**“Large Volume Data Visualization**”

***PROJECT REPORT***

*FOR THE*

***SUMMER RESEARCH INTERNSHIP PROGRAMME 2017***



*BY*

Jalaz Kumar

Department of Computer Science & Engineering

NIT Hamirpur, H.P

Aman Agrawal

Department of Information Technology

HBTU Kanpur, U.P

***UNDER THE SUPERVISION OF***

Dr. Anupam Agrawal

Professor

IIIT-ALLAHABAD

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,**

**ALLAHABAD**

**10th July, 2017**

# CANDIDATES’ DECLARATION

We hereby declare that the work presented in this project report entitled “**Large Volume Data Visualization**”, submitted towards fulfilment of Summer Research Internship atIndian Institute of Information Technology, Allahabad, is an authenticated record of our original work carried out from May 15, 2017 to July 10, 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.

Aman Agrawal

HBTU Kanpur

Jalaz Kumar

NIT Hamirpur

Allahabad

July 10, 2017

**CERTIFICATE**

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

Dr. Anupam Agrawal

Professor

IIIT Allahabad

Date: 10 July 2017

Place: Allahabad

**ACKNOWLEDGEMENT**

We would like to acknowledge and extend our heartfelt gratitude to **Dr. Anupam Agrawal**, 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 thank the various research scholars of MHRD Lab namely Upendra Sir, Parmatama Sir, Nayaneesh Sir & Piyush Sir who all helped us in the hour of need.

We would like to show our gratitude to all our fellow summer interns at IIITA for SRIP 2017 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.

Aman Agrawal

HBTU Kanpur

Jalaz Kumar

NIT Hamirpur

**ABSTRACT**

A novel approach for GPU-based 3D volume rendering in high quality of large out-of-core volume data has been implemented. Our prime focus was on the storage of large volume data in a 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 accelerating structure traversal out of GPU ray casting loop in an intensive manner and introduce an efficient & reliable empty-space culling methodology by rasterizing the proxy geometry of a view-dependent portion of the octree nodes. Octree traversal is now performed on CPU while rasterization and visualization processes are performed on the GPU. Rasterization pass is able to capture all of the bricks that the ray penetrates in a per-pixel list. Moreover, as the per-pixel list is captured in a front-to-back order, our ray-casting pass requires only to cast rays inside the tighter ray segments. During the phase of evaluation and testing, this approach achieved 2 to 4 time faster rendering speed than the 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.

Moreover, efforts have been made to build the whole system more user-friendly & interactive. By enabling both keyboard as well as mouse functionalities, extensively working over the UI part, the final product is more user-oriented.

**Table of Contents**

1. Introduction……………………………………...…….………………………………1
2. Problem definition and Objectives…………...………………………………………..1
3. Literature Survey…………………………..………………………………………..2-9
4. Methodology……..……………………………………………………………….10-15
5. Hardware and Software Requirements ……………….……………………………...16
6. Activity Time Chart ………………………………….……………………………...17
7. Experimental Setup and Results…….……………….……………………...…….18-19
8. Performance Comparison…………………………….…………………………………..20
9. Conclusion and Future Scope...……………………………………………………....21
10. References…………………………………………………………………………...22

**1. Introduction**

Our project focuses on 3D visualization of human medical 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, so as to deal with the ever-increasing resolution and size of today’s volume data, it is really crucial to use highly scalable visualization algorithms, data structures, and efficient architectures in order to circumvent the restrictions imposed by the constrained amount of on-board GPU memory. Volume visualization deals with methods to explore, analyse 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, modelling, manipulation and rendering. Interactivity in resolution is making both the computational and the visualization effort proportionally equivalent to the amount of data that is actually visible on screen (output-sensitive algorithms and system designs).

GPU-based large volume data visualization techniques based on the notions of actual output-sensitive resolution visibility and the current working set of volume, bricks the current subset of dataset 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 & Size** | **Advantage** | **Disadvant-ages** | **Challenges 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 large scalar data 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 the GPU | Multi-gigavoxel CT dataset | Octree is used for data representation maintained out-of-core. At runtime, an adaptive loader, which executes on the CPU, updates the view and transfers the function-dependent working set of bricks maintained on GPU memory by asynchronously fetching data from the out-of-core octree traversal. Out-of-core data management is beneficial for filtering out as much data in an efficient way that is not contributing to a particular image | The rendering working set will not efficiently update as the integration of level-of-details available and the visibility culling techniques are not used which is required | To handle vast data efficiently | To exploit the capability of this system to perform a full- volume ray tracing so as to produce higher quality images that incorporate in them 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 data  volume  visualization |  | Large Volumetric Data | Octree is use as the data structure in volume rendering which enables adaptive level  of details  Along with empty space Skipping.  Using Output  Sensitives  Algorithm  Was making  their running  time | For large-scale  Rendering of data, 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 generated rather than the size of the input given.  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 | The prime target is to make high-quality  visualization  more practically available in  the clinic by the use of a consumer-  level GPU | Store volumes of data in an octree  (of bricks)  Each having further split  Into regular  Macrocells.  Using  accelerating  structure  traversal in an efficient manner  out of the GPU  raycasting loop and by  rasterizing the proxy  Geometry of a view dependent portion of the octree Nodes.  This  Rasterization pass captures  majority of the bricks  that the  ray  penetrates in a per-  Pixel list. | Large out-of-  Core 3D volume  Data. | It prime focus is to make  high-quality  visualization to be more practical by the  use of a  consumer level  GPU. It introduces a new out-of-core  GPU 3D volume  Rendering way,  which combines  object-order and  image-order  advantages and  acts as a general  acceleration  approach, which  makes complex  visualizations  possible, while  maintaining  interactivity at  the same time.  The algorithm  proposed can  achieve 2-4 times faster rendering speed than the 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  octree  scheme involving bricks  often suffers  from per-pixel  traversal of  the  acceleration structure,  which is a  branch-  intensive  operation  and may not  perform well on some 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 various  visualization systems,  such as in  multiple user  activities,  comparative  and multiple  volume  studies, or on  time-varying  data  visualization in future  work |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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  some well-developed  hardware  resource, a  graphics  processing  unit (GPU)-  based 3D volume ray-casting  algorithm is  proposed |  | CT scan of the  Stanford  terracotta 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 the  storage of volume data in texture, resampling and interpolating  them using  hardware instead  of the software.  This paper  presented a novel  3D volume-rendering  algorithm for  medical data  visualization  using NV40  GPU. Based on  the flexible  programming  model of FV40  pixel shader. | It does not  use any  data  pipeline to  implement the 3D  ray-casting  operation  completely  in GPU so, the large-scale  data cannot  be visualized  using this  Algorithm. | The  algorithm  employed a native pre-  classification method to classify the voxels before  interpolation, since this must be done in CPU, it  costs much  time | Transferring  classification  operation  into the 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 methodology  to manage  and render  large scale  time-varying  data using  the wavelet-  based time-  space  partitioning (WTSP) tree | The design  goal of the  WTSP  tree is to  support the  interactive  browsing of data at some  arbitrary  spatio-  temporal  scales using Haar  wavelets, we can build a  binary time  tree  associated with each of the spatial nodes with a method  similar to the error | Large  scale time-  varying data | We present a  new approach  to manage and  render large  scale time varying data using the  wavelet-based  time-space  Partitioning (WTSP) tree.  We utilize  the hierarchical  WTSP tree data structure to capture the spatio-temporal locality  and coherence of the underlying time-varying dataset and exploit the  wavelet transform to convert the data into a  multiresolution | The  Primary goals of  our work is  to decorrelate  the time-  varying  dataset into some range of spatio-temporal level of details, and to develop  an efficient data management  Scheme to  Enable rapid dynamic run-time data  retrieval  and  Reconstruction. | 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 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 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  spatial and temporal  resolutions at runtime. | 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 | 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  structure,  point location and  nearest neighbour  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 accelerated  octree and scalar data in a single  structure and  employing  optimized octree  hash schemes, we achieve the competitive frame rates on some common multicore  architectures, and are able to  render large  time-variant data  that could not  otherwise be  accommodated.  It involves  Compressing the large-scale  volumes into an octree structure, and employing that for ray traversal. | Adaptive  isosurfaces  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  isosurfaces  as defined  by the source  data. | Interactive  Rendering of large  volumes is a difficult  problem in  visualization. With direct  volume  rendering,  GPU memory  Is able to impose an  absolute  constraint on the  volume size, and  the video bus  restricts real time 3D rendering of time-variant datasets. | Exploiting  the multi  resolution  nature of  the octree  to generate a dynamic,  view-  adaptive  levels of  detailed  scheme.  Such a  system  would be able to  reduce the  complexity and  variance of the overall  scene |
|  |  |  |  |  |  |  |  |  |  |  |

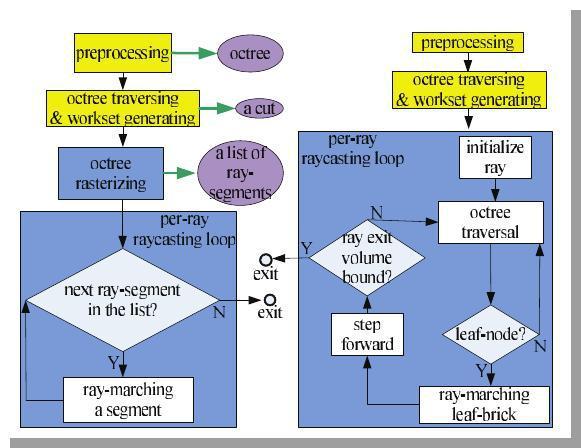
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | Interactive  GPU-based  Volume  Rendering for  Medical Image  [7] | 2009 | Conference  Conference on  Biomedical  Engineering and  Informatics | This method  makes use of  the tri-linear  interpolation  to Accelerate  the speed of rendering.  Besides the usage of  volume ray-  casting in  back-to-  frond order when  the voxel’s  opacity is  accumulated to a threshold then after  calculation  will be discarded. | The  proposed  algorithmic approach  implements ray casting  operation  completely in GPU. It performs  re-sampling  3D volume  data,  represented as a stack of 3D texture, onto a  sampling  surface.  The 3D volumetric ray-  casting  algorithm  performs in  fragment  shaders | Skull CT scan  data set  (256x256x256)  Human head  MRI data set  (256x256x256)  Pet data set for  chest  (512x512x256)  Head CT scan  data  (256x256x84) | | To facilitate the  use of well- developed hardware resource,  GPU-based  3D volume ray- casting algorithm is proposed which implements the volumetric  ray casting  operation  completely in the GPU. The algorithm resamples 3D volume data, represented  as a stack of 3D texture onto a sampling surface.  It is processed on an interactive rate even for direct volumetric  rendering while  keeping the high image quality. | | Due to the  Very Large number of trilinear  interpolations that have to be processed  so as to  produce image results of high  quality, the  availability  of direct 3D volume rendering has yet been  restricted  to high-end  workstations and special  purpose graphics  hardware. | | The  visualization of such data in an interactive manner is a challenge,  since the frame rate is heavily  dependent on the amount of  data that have 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 the  Thread and  data  parallel  implement-ation of ray-  casting that  makes it  suitable to key  architectural  trends of three major  commodity  parallel  architectures:  multi-  core, GPUs,  and an  upcoming  many-core  Intel R  architecture  code-named  Larrabee | Three sets of  human CT data  (16-bit)  Large medical  human dataset  (750x750x1000) | | This paper describe a thread and data parallel  implementation  of volumetric ray-casting that makes it suitable to key architectural trends of three modern commodity parallel architectures:  multi-core, GPUs,  and an upcoming  many-core Intel  R architecture code-named Larrabee. Overall  implementation  of ray-casting in parallel manner  delivers close to  5.8x speed-up on quad-core Nehalem over an  optimized scalar version running on a single core. | | The advantage is the challenge  for providing  improvised  health care  efficiently,  which is  complicated by the  scale & size  of the data.  Despite the  availability  of various  general  purpose and  specialized  rendering  engines, the medical community has not yet widely adopted the volumetric visualization except in | | 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 widely adopted by the medical community except in certain specific cases | |  |
|  |  |  |  |  |  | |  |  | Certain specific  cases. In ray-casting, as the ray traverses  through the volume, they access voxels with a non-  constant stride. | |  | |  | |
| 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  trade-off  between efficient  memory usage  and access  performance | Present a new sparse volume  hybrid data  representation  (JiTTree) that  makes it possible to combine the  various  elemental data structures  resulting in  lower memory  usage while enabling  high runtime  performance.  No memory  Fetches & swaps are  required for the kd-tree traversal.  The encoding of splitting axis order is directly done in the conditionals  Moreover, the splitting plane  positions are generally  inserted as  literals. | | Stag beetle  dataset  (brick size-  32\*32\*32)  Kingsnake  dataset  (brick size-  128\*128\*128) | This paper  presents JiTTree, a novel & new 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 efficiency in accessing memory  storage and high speed  data access. The JiTTree is the just-in-time  compilation  phase that  transforms data into efficient access functions.  Thereby it  enables the use  of several kinds of data structures for representing data 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  representation 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 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | 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 perform surveys and compares the existing works which employ octrees for  3D volume rendering. It’s  Main focus is specifically on extracting out direct  volume rendering, and  iso-surface ray tracing.  It surveys the varieties of octree available and the efficient hashing schemes for their traversal. | It only examines  Octrees that are of considerable interest in  Field of volume  Rendering only, we are not able to compare its usefulness  towards any particular purpose. |  | Future  applications of volume  rendering using octrees could attempt to combine the pure octree  approach with GPU rendering  approaches  , using out-  of-core  methods |
| 11 | Cell Octrees:  A New Data  Structure for  Volume  Modelling and  Visualization  [11] | 2001 | Conference,  Proceedings of the Vision  Modelling  and  Visualization | An  Improvisation in the bono  approach which uses an  incomplete  octree structure with a relatively  smaller  memory |  | Large Volume  Data | A new indexing method was proposed which uses an incomplete octree structure and requires less storage memory.  It present an improvement of BONO (Branch On Need Octree) approach which uses an incomplete  Octree structure with a smaller memory. The methodology indexes various set of cells as a block when their property value is nearly uniform.  In this method, the prune  condition is totally independent of threshold and the point of view. The tree branches that do not index cells containing the isosurface are not processed, so the building time of iso-surfaces is reduced. | When the number of cells of any dimension  in the volumetric data is not a power of  two, or is not equal to the  number of cells of another  dimension of it,  then nodes will not have all eight children. In such  cases, octree will be having a larger spread in the lower levels i.e. closer to the  leaf nodes, and a  smaller spread in the levels close  to the root node.  Consequently, having smaller storage requirements. | An  improvement of bono,  noted as  octree with cells, has been accomplished | Values of properties may be stored in the octree, so grid storage is not necessary and there may be further  reductions in the storage  constraints. |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | Multi  Resolution  volume  visualization  with a texture -  based octree  [12] | 2001 | Journal  The Visual  Computer  2001 | Reduction in  the amount  of texture  memory  that is required for rendering a 3D volume  data, and  thus reducing the texture  loading  overhead of the rendering engine. | 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 prime focus of this paper is to speed up the texture based  rendering of volumetric datasets. The paper do propose a new texture memory representation & some management policy that was able to substitute the classical one- texel per voxel concept for a  hierarchical approach. This method benefits nearly homogeneous  regions and regions of somewhat lower interest. The  proposed algorithm is  totally based on a simple traversal of the octree representation of large-scale Volumetric data  Driven by a user-  defined image  quality, defined  as a combination of data homogeneity and  prime importance, a set of octree nodes are selected to be rendered on the fly. The degree of accuracy that is applied to represent each one of the nodes of the cut in the  texture memory is basically set  independently  as per the user-defined parameters. | The algorithm that is proposed does not  consider view-  dependent criteria (which are based on the  perspective  distortion and  the shrinkage  of farthest of data  sections)  as, in 3D volume rendering, the  differences in the ratio of projections are very limited. This paper is basically focused on  The speedup of the 3D  volume  Visualization in  so the other  attributes  (i.e. image  quality) are  considered as a secondary  choice |  | "Future work” is Addressed  So to analyze  The various  approaches for selecting the interest Function which is able to better Satisfies  user-Defined  Constraints viz, Problem of the Combination of surface and  the Volume  Information |
| 13 | Enabling the interactive display of large medical volume datasets by multiresolution bricking | 2009 | Journal Springer Science | To implement a bricked, hierarchical, out-of-core partition-based strategy to balance the usage of main (CPU) memory and external (GPU) memory | Retrieving data present in a discrete multi-resolution model, along with a bricking technique combined with it. | Visible Human Male (Size-3.15 GB)  (1760 x 1024 x 1878)  Visible Human Female  (Size-12.03 GB)  (2048 x 1216 x 5186) | The purpose of research was to find out a significant faster approach to 3d visualize large datasets using multiresolution bricking & hierarchical data structures like octree | Approach was not able to compress datasets in presence of noise |  | Further can be extended for exploiting the use of multithreading in the GPUs to speed up the pre-processing. Also possible to further compress the pre-processed bricked layout data |

**4. Methodology**

The visible 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] & [13] are our base research paper as they introduces a novel method to process the large volume data efficiently.



**Figure 1: *Flowcharts of our approach (left) and the previous out-of-core GPU 3D volume rendering approach (right).***

***Yellow boxes are executed on the CPU, all other boxes on the GPU. Control flow is indicated by the black arrows & the output of intermediate data is indicated using green arrows.***

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 Bres 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 dataset can be represented as a multi-resolution hierarchy maintained on the CPU. Each node of the octree points to a particular brick having a constant resolution that resembles the part of the volume corresponding to that particular node.

Pre-processing steps comprises of 3 main steps:

1. Sampling
2. Bricking
3. Compression

Bricks store extra overlapping voxels, which helps in accessing the neighbouring 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 for storing the min-max scalar values for each of the corresponding bricks, for efficient brick culling. The dimension for macro-cell is Mres

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 every 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. Structure 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. Generation of a View-Dependent Working Data-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-portion is used for updating the GPU pool of bricks 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 Brick Sorting**

The nodes in the cut-portion are rasterized as 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 node that is traversed is always replaced by its children to front-back order during depth-first search, this is done till a node labelled as node cut-portion is found. It is implemented using an 8 \* 8 table for lookup purpose. Each row of the look up table is able to encode a possible order of all the 8 octants according to the viewpoint in the octree space.

**4.2.2. Memory Management of the Working Dataset**

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

* **Brick Pool**

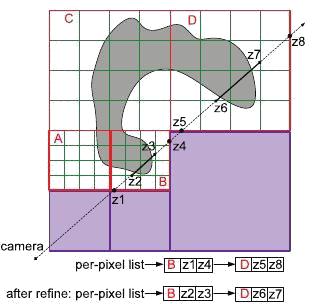
It is organized ad a 3D texture of specific size and dimension. Each of the cell of this pool corresponds to a particular brick, and the cell stores the corresponding brick Id BID.

* **Macro-cell Pool**

The 3D macro-cell pool is packed with the corresponding brick’s macro-cell at particular brick id BID.

**4.3. Proxy-Geometry Rasterization**

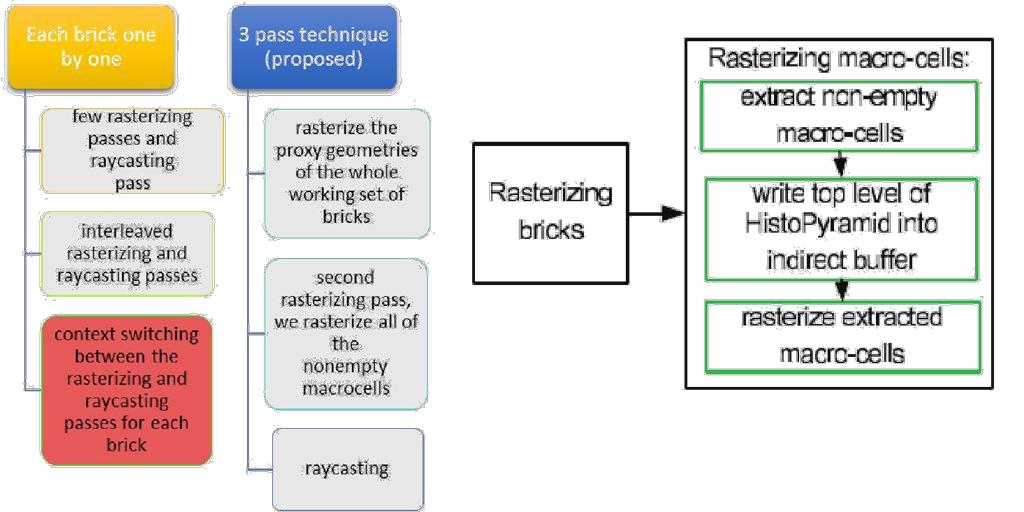
The comparison between 2 pass and normal rasterization is shown in Fig 3. The explained process of two pass rasterization could be seen in Fig 4. In the first pass, rasterization of the proxy geometries of the whole working set of bricks is performed and all bricks are captured that a ray penetrates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



**Figure 2: *Philosophy of Approach: The cut-portion is composed of all of the active bricks (red—A, B, C, and D) at different ROIs depending on the current view. Each active brick is subdivided into macrocells which are green coloured. Empty bricks of purple colour are never added to the octree-cut. We are firstly rasterizing active blocks so that the active blocks can be captured by the fragment shaders that the ray penetrates (B and D) into a list in front-to-back order. Then we are rasterizing the nonempty macrocells and refining 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 macro cell level***.

**4.4. Raycasting**

For each desired image pixel, a ray is generated. Using a simple camera model, the ray originates at the centre of projection of the camera and traverses 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 extreme end points of the ray-segment that performs the GPU raycasting for that corresponding brick.



***Figure 4: Flowchart of rasterization***

***Figure 3: Comparison between 2 pass rasterization***

***Process and normal method***

**4.5. Interactive Visualization**

Once we are ready with the pre-processed data organised in the form of multibricks in octrees, we need to interactively visualize it. Keyboard as well as Mouse both functionalities are incorporated to enhance UI features.

**4.5.1. Keyboard Approach**

In this method, various keystrokes are devised to interactively visualize the datasets. Initially the root file of pre-processed data organised in octree is loaded, which is of lowest resolution. Now, from this we can visualize any portion of body using these keystrokes:

**Key ‘0’:** Upper Right Front

**Key ‘1’:** Upper Left Front

**Key ‘2’:** Upper Right Back

**Key ‘3’:** Upper Left Back

**Key ‘4’:** Lower Right Front

**Key ‘5’:** Lower Left Front

**Key ‘6’:** Lower Right Back

**Key ‘7’:** Lower Left Back

**Key ‘b’:** Back Key to come back to parent node

**Key ‘h’:** Reset Key to come back to root node

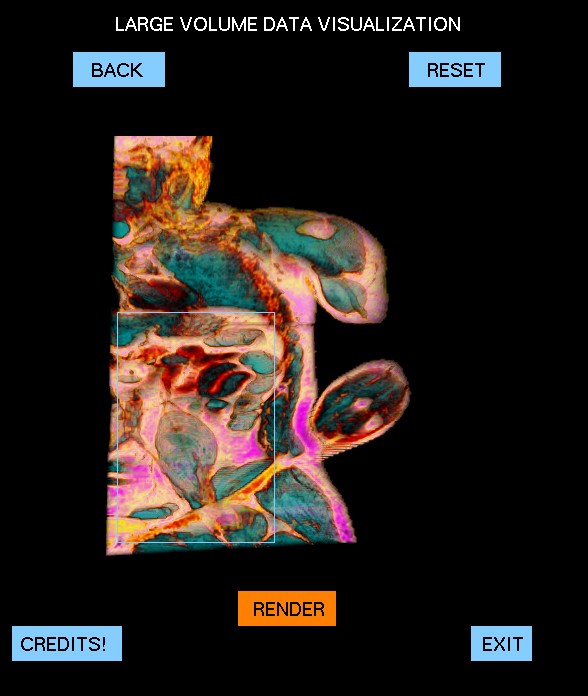
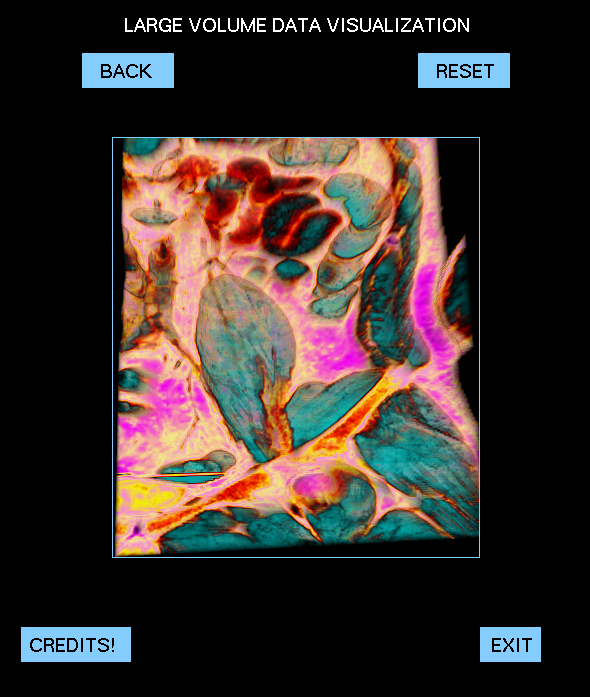
**Key ‘q’:** Exit

**4.5.2. Mouse Approach**

In this, we are using ROI (Region of Interest) selection using mouse, a rectangular box is used to select the same. Once the rectangle is positioned appropriately, we can render the selected portion in higher resolution by selecting the appropriate child node of the current node using octree traversal.

Moreover, Left Mouse click is used to rotate the body throughout the screen & Right Mouse click is reserved for zoom In & Out purpose.

ROI Selection is done using Mouse Scroller Click/ Mid Mouse click.

****Algorithm applied for selection of appropriate file to be rendered on the fly:

Step 1: Identification of Body Orientation using viewRotation matrix

Note: Currently 4 Orientation are supported Front, Back, Right Side, and Left Side

Step 2: Identification of Quadrant in a particular orientation

Using extreme coordinates of ROI selection rectangle viz (Xr1, Yr1), and (Xr2, Yr2) and comparing with the mid points of the cubical box viz (Xm, Ym).

Step 3: Using the selected orientation and the quadrant, appropriate brick is selected from the CPU and

copied to the GPU for rendering purpose.

1. **(b)**

***Figure 5(a): ROI Selection and (b): Rendered Output of ROI in Higher Resolution***

1. **Hardware and Software requirements**
   1. **Software Requirements**
      * CUDA enabled system i.e. nvcc Compiler
      * OpenGL
      * Windows: Visual Studio (8+)
      * Linux: gcc compiler
      * National Library of Medicine Visible Human Body (Male) 3.15 GB Dataset
   2. **Hardware Requirements**



**SPECIFICATIONS**

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

**6. Activity Time Chart**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Work done (Phase 1)** | | | | | | | |  |  |  |  |  |  | **Work done (Phase 2)** | | | |  |  |  |  |
|  | **Week 1:**  **15th May -**  **22nd May** |  | **Week 2:**  **22nd May –**  **29th May** | |  |  | **Week 3:**  **29th May –**  **5th June** |  |  | **Week 4:**  **5th June –**  **12th June** |  |  | **Week 5:**  **12th June –**  **19th June** |  |  | **Week 6:**  **19th June –**  **26th June** |  |  | **Week 7:**  **26th June –**  **3rd July** |  |  | **Week 8:**  **3rd July –**  **10th July** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Literature Survey |  | Environment Setup: OpenGL and Cuda C | | |  | Learning OpenGL and Cuda C | |  | Data Collection & Code Walkthrough of the previous version | |  | Fixing Initial Bugs in the previous version & pre-processing of data | |  | Adding Keyboard functionality and Interactive visualization using octree traversal | |  | Adding UI Support & Mouse Functions. | |  | Extensive Bug-Fixing and Testing |  |

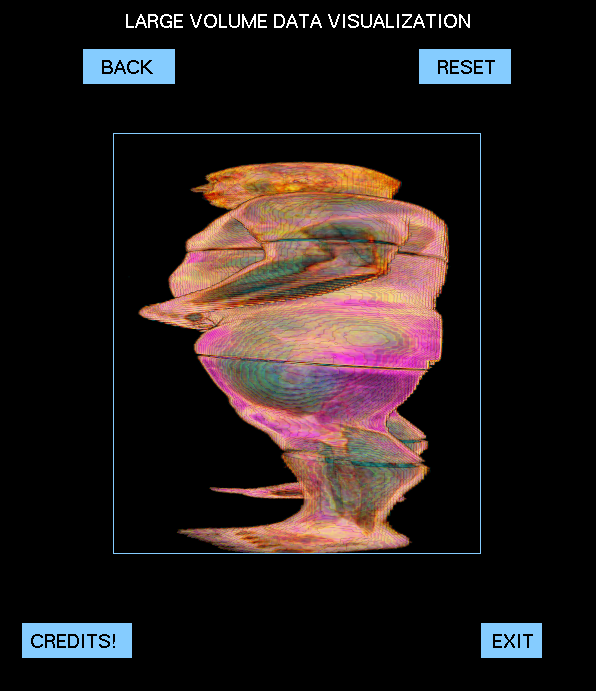
**7. Experimental Setup and Results**

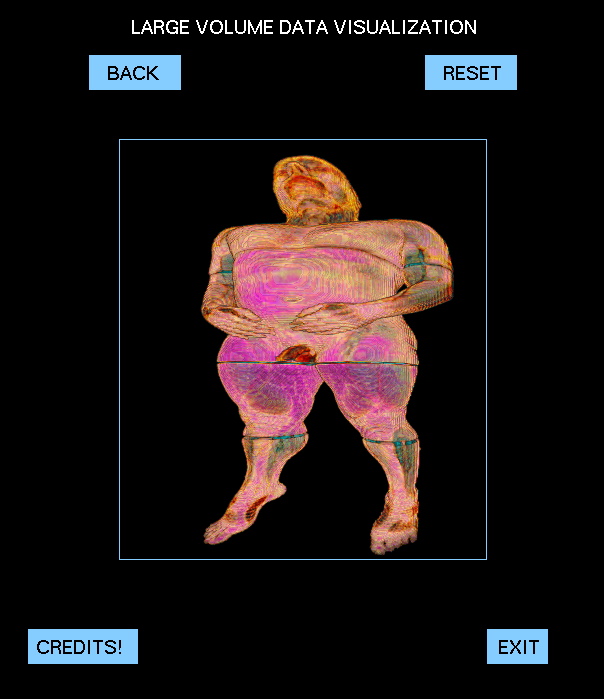
Experimental Setup was as follows:

1. Linux Operating System
2. OpenGL Api
3. Cuda C Enabled
4. gcc compiler to run native C code
5. nvcc compiler to run Cuda C code

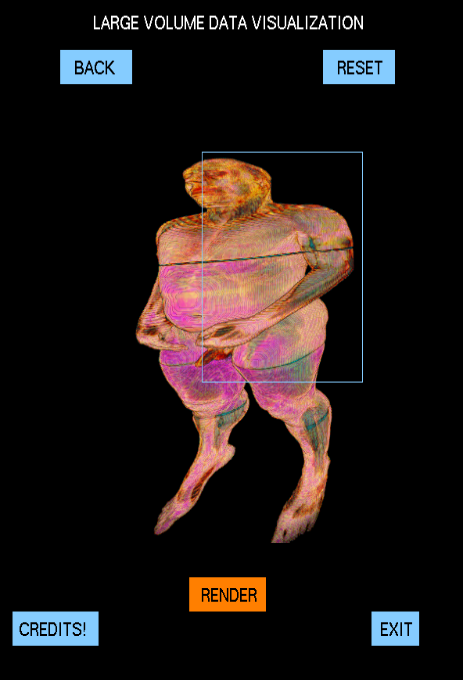
Following are the screenshots of the product for interactive visualization of the medical dataset that we have done so far:-

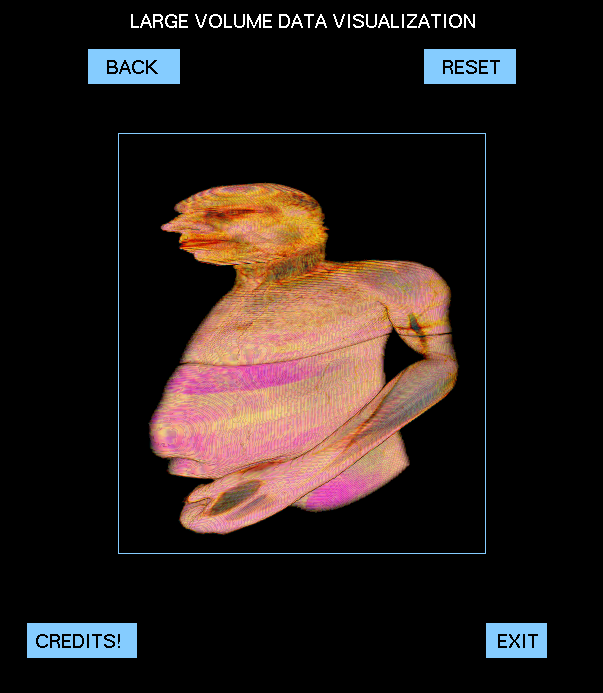
(a) (b)





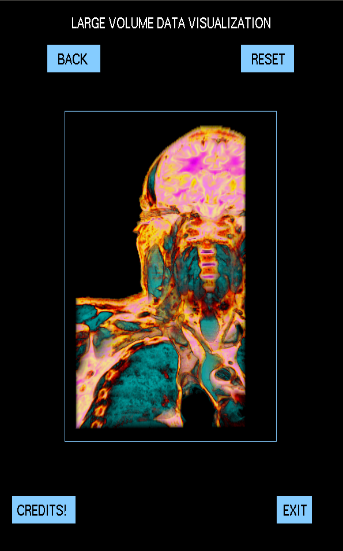
***Figure 6: (a) and (b) are 2 different orientation of root node***

(a) (b)



***Figure 7: (a) Original Portion of root level and (b) Interactively Visualized Portion***

(a) (b)



***Figure 8: (a) Original Portion (b) Interactively Visualized Portion***

**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 Traversing Algorithms using GPU Octree. 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

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Time Require from** | **Time Require from** |
| **S. No.** | **Processor** | **8 to 1 brick** | **64 to 1 Brick** |
|  |  | **construction** | **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 3D volume ray-casting, which moves the branch-intensive octree traversal out of the GPU ray-casting loop and after the traversing completes, execute it on the GPU. By introduction of the tighter depth range for the GPU ray-casting it exercised the greater control over the rendering process by using the hardware rasterization unit performance. This method also offers somewhat more sophisticated & efficient empty space skipping, and also makes possible the use of advanced features like cubic interpolation in an interactive manner for the large out-of-core data sets. Since the presented method provides a general accelerating scheme by rasterizing an octree to generate tight ray segments, and the approach can also be used to improvise the performance of the other visualization & rendering systems, such as in multiple user activities, comparative and multiple volumetric 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.

A viable future scope of this approach is to be able to visualize multiple bricks that are falling in the Reason of Interest (ROI).These multiple bricks may be present at any level of the octree.

1. **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] obbetti . arton and . . uiti 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

1. 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
2. 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
3. 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
4. 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
5. 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
6. 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
7. A. Knoll, "A Survey of Octree Volume Rendering Methods", Scientific Computing and Imaging Institute University of Utah, 2006
8. 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
9. I. Boada, I. Navazo and R. Scopigno, "Multi- Resolution volume visualization with a texture-based octree", Springer-Verlag, pp. 185-197, 2001
10. 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, 2000