"Large Scale Volume Visualization on GPU"

MINI PROJECT REPORT
FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

BY

Juhi Kumari (IIT2013153)
Sneha Jha (IIT2013174)
Tanushree Anand (IIT2013192)
Nikita Agarwal (ISM2013016)
KM Ishu (ISM2013019)

UNDER THE SUPERVISION OF

Dr. Anupam Agrawal
Professor
IIIT-ALLAHABAD

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD

(A UNIVERSITY ESTABLISHED UNDER SEC.3 OF UGC ACT, 1956 VIDE NOTIFICATION NO. F.9-4/99-U.3 DATED 04.08.2000
OF THE GOVT. OF INDIA)
A CENTRE OF EXCELLENCE IN INFORMATION TECHNOLOGY ESTABLISHED BY GOVT. OF INDIA

6th May, 2016

CANDIDATES' DELARATION

We hereby declare that the work presented in this project report entitled "**Large Scale Volume Visualization on GPU**", submitted towards fulfillment of 6^{th} Semester report of B.Tech. (IT) at Indian Institute of Information Technology, Allahabad, is an authenticated record of our original work carried out from Jan 2016 to May 2016 under the guidance of **Prof. Anupam Agrawa**l. Due acknowledgements has been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Place: Allahabad

Date:

Juhi Kumari (IIT2013153)

Sneha Jha (IIT2013174)

Tanushree Anand (IIT2013192)

Nikita Agarwal (ISM2013016)

KM Ishu (ISM2013019)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Place: Allahabad

Dr.AnupamAgrawal

Professor

IIIT Allahabad

ACKNOWLEDGEMENT

We would like to acknowledge and extend our heartfelt gratitude to **Dr. Anupam Agarwal**, Indian Institute of Information Technology Allahabad, who guided us through this project. His keen vital encouragement, superb guidance, and constant support are the motive force behind this project work. We would like to show our war thanks to all our friends for their continuous motivation and encouragement during this project. We are very thankful to all the technical and non-technical staffs of the college for their assistance and co-operation.

Juhi Kumari
IIT2013153

Sneha Jha
IIT2013174

Tanushree Anand
IIT2013192

Nikita Agarwal
ISM2013016

KM Ishu
ISM2013019

ABSTRACT

A novel approach for GPU-based high-quality volume rendering of large out-of-core volume data has been implemented. By focusing on the storage of large volume data in an hierarchical data structure (i.e. octree) which stores the data in form of bricks , in addition every brick is further divided into macro-cells for easier traversal of data during run-time.

The approach performs branch-intensive accelerating structure traversal out of GPU raycasting loop and introduce an efficient empty-space culling method by rasterizing the proxy geometry of a view-dependent cut of the octree nodes . Octree traversal is now performed on CPU while rasterization and visualization processes are performed on the GPU. Rasterization pass can capture all of the bricks that the ray penetrates in a per-pixel list. Since the per-pixel list is captured in a front-to-back order, our ray-casting pass needs only to cast rays inside the tighter ray segments. During the process of evaluation and testing , this technique achieved 2 to 4 times faster rendering speed than a current state-of-the-art algorithm (which performs traversal of data structure on GPU) across a variety of data sets. As branch-intensive operations are operated in CPU at a better rate than on a GPU while floating point computations are better performed on GPU.

Table of Contents

1. Introduction

Our project focuses on volume or 3D visualization of human data which is large-scale and has to be interactive, hence we are motivated to use GPU techniques.

1.1 Motivation

With the improving technology, medical diagnosis and increasing population the size of medical data (MRI, CT Scan etc.) is increasing day by day. Thus, the problem of visualization of large data arises which is very prominent in this era. However, in order to deal with the ever-increasing resolution and size of today's volume data, it is crucial to use highly scalable visualization algorithms, data structures, and architectures in order to circumvent the restrictions imposed by the limited amount of on-board GPU memory. Volume visualization deals with methods to explore, analyze and visualize volumetric data acquired in medicine, computational physics and various other scientific disciplines. In other words, it is a method of extracting meaningful information from volumetric data using interactive graphics and imaging and it is considered with data representation, modeling, manipulation and rendering. Interactivity is making both the computational and the visualization effort proportional to the amount and resolution of data that is actually visible on screen (output-sensitive algorithms and system designs).

GPU-based large-scale volume visualization techniques based on the notions of actual output-resolution visibility and the current working set of volume bricks the current subset of data that is minimally required to produce an output image of the desired display resolution.

2. Problem definition

To visualize large volume medical human data on GPU efficiently. It can be used to view human body in different resolutions and perspectives based on the requirements of the user, hence interactive. This requires effective handling of large data in a structured way, keeping in mind the memory constraints.

2.1 Objective

To build a robust system which could be used to visualize data independent of its size on any inexpensive hardware having minimum specifications.

- Modelling the data using multi resolution model like octree.
- Selective rendering of data as required by the user, hence output-specific and Interactive.

3. Literature Survey

Table 1. Literature Survey Table

S. No.	Title	Year	Journal/ Conference	Objective	Method	Dataset Used & Dataset Size	Advantage	Disadvantage	Challenge(s) Dealt	Future Scope
1	A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets [3]	2008	Journal Visual Comput 2008	To present an adaptive out-of-core technique for rendering massive scalar volumes employing single-pass GPU ray casting	Volumetric dataset is decomposed into small cubical bricks, which are then organized into an octree structure maintained out-of-core and during run time loading the working set on GPU	Multi-gigavoxel CT dataset	Octree is used for data representation maintained out of Core. At runtime an adaptive loader, executing on the CPU, updates a view and transfer function-dependent working set of bricks maintained on GPU memory by asynchronously fetching data from the out-of-core octree representation. Out-of-core data management is used for filtering out as efficiently as possible the data that is not contributing to a particular image.	The rendering working set will not efficiently Update as the integration of level-of-detail and visibility culling techniques are not used which is required.	To handle vast data efficiently	To exploit the capability of our system to perform a full-volume ray tracing to produce higher quality images that incorporate more advanced shading effects
2	A Survey of GPU-Based Large-Scale Volume Visualization [2]	2014	Conference Eurographics Conference on Visualization (EuroVis) (2014)	To give an overview of the current state of the art in GPU techniques for interactive large-scale volume visualization		Large Volumetric Data	Octree is used as data structure in volume rendering Which enable adaptive level of detail, in addition to enabling empty space skipping. Used Output Sesitive Algorithm Was making their *running time*	For large-scale rendering, all the stages in this pipeline have to be scalable (i.e., in our context: output-sensitive), or they will become the bottleneck		

							dependent on the size of the *output* instead of the size of the *input*. Focus on Ray-guided and visualization-driven architectures.	for the entire application otherwise the visualization will not be done perfectly and accurately.		
3	Octree Rasterization: Accelerating High-Quality Out-of-Core GPU Volume Rendering [1]	2013	Journal IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS	Our target is to make high-quality visualization more practical in the clinic by the use of a consumer-level GPU	Store volume in an octree (of bricks) each having further split into regular macrocells. Using branch-intensive accelerating structure traversal out of the GPU raycasting loop and by rasterizing the proxy geometry of a view-dependent cut of the octree nodes. This rasterization pass captures all of the	Large out-of-core volume data.	It focus to make high-quality visualization more practical by the use of a consumer-level GPU. It introduce an new out-of-core GPU volume rendering framework, which combines object-order and image-order advantages and acts as a general acceleration technique, which makes more complex visualizations possible, while maintaining interactivity at the same time. The algorithm proposed can achieve 2-4 times faster rendering speed than a current state-of-the-art algorithm for large out-of-core data sets, while also producing high-quality rendering using tricubic interpolation.	The efficiency of the hierarchical bricking scheme often suffers from per-pixel traversal of the hierarchical acceleration structure, which is a branch-intensive operation and may not perform well on modern GPU architectures, which have a heavy performance penalty for divergent branching.	Maintaining octree on GPU limits the data size. Raycasting by passing each brick, hence increasing the overhead	This methodology can be used in other visualization systems, such as in multiple user activities, comparative settings and multiple volume studies, or time-varying data visualization in future work

					bricks that the ray penetrates in a per-pixel list.					
4	GPU-based Volume Rendering for Medical Image Visualization [4]	2005	Conference Engineering in Medicine and Biology 27th Annual Conference	To facilitate the usage of well-developed hardware resource, a graphics processing unit (GPU)-based volume ray-casting algorithm is proposed		CT scan of the Stanford terra-cotta bunny (512*512*360) real clinical human abdomen CT data, (400*400*344) MRI head scans (190*217*190) MRI head scans (256*256*109)	The work that it involved is to store volume data in texture, resample and interpolate them using hardware instead of software This paper presented a novel volume-rendering algorithm for medical image visualization using NV40 GPU. Based on the flexible programming model of FV40 pixel shader.	It does not used any data pipeline to implement ray-casting operation completely in GPU so because of this the big data cannot be visualize using this algorithm.	the algorithm employed a pre-classification method to classify voxels before interpolation, since this must be done in CPU, it costs much time	moving classification operation into GPU to get even quicker interactive speed
5	A Framework for Rendering Large Time-Varying Data Using Wavelet-Based Time-Space Partitioning (WTSP) Tree [5]	2004	Conference VG'05 Proceedings of the Fourth Euro graphics / IEEE VGTC conference on Volume Graphics	To present a new framework for managing and rendering large scale time-varying data using the wavelet-based time-space partitioning(WTSP) tree	The design goal of the WTSP tree is to support interactive browsing of data at arbitrary spatio-temporal scales.Using Haar wavelets, we can build a binary time tree associat	Large scale time-varying data	We present a new framework for managing and rendering large scale time-varying data using the wavelet-based time-space parti-tioning (WTSP) tree. We utilize the hierarchical TSP tree data structure to capture both spatial and temporal locality and coher-ence of the underlying time-varying data and exploit the wavelet transform to convert the data into a multiresolution	The primary goals of our work is to decorrelate the time-varying data into a range of spatial and tem-poral levels of detail, and to develop efficient data management Scheme to enable rapid run-time data retrieval and reconstructi	The formidable challenge for interactive volume rendering is the huge amount of data	Studying different approaches to combine the WTSP tree data structure with other data compression schemes. For instance, to incorporate the Laplacian pyramid structure into the WTSP tree and trade space for reconstruction and rendering time

					ed with each of the spatial nodes with a method similar to the error tree algorithm		spatio-temporal representation. During rendering, the wavelet-compressed data stream is decompressed on-the-fly and rendered using 3D hardware texture mapping. WTSP tree allows random access of data at arbitrary spa- tial and temporal resolutions at run time.	on. So basically it focused on the space and time management of rendering and visualization rather than the image quality and the results accuracy.		
6	Interactive Iso-surface Ray Tracing of Large Octree Volumes [6]	2006	Conference Interactive Ray Tracing 2006, IEEE Symposium on	To present a technique for ray tracing iso surfaces of large compressed structured volumes of data	Implementation is based on the following theoretical concepts: octree data format, point location and neighbor-finding, and the octree traversal	UNC DATA Blunt Fin (40x32x32) Protein (64x64x64) Enzyme (97x97x116) Dolphin (320x320x40) NMR Brain (256x256x109) CT Head (256x256x113)	It present a technique for ray tracing isosurfaces of large compressed structured volumes. By embedding the acceleration tree and scalar data in a single structure and employing optimized octree hash schemes, we achieve competitive frame rates on common multicore architectures, and render large time-variant data that could not otherwise be accomodated. It involves compressing volumes into an octree structure, and employing that for ray traversal.	Adaptive isosurface extraction techniques are fast, but depend on effective processing and streaming of large data to the CPU. They render a piecewise linear mesh that may be topologically different from the true isosurface as defined by the source data.	Interactive rendering of large volumes is a difficult problem in visualization. With direct volume rendering, GPU memory imposes an absolute limit on the volume size, and the video bus restricts real time rendering of time-variant data	Exploiting the multi resolution nature of the octree to provide a dynamic, view-adaptive level of detail scheme. Such a system would reduce the complexity and variance of the overall scene
7	Interactive GPU-based Volume	2009	Conference Conference	This method makes use of tri-linear	The proposed	Skull CT scan data set (256x256x256)	To facilitate the usage of well-developed	Due to the large number of	The interactive visualizatio	

	Rendering for Medical Image [7]		on Biomedical Engineering and Informatics	interpolation to Accelerate rendering speed. Besides for volume ray-casting back-to-frond order is used when the voxel's opacity is Accumulated to a threshold the after calculation will be discard.	algorithm implement ray casting operation completely in GPU. It re-samples volume data, represented as a stack of 3D texture, onto a sampling surface. The ray-casting algorithm performs in fragment shaders	Human head MRI data set (256x256x256) Pet data set for chest (512x512x256) Head CT scan data (256x256x84)	hardware resource, a graphics processing unit (GPU)-based volume ray-casting algorithm is proposed which implement ray casting operation completely in GPU. The algorithm re-samples volume data, represented as a stack of 3D texture, onto a sampling surface. It can perform an interactive rate even for direct volume rendering while keeping the high image quality.	trilinear interpolations that must be processed in order to produce image results of high quality, the availability of direct volume rendering has yet been restricted to high-end workstations and special purpose graphics hardware.	n of such data is a challenge, since the frame rate is heavily depending on the amount of data to be visualized	
8	Mapping High-Fidelity Volume Rendering for Medical Imaging to CPU, GPU and Many-Core Architectures [8]	2009	Journal IEEE Transactions on Visualization and Computer Graphics	To learn and analyze new volumetric rendering algorithms that are suited to modern parallel processing architectures	It uses thread- and data-parallel implementation of ray-casting that makes it amenable to key architectural trends of three modern commodity parallel architectures: multi-core,	Three sets of human CT data (16-bit) Large medical human dataset (750x750x1000)	This paper describe a thread- and data-parallel implementation of ray-casting that makes it amenable to key architectural trends of three modern commodity parallel architectures: multi-core, GPU, and an upcoming many-core Intel R architecture code-named Larrabee. Overall parallel implementation of ray-casting delivers close to 5.8x speed-up on	The advantage is the challenge to provide improved health care efficiently, which is complicated by the magnitude of the data. Despite the availability of several general purpose and specialized rendering engines, volume visualization has not	The challenge is to provide improved health care efficiently, which is complicated by the magnitude of the data. Despite the availability of several general purpose and specialized rendering engines, volume visualization has not been	

					GPU, and an upcoming many-core IntelR architecture code-named Larrabee		quad-core Nehalem over an optimized scalar baseline version running on a single core Harpertown.	been widely adopted by the medical community except in certain specific cases. In ray-casting, as the ray's traverses through the volume, they access voxels with a non-constant stride.	widely adopted by the medical community except in certain specific cases	
9	Large Scale Volume visualization on GPU A Just-in-Time Compiled Sparse GPU Volume Data Structure [9]	2015	Journal IEEE Transactions on Visualization and Computer Graphics	To reduce the memory bandwidth (bottleneck in GPU) for sparse volume data structures that pose a tradeoff between memory efficiency and access performance	Present a new sparse volume hybrid data representation (JiTTree) that makes it possible to combine multiple elemental data structures resulting in lower memory requirements while enabling high runtime performance. No memory fetches are required for the kd-tree	Stag beetle dataset (brick size-32*32*32) Kingsnake dataset (brick size-128*128*128)	This paper presents JiTTree, a novel sparse hybrid volume data structure that uses just-in-time compilation to overcome the representation problems. In this paper we present JiTTree, a new type of sparse volume data representation that enables memory-efficient storage and highperformance data access. The JiTTree is the just-in-time compilation stage that transforms data into efficient access functions. Thereby it enables the use of multiple kinds of data structures to represent one volume, without introducing a significant traversal overhead. By	The basic principle of the data structure is to adapt to the local sparsity of a specific data set. Other volume representations often make the distinction between dense and empty (or homogenous) regions and treat these regions differently.	The JIT approach transforms memory-bound programs into instruction-bound programs. Although data structure is not designed for dense data, it outperforms other representations, such as a dense bricking, for most cases. However, for a better analysis of the data structure we used data sets with varying scarcity.	just-in-time compilation of the root level to improve performance ;to add dynamic write capabilities in addition to the read-only access; improve the JIT compilation approach for other memory access patterns like ray traversal

					traversal. The splitting axis order is directly encoded in the Conditionals. Splitting plane positions are inserted as literals.		combining multiple sparse data structures and reducing traversal overhead we leverage their individual advantages. JiTTree reduces the traversal overhead of the resulting optimal data structure			
10	A Survey of Octree Volume Rendering Methods [10]	2006		survey and comparison of existing works employing octrees for volume rendering		Large Volume Data	This paper surveys and compares existing works employing octrees for volume rendering. It focuses specifically on extraction, direct volume rendering, and iso surface ray tracing. It survey several varieties of octree and efficient hashing schemes for their traversal.	It only examine octree structures that are of interest in volume Rendering not for the other purpose so we cannot compare its usefulness towards the particular purpose.		Future applications of octree volume rendering could attempt to combine the pure octree volume with GPU rendering approaches, using out-of-core methods
11	Cell Octrees: A New Data Structure for Volume Modeling and Visualization [11]	2001	Conference, Proceedings of the Vision Modeling and Visualization	An improvement of bono approach which uses an incomplete octree with a smaller memory		Large Volume Data	It propose a new indexing method that uses an incomplete octree and which requires less memory to be stored. It present an improvement of BONO (Branch On Need Octree) approach which uses an incomplete octree with a smaller memory. The method indexes a set of	When the number of cells of any dimension of the volume is not a power of two, or is not equal to the number of cells of another dimension of it, nodes will not have all eight sons.	An improvement of bono, noted as cells octree, has been done	property values may be stored in the tree, so it is not necessary to store the grid and there may be further reductions in the storage requirements

							cells as a block when their property value is nearly uniform. In this approach the prune condition is independent on threshold and point of view. The branches of the tree that do not index cells containing the isosurface are not processed, so the isosurface building time is reduced.	In such cases, nodes will have a larger ramification in the levels which are close to the leaf nodes, and a smaller ramification in the levels close to the root node. Consequently, the storage requirement is smaller.		.
12	Multi-Resolution volume visualization with a texture-based octree [12]	2001	Journal The Visual Computer 2001	Reducing the amount of texture memory needed to render a volume dataset, and thus reducing the texture loading overhead	Texture-based octree	MRI brain (32 MB) MRI Brain (4 MB) CT jaw (32 MB) CT Jaw (11 MB) CT Vertebra (64 MB) CT Vertebra (42 MB)	The purpose of this paper is to speed up the texture based rendering of large datasets. The paper propose a new texture memory representation and a management policy that substitute the classical one-texel per voxel approach for a hierarchical approach. The hierarchical approach benefits nearly homogeneous regions and regions of lower interest. The proposed algorithm is based on a simple traversal of the octree representation of the volume data.	The algorithm proposed in the paper does not consider view-dependent criteria (which are generally based on the perspective distortion and shrinking of farthest data sections) because, in volume rendering, the differences in the projecting ratio are very limited. This paper is basically focused on the		"Future work" is Addressed to analyze Different Heuristics to select the interest Function that better Satisfies user-Defined Restrictions. Problem of the Combining surface and Volume Information n

							Driven by a user-defined image quality, defined as a combination of data homogeneity and importance, a set of octree nodes (*the cut*) is selected to be rendered. The degree of accuracy applied for the representation of each one of the nodes of the cut in the texture memory is set independently according to the user-defined parameters.	speedup of the 3D volume visualization so the other attributes (i.e. image quality)are considered as a secondary choice		

4. Methodology

The visual Human male dataset is used for experimentation. It is of 3.15 GB RAW file. The data is made up of a stack of images of resolution 1760 X 1024. The stack consists of 1878 slices. Each slice represents the body cut horizontally. The paper [1] is our base research paper as it introduces a novel method to process the large volume data efficiently.

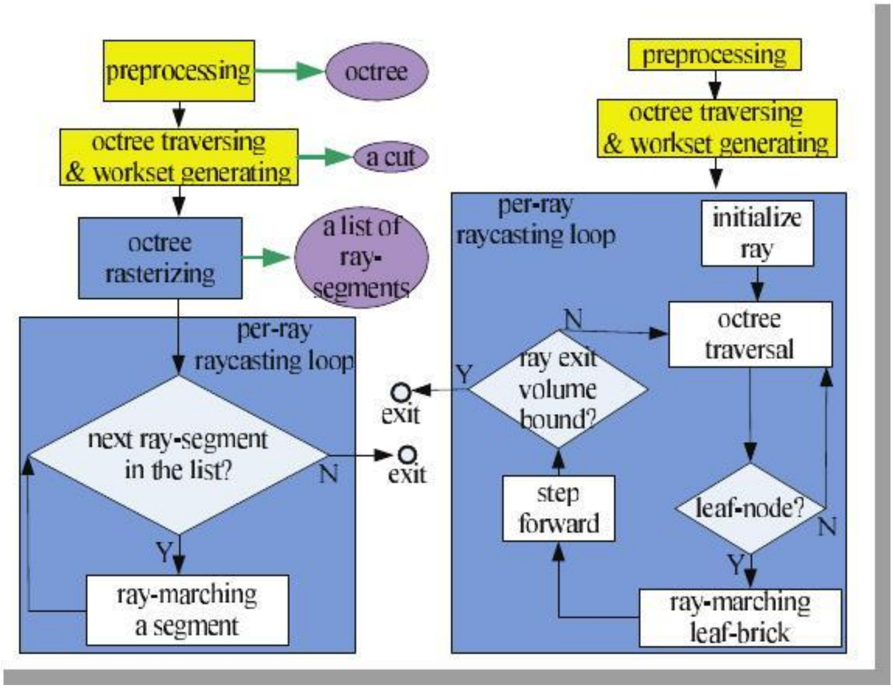

Figure 1: Flowcharts of our algorithm (left) and previous out-of-core GPU volume Rendering algorithms (right). Yellow boxes are executed on the CPU, all other boxes on the GPU. Black arrows indicate control flow; green arrows indicate output of intermediate data.

The flow of method in Figure 1 is explained as follows [1] :

4.1. Pre-processing

A large volume dataset is first pre-processed using blocking techniques (division of large dataset in blocks of data) and storing it in Octree (a 3-D data structure is represented in the form of recursively sub-dividing into eight octants). An **octree of bricks** is constructed where the actual resolution data is stored in leaf nodes. At each level or node a brick of same dimension B_{res}^{3} is made. The length of the tree is kept shallow which results in courser bricks. Inner bricks are built using down sampling of the lower level nodes like averaging filter.in this way the whole data is represented as multi-resolution hierarchy which is maintained on CPU (out of core). Each node of the octree points to the brick with a constant resolution that approximates the part of the volume corresponding to the octree's node.

Bricks store extra overlapping voxels, which helps in accessing the neighboring voxels at runtime using the tri-cubic interpolation and gradient computations. Since the octree don't use any transfer function for empty space culling it uses macro-cells to store the min-max scalar values for the corresponding brick, for efficient brick culling. The dimension for macro-cell is M_{res}^{3}.

4.1.1. Octree Creation and Visualization of View-Dependent Working Set :

1878 slices were first constructed into a single 3D-Volume. Dimensions of each image slice is 1760 * 1024. Brick size of evry node in octree is taken as 220*128*235.Single large Volume is splitted into 512 bricks of resolution 220*128*235. These bricks are then averaged by a factor of 8 to construct 64 bricks of same resolution which are further averaged by a factor of 8 to construct 8 bricks which are then combined to form a single root. These bricks are then saved as nodes of an octree where root acts as parent having 8 children which are individually divided into 8 more children. Struct of the octree node is as follows –

```
struct node{
        ifstream fp;
        struct node *child[NO_CHILDREN];
        struct node *parent;
        int level;
        int a[3];
        int no_children;
        int is_leaf;
        string name;
};
```

Table 2: Structure of octree

4.2. Generating a View-Dependent Working Set

For viewing different frames a cut is decided out of the octree according to the required frame. The cut includes different resolution nodes as per needed dependent on the viewing angle. This cut is used to update the GPU brick pool for rendering. If we zoom the image the children of the current node has to be loaded and if we zoom out then multiple nodes has to be fused together. For deciding the cut breadth first order octree traversal is used starting from the root node and is continued till the required node is found.

Work-set Generation Algorithm

```
Initialize queue with root tag = 2

while(stack_not_empty) {
   cur = queue.front()
   queue.pop()

   split cur in 8 octants
   if possible
   for (i = 0; i < 8; i++) {
      if ROI lies in octant[i]
         if ROI lies completely in octant[i]
            tag = 1
            push in brick_pool
         else
            tag = 2
            push in queue
      else
         tag = 0
   }

}
copy brickpool to cudaMemory
```

Table 3: Workset Generation Algorithm

4.2.1. View-Dependent Sorting of the Bricks

The nodes in the cut are rasterized before. Hence we sort the nodes and store them in front to back fashion using a pointer list which is stored for further use. Pointers to the traversed cut-nodes are stored in a STL list. A traversed node is always replaced by in front its children to back order during depth-first search, this is done till a node labeled as node cut is found. It is implemented using an 8 X 8 lookup table. Each row of the look up table encodes a possible order of the 8 octants according to the viewpoint in the octree space.

4.2.2. Memory Management of the Working Set

The cut represents the brick data and macro-cell data of the nodes. Tis data is then transferred to GPU asynchronously. The working set is loaded into 2 memory pools:

- **Brick Pool**

   It is organized ad a 3D texture of specific size and dimension. Each of the cell of this pool corresponds to a particular brick, and the cell stores the corresponding brick Id B_{ID}.

- **Macro-cell Pool**

   The 3D macro-cell pool is packed with the corresponding brick's macro-cell at particular brick id B_{ID}.

4.3. Proxy-Geometry Rasterization

The comparison between 2 pass and normal rasterization is shown in Fig 3. The explained process of two pass rasterization could be seen in Fig 4. Proxy geometries of the whole working set of bricks is rasterized in the first pass and capture all bricks that a ray penetrates into a per-pixel list. In second pass, we rasterize all non-empty macrocell (of all bricks) and refine per-pixel list in which each element will contain a ray-segment corresponding to the brick that the ray penetrates. This results in skipping of empty spaces of macrocell. Fig 4. Gives detailed process for the 2nd pass i.e. traversing the macrocells.

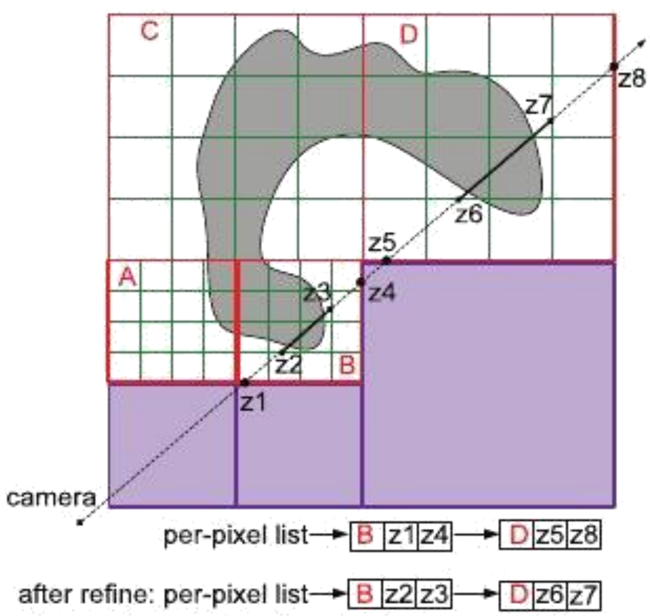

Figure 2: Algorithm philosophy. The octree-cut is composed of all of the active bricks (red—A, B, C, and D) at different LoDs depending on the current view. Each active brick is subdivided into macrocells (green). Empty bricks (purple) are never added to the octree-cut. We first rasterize active blocks so that the fragment shader can capture all of the active blocks that the ray penetrates (B and D) into a list in front-to-back order. Then we rasterize nonempty macrocells and refine the z-values in the list in order to get a tighter ray segment (shown as the solid black line) for each brick at the accuracy of the macrocell level.

4.4. Raycasting

A ray is generated for each desired image pixel. Using a simple camera model, the ray starts at the centre of projection of the camera (usually the eye point) and passes through the image pixel on the imaginary image plane floating in between the camera and the volume to be rendered. In the raycasting pass, depth interval is defined by the two end points of a ray-segment that performs the GPU raycasting for the corresponding brick.

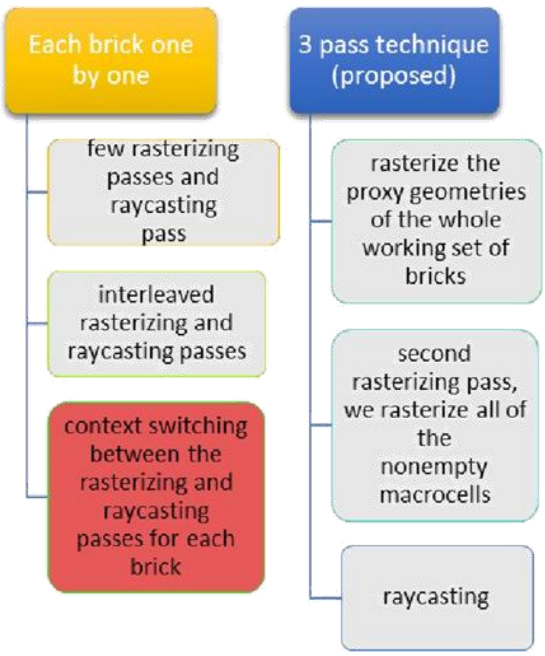

Figure 3: Comparison between 2 pass rasterization Process and normal method

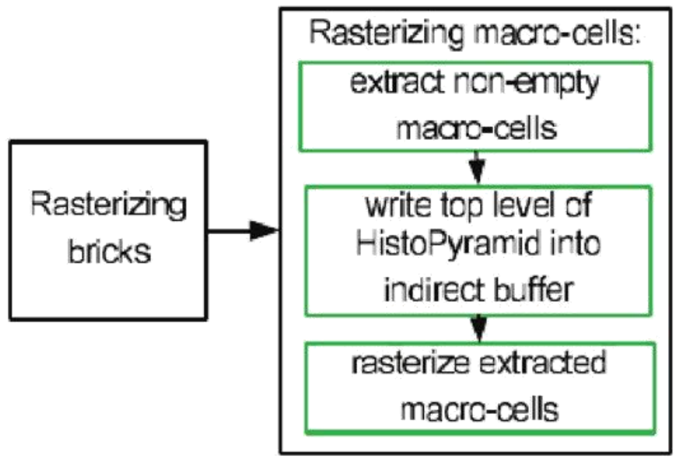

Figure 4: Flowchart of rasterization

5. Hardware and Software requirements

5.1 Software Requirements

- CUDA enabled system
- OpenGL
- Visualization Toolkit (VTK)
- Visual Studio (8+)
- National Library of Medicine Visible Human Body (Male) 3.15 GB Dataset

5.2 Hardware Requirements

SPECIFICATIONS	
GPU processor	Tesla C1060
CUDA Cores	240
Shader clock	1296 MHz
Memory interface	512-bit
Memory	5888 MB
Total available graphics memory	4096 MB
Bus	PCI Express x16

6. Activity Time Chart

Work done till Mid-Sem				End-Semester			
Phase 1: 10th Jan - 25th Jan	**Phase 2: 25th Jan – 5th Feb**	**Phase 3: 5th Feb – 12th Feb**	**Phase 4: 12th Feb – 25th Feb**	**Phase V: 25th Feb – 10th Mar**	**Phase VI: 22nd Mar – 10th Apr**	**Phase VII: 10th Apr – 20th Apr**	**Phase VIII: 20th Apr – 4th May**
Literature Survey	Data Collection and analyzing the format of the data	Environment Setup: OpenGL, CUDA	Learning OpenGL and CUDA	Visualization of small dataset	Implementation of octree data structure to manage the large volume data	Rendering small portion of octree	Visualization Pipelining and Multi – Resolution Model

7. Experimental Setup and Results

Experimental Setup requires following:

5. Visual Studio
6. Setting OpenGL
7. CUDA

Following are the screenshots of the code for visualization of small dataset that we have done so far:-

(a) (b)

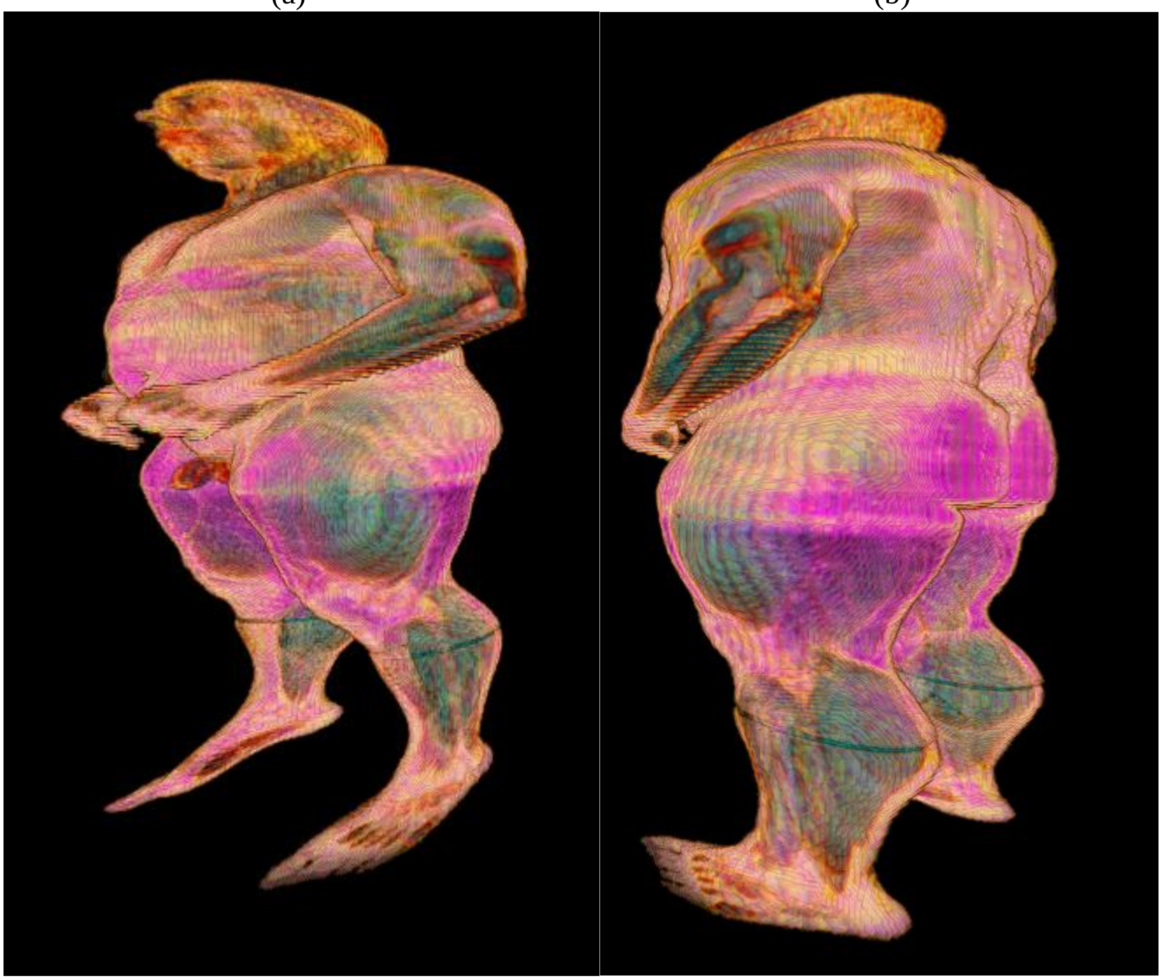

Figure 5: (a) and (b) are 2 different orientation of root node

(a) (b)

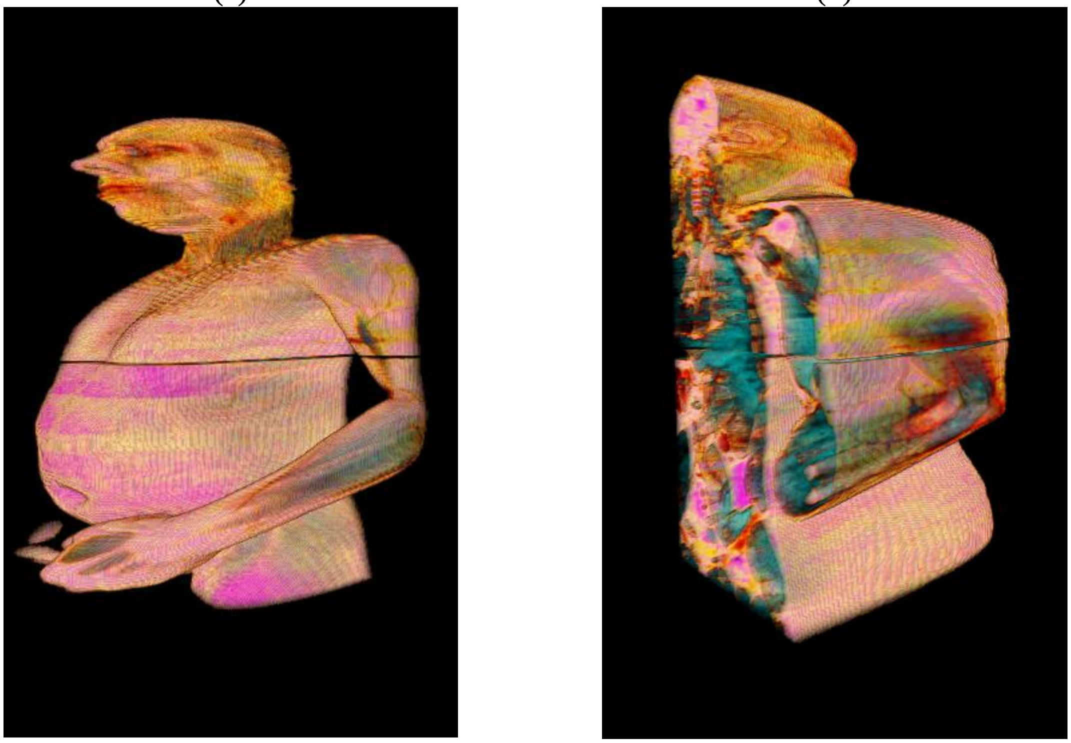

Figure 6: (a) 0^{th} child and (b) 2^{nd} child of root node

(a) (b)

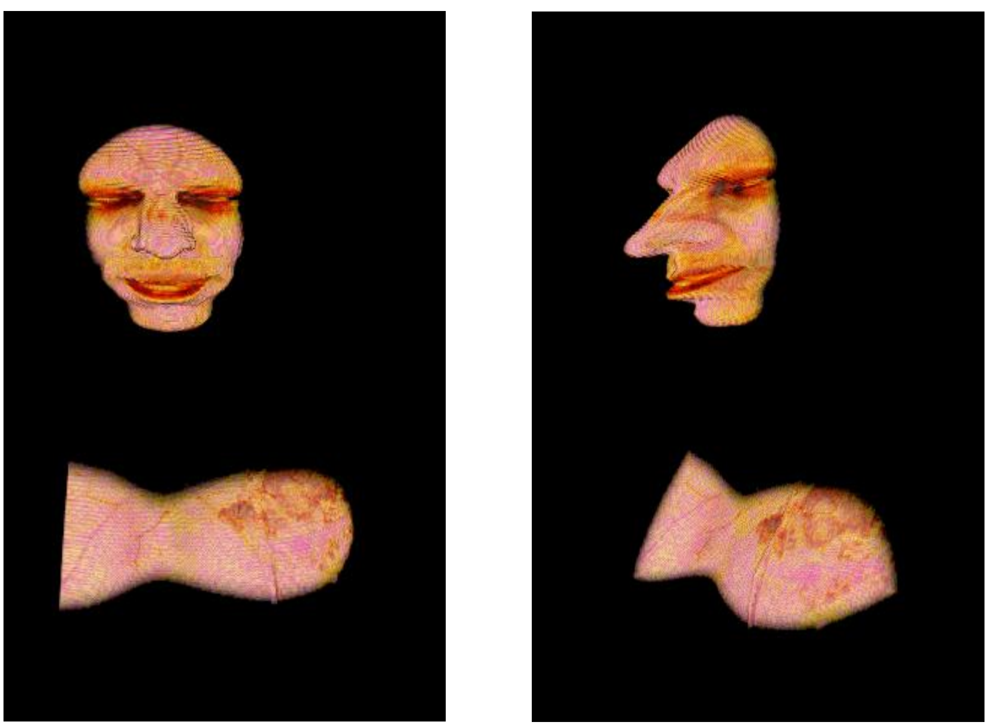

Figure 7: (a) shows 011 (b) 012

8. Performance Comparison

Performance comparison on the basis of rendering quality, when we performed visualization using the source codes of different algorithms which are available as open source and on our technique, the results obtained have smoother image quality from the technique using GPU octree rasterization methods (i.e. the proposed methodology).

Performance comparison on the basis of Rendering Speed with a GPU Octree Traversing Algorithm Octree creation method takes 5-10 minutes on any type of datasets. After which octree traversal, rasterization , per-pixel list generation and rendering are done in matter of minutes as they are performed on GPU using maximum number of cores which can be utilized. As octree creation for a dataset is only performed once and stored in hard disk , the rendering speed is 2-4 times greater than what was performed by previous algorithms which performed octree traversal on GPU.

Table 4: Comparison between CPU and GPU time

S. No.	**Processor**	**Time Require from 8 to 1 brick construction**	**Time Require from 64 to 1 Brick construction**
1.	**Quad Core**	~20 mins	~2 hrs
2.	**Intel i3**	~12 mins	~45 mins
3.	**Intel i5**	~5 mins	~25 mins
4.	**GPU**	~micro secs	~secs

9. Future Scope and Conclusion

In our method which we have implemented is a fast, GPU based out-of-core volume ray-casting, which moves the branching intensive octree traversal out of the GPU ray-casting loop and after the traversing execute it on the GPU. By introducing the tighter depth range for GPU ray-casting it exercised the greater control over the rendering process by using the hardware rasterization unit performance. The method is also offers more sophisticated empty space skipping, and which makes possible the interactive use of advanced features like cubic interpolation for the large out-of-core data sets. Since the method that is presented provides a general accelerating scheme (by rasterizing an octree to generate tight ray segments), and the method can also be used to improve the performance of the other visualization systems, such as in multiple user activities, comparative settings and multiple volume studies, or as a future work apart from these we can also use it for time-varying data visualization. Improvements like a more generalized method to deal with the size of brick irrespective of the size of data and optimum number of levels will help to enhance the performance.

10. References

[1] B. Liu, G. J. Clapworthy, F. Dong and E. C. Prakash "Octree Rasterization: Accelerating High-Quality Out-of-Core GPU Volume Rendering" IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 19, NO. 10, pp. 1731-1745, OCTOBER 2013

[2] J. Beyer, M. Hadwiger and H. Pfister, "A Survey of GPU-Based Large-Scale Volume Visualization", Eurographics Conference on Visualization (EuroVis), 2014

[3] E. Gobbetti, F. Marton and A. I. Guiti ́an, "A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets", Springer-Verlag 2008 pp: 797–806, 2008

[4] Y. Heng and L. Gu, "GPU-based Volume Rendering for Medical Image Visualization", Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, pp. 5145-5148, September 1-4, 2005

[5] C. Wang and H. Shen, "A Framework for Rendering Large Time-Varying Data Using Wavelet-Based Time-Space Partitioning (WTSP) Tree", Department of Computer and Information Science, The Ohio State University, 2004

[6] A. Knoll, I. Wald, S. Parker and C. Hansen, "Interactive Iso-surface Ray Tracing of Large Octree Volumes", Scientific Computing and Imaging Institute, University of Utah, Technical Report No UUSCI-2006-026, 2006

[7] S. Chen and C. Hao, "Interactive GPU-based Volume Rendering for Medical Image", Biomedical Engineering and Informatics, BMEI '09. 2nd International Conference on 17-19 Oct, 2009

[8] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D. M. Carmean, D. Hanson, P. Dubey, K. Augustine, D. Kim, A. Kyker, V. W. Lee, A. D. Nguyen, L. Seiler and R. Robb, "Mapping High-Fidelity Volume Rendering for Medical Imaging to CPU, GPU and Many-Core Architectures", IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 15, NO. 6, Nov/Dec, 2009

[9] M. Labschutz, S. Bruckner, M. E. Groller, M. Hadwiger and P. Rautek, "JiTTree: A Just-in-Time Compiled Sparse GPU Volume Data Structure", Visualization and Computer Graphics, IEEE Transactions, Vol. 22, Issue. 1, pp. 1025-1034, 2015

[10] A. Knoll, "A Survey of Octree Volume Rendering Methods", Scientific Computing and Imaging Institute University of Utah, 2006

[11] F. Velasco and J. C. Torres, "Cell Octrees: A New Data Structure for Volume Modeling and Visualization", VMV '01 Proceedings of the Vision Modeling and Visualization Conference, pp. 151-158, 2001

[12] I. Boada, I. Navazo and R. Scopigno, "Multi- Resolution volume visualization with a texture-based octree", Springer-Verlag, pp. 185-197, 2001

[13] A. Agrawal, J. Kohout, G. J. Clapworthy, N. J.B. McFarlane, F. D. M. Viceconti, F. Taddei and D. Testi, "Enabling the interactive display of large medical volume datasets by multiresolution bricking", Springer, pp. 3-19, 2009