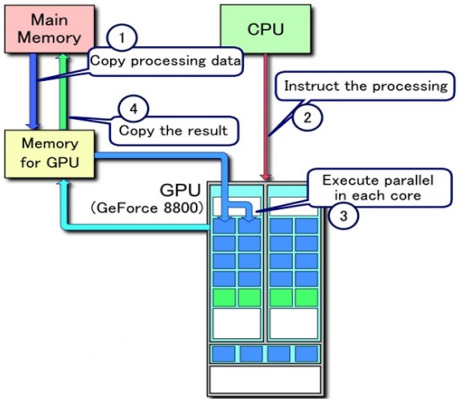

Fig. 4. Processing Flow on CUDA [9]

3 Proposed Methodology

Direct volume rendering or volume rendering method is used to generate 3D volume data visualization without extracting the surface geometry from the sampled data.

The basic idea is to accumulate the optical properties such as color and opacity as we travel along the ray emanating from each pixel of the screen. We have used a form of direct rendering approach called the Ray casting. It's a form of image rendering approach based on volume rendering. The whole procedure is defined as a block

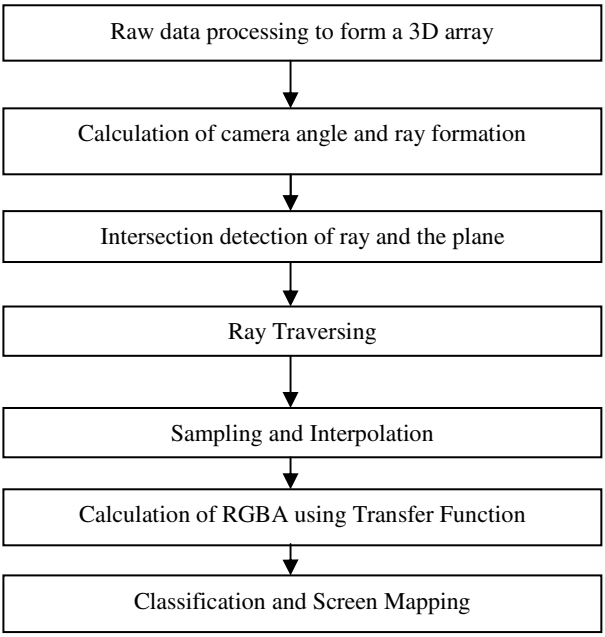

Fig. 5. Block Diagram of Proposed Methodology

diagram as shown in Fig.5. Here, first take the raw data for processing to form a 3D array. Raw data is format in which two or multiple dimensions. The volume rendering would be done on Nvidia Tesla C1060 card.

3.1 The Graphics Model

In the graphical process model the GPU rendering is a fixed pipeline mode. The approach would first take a volume data and evaluate the vertices. Then rasterization is applied for the segmenting the data. Rasterization is used to defining the data in a sequential grid form. After segmentation then check the frame cache for the segmented voxels. This is shown in Fig.6.

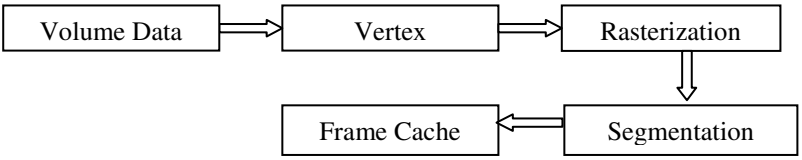

Fig. 6. Block Diagram of Rendering Pipeline

3.2 Ray Casting Implementation Using CUDA

Volume rendering is used for describing the visualization of 3D data. This visualizes the sampled functions of all three spatial dimensions by evaluating of the 2D projections. The volume rendering has been performed using two type of datasets of size 300 MB (colon_phantom8) and 45 MB (stent8 (human abdomen)) using CUDA. The dimensions of volume size of one data set are 512x512x442 and another is 512x512x128 [16].

Algorithm: Ray Casting Technique

For each Pixel

1. For each f(i,j,k) along a Ray from pixel
2. Check f(i,j,k) in classification tables
3. If new substance
   a. Find Surf Normal/Compute color
   b. Weight color by Opacity
   c. Accumulate color contribution and Opacity
4. Pixel gets accumulated color

The advantages of using ray casting algorithms are [6]:

- No binary classification, e.g., inside or outside as in surface fitting methods
- Shows structure between surfaces
- Displays small and poorly defined features
- Readily parallelizes

Algorithm: CUDA-based Volume Rendering with Ray Casting Technique

(Each Block executes the following in parallel on GPU)

1. Render image using CUDA
   map backbuffer to get CUDA device pointer
   cutilSafeCall(cudaGraphicsMapResources();
   cutilSafeCall(cudaMemset();
   call CUDA kernel, writing results to backbuffer
   render_kernel(gridSize, blockSize, d_output, width, height, density, brightness, transferOffset, transferScale);
   Display results using OpenGL (called by GLUT)
   Now encode for performing the operations
   gridSize=dim3((width, blockSize. x), (height, blockSize. y));

2. Load raw data from disk
   void *loadRawFile()
   if (check fps limit) CUDA device with highest Gflops/s
   else (First initialize OpenGL context, so we can properly set the GL for CUDA then use command-line specified CUDA device, otherwise use device with highest Gflops/s)
   Then load volume data and synchronize
   cudaThreadSynchronize();
   calculate new grid size
   gridSize = dim3((width, blockSize.x), (height, blockSize.y));
   call CUDA kernel, writing results to Buffer
   copyInvViewMatrix(invViewMatrix, sizeof(float4)*3);
   Start timer 0 and process n plane loops on the GPU

3. Then free CUDA Buffer Memory
   freeCudaBuffers();

This system is doing by a single CUDA kernel. Conventional ray casting algorithms specify the ray attributes through the volume rendering of the volume dataset. In this algorithm, the opacity required for the changing the sampling rate globally or locally will be evaluated by equation 1:

$$\propto_{corrected} = 1 - (1 - \propto_{stored})^{n} \quad (1)$$

Where, $\alpha_{corrected}$ is adjusted opacity and α_{stored} is opacity stored in transfer function. Transfer functions shows variation between opacity and scalar values. Now the whole algorithm is given below:

4 Experimental Results

We have implemented the ray casting based volume rendering using CUDA and tested it on a datasets of size 300 MB and 45 MB. The colon_phantom8 (512x512x442) and human abdomen (512x512x128) datasets can be downloaded from http://www.gris.uni-tuebingen.de/edu/areas/scivis/volren/datasets/new.html [16]. Skin transparency based on density achieved along with the ability to rotate camera. This is shown in Fig.7. and Fig.8.

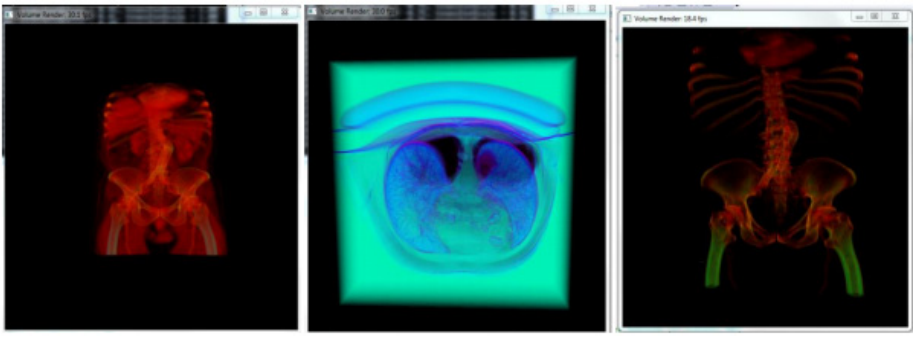

Fig. 7. Results from 45 MB dataset of human abdomen

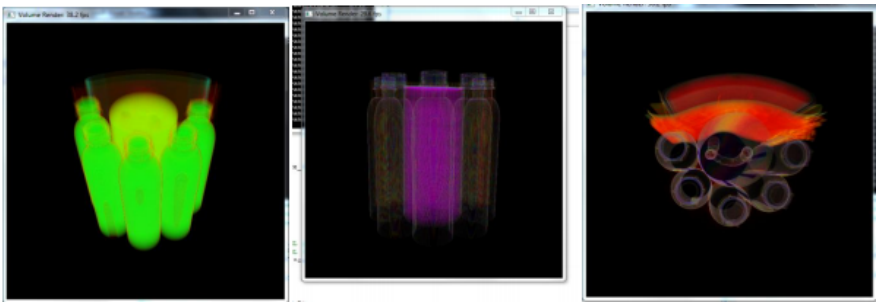

Fig. 8. Results including transparency from a 300MB dataset

The volume size of one data set is 512x512x442 and another is 512x512x128. These data sets give better result which is visualized with the help of ray casting algorithm using OpenGL in CUDA.

The above Table 1 and Fig.9 are showing the result analysis on two types of volume data sets. This shows that if use data of 512x512x442 in CPU the result would be problem in execution. But for another dataset CPU is given 10 fps which is less

Table 1. Comparison Results of performance on a NVIDIA Tesla C1060 GPU

Data Set Name	Volume Size Dimension	Rendering speed (FPS)	
		CPU	GPU(CUDA)
colon_phantom8 (300 MB)	512x512x442	--	30
stent8 (human abdomen 45 MB)	512x512x128	10	40

than as compared to GPU's result. Both data sets give almost 30 and 40 frames per second result with CUDA based rendering on GPU. The volume rendering has be done on a system equipped with Nvidia Tesla C1060 card with Nvidia GeForce 9500 GT. CUDA toolkit 4.0 is used in i7 950 CPU.

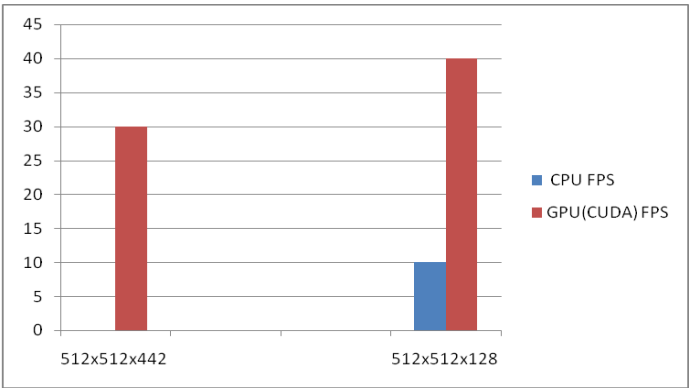

Fig. 9. Graph shows difference in CPU fps and GPU fps

5 Conclusion and Future Work

The advent of GPU's is changing the way complex computations were done till now. We envisaged that it would play a major role in the medical domain where doctors would be able to diagnose and analyze results provided by CT scans and MRI scans. The computational ability provided by the modern GPU's has enabled us to produce a three dimension interactive virtual human form.

To prototype the potential of CUDA in volume rendering, we used a data set of 300 MB through which we can zoom inside to get a better view or rotate to see the figure from a different angle. Further, through transparency change different organs hidden by layers of skin would also be visible.

The CUDA based code can be made to run on huge data sets of the order of giga bytes as explored in [15]. Currently a lot of research is going on in this front.

References

1. 1. Bi, W., Chen, Z., Zhang, L., Xing, Y., Wang, Y.: Real-Time Visualize the 3D Reconstruction Procedure Using CUDA. In: IEEE Nuclear Science Symposium Conference Record, pp. 883–886 (2009)
2. Zwecke, Eduard, Markus, Katja, Wien: GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery. PhD Thesis. Eurographics Digital Library Vienna University of Technology (2009), http://www.cg.tuwien.ac.at/research/publications/2009/beyer-2009-gpu/
3. James, K.T.: Ray Tracing Volume Densities. In: Proc. SIGGRAPH ACM, pp. 165–174 (1984)
4. Zhang, C., Xi, P., Zhang, C.: CUDA-based Volume Ray-Casting Using Cubic B-spline. In: IEEE International Conference on Virtual Reality and Visualization, pp. 84–88 (2011)
5. Wikipedia: Volume Ray Casting (last accessed November 10, 2012)
6. John, P.: Volume Visualization with Ray casting (1997), http://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm (last accessed November 30, 2012)
7. Suryakant, P., Narayanan, P.J.: Ray Casting Deformable Models on the GPU. IEEE ICVGIP, 481–488 (2008)
8. Susanne, S.K., Jose, G., Fabio, M., Andreas, A.M.E., Chrestoph, Z., Enrico, G., Renato, P.: Interactive Multiscale Tensor Reconstruction for Multiresolution Volume Visualization. IEEE Transactions on Visualization and Computer Graphics, 2135–2143 (2011)
9. NVIDIA CUDA (Compute Unified Device Architecture) programming guide version 1.0 (2007), http://www.nvidia.in (last accessed November 26, 2012)
10. Jens, F., Susanne, K.: Parallel Volume Rendering Implementation on Graphics Cards Using CUDA, pp. 143–153. Springer, Heidelberg (2010), http://link.springer.com/content/pdf/10.1007%2F978-3-642-16233-6_15 (last accessed November 30, 2012)
11. Bi, W., Chen, Z., Zhang, L., Xing, Y., Wang, Y.: Real-Time Visualize the 3D Reconstruction Procedure Using CUDA. In: IEEE Nuclear Science Symposium Conf., pp. 883–886 (2009)
12. Milan, I., Joe, K., Aaron, L., Charles, H.: Volume Rendering Techniques. Book Randima Fernando. GPU Gems NVIDIA, pp. 667–672 (2004), http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html (last accessed November 15, 2012)
13. Philipp, S., Maxim, M., Renato, P.: Extinction-based Shading and Illumination in GPU Volume Ray-Casting. IEEE Transactions on Visualization and Computer Graphics, 1795–1802 (2011)
14. Zhao, Y., Cui, X., Cheng, Y.: High-Performance and Real-Time Volume Rendering in CUDA. In: IEEE International Conference on Biomedical Engineering and Informatics China, pp. 1–4 (2009)
15. Agrawal, A., Josef, K., Gordon, C.J., Nigel, M.J., Feng, D., Marco, V., Fulvia, T., Debora, T.: Enabling the interactive display of large medical volume datasets by multiresolution bricking. ACM The Journal of Supercomputing, 3–19 (2010)
16. New Real World Medical Datasets, http://www.gris.uni-tuebingen.de/edu/areas/scivis/volren/datasets/new.html (last accessed December 20, 2012)