3-Dimensional Astronomical Visualisation

Bilal Hasan Aslan
ASLBIL001@myuct.ac.za
University of Cape Town

ABSTRACT

Several applications of astronomical visualization tools demonstrate the importance of interactive visualization in astronomical data exploration. The new generation of radio astronomy telescopes, due to the increased size of the data being captured by them, requires improved software designs and visualization techniques. This is required to maintain or enhance contemporary applications' interactive performance and user experience when dealing with large datasets. This paper presents the development of 3DAVis, a system that renders astronomical data into a 3-dimensional (3D) visualization, using a client-server hybrid rendering approach. Its goal of serving as a proof of concept for future 3D astronomical data visualization tools, where the computationally intensive tasks are handled by the server and the data exploration and interaction are handled by the client. Currently, there are no 3D astronomical data visualization applications which do this that are widely available to the public. In this paper, using our application, we explore various techniques and algorithms which could potentially be included in such future applications and discuss the beneFITS, drawbacks, bottlenecks, and opportunities associated with each.

CCS CONCEPTS

• **Applied computing - Physical sciences and engineering (Astronomy)**;

KEYWORDS

Astronomy, 3D-Visualization, Visualization tools

1 INTRODUCTION

Scientific visualization has been applied in many different branches of science in an effort to support the knowledge discovery process, and one of the branches is astronomy. [5] Visualizing data sets that are gathered by modern telescopes is necessary for astronomers. Astronomy is a visual science that relies heavily on visualisation and interpretation of data, as claimed by Rolowsky et al [12]. Without current visualization tools, astronomers will have a hard time when observing data sets gathered by the telescopes.

Astronomy is a data-intensive science, and already petabytes of observational data are stored in archives. [1, 13] Rendering and interacting with these data sets are very computationally expensive. One of the most significant challenges astronomers face is trying to use 3D visualization tools that are incapable of efficiently processing large amounts of multi-dimensional data.

Some 2D visualization tools can process huge data sets efficiently and certainly are very useful for astronomers, but compared to static 2D renders, 3D renders will give astronomers a more immersive and engaging experience, and allows complex astronomical data sets to be visualized in a way so that it is better understood. It has recently been discovered that there is a lack of an application that can deal with large astronomical data cubes in more than two dimensions. Current 3D visualization tools fail to render these data sets effectively because these tools are not designed to deal with the huge data sets that are gigabytes in size. Besides memory and speed issues, other issues are required to be dealt with when developing a good modern 3D visualization tool such as being real-time interactive when rendered, accurate and fast 3D rendering, and a good user interface.

Another problem with these 2D and 3D visualisation tools is that a computer must store colossal astronomical data on its hardware to be able to render. Instead, having a central data storage and fetching this data from storage to be rendered on the client-side is a very efficient and effective way store these large amounts big astronomical data. This server-client design is implemented by CARTA [4] visualisation tool but CARTA is only capable of 2D rendering currently.

This project aims to solve scalability, memory, and storage issues when rendering large amounts of astronomical data for 3D visualization, by using a server-client hybrid rendering approach.

2 BACKGROUND

2.1 Astronomical Data

Space is usually observed as electro-magnetic radiation within some range of the electromagnetic spectrum. There is a lot to observe and many wavelengths to observe it in [15]. The Astronomical data contains a range of wavelength values and the presence or absence of these wavelengths provides us different information such as the chemical formation of astronomical objects, velocity of the astronomical objects, etc. A visualization tool becomes necessary to understand such data.

Astronomical data is stored in different formats such as FITS, HDF, or ASDF. Around one billion astronomical data that is gathered by astronomical instruments worldwide are stored in FITS format. [7] There are already some tools that convert FITS to other standardized data formats. If another data format needs to be used in a new modern visualization tool because of its speed, concurrency, and other beneFITS, it should not be an issue as long as FITS files can be converted to such format.

2.1.1 FITS File Format. The Flexible Image Transport System (FITS) format is a very generic format and is used to represent a large amount of different data types. There are numerous advantages of FITS file format, some of which are: it is widely used, it is easily understood serialization, and it has extensive documentation. [14] Petabytes of astronomical data are archived in FITS format. [7].

2.1.2 HDF5 File Format. Hierarchical Data Format version 5 (HDF5) file format is complexly structured, can be used to archive large data sets, and supports various types of data sets. HDF5 file format has more capabilities and potentially higher performance over FITS format. HDF5 datasets with a contiguous layout strategy, nearly constant access time to any element in the array and zero overhead for locating elements in the dataset is assured. [6]

HDF5 file format can stream sections of files across multiple spindles and has a very high read/write speed. [10] Nowadays, many astronomers tend to use the HDF5 file format. The Low-Frequency Array (LOFAR) has been archiving its astronomical data in the HDF5 file format because of its speed advantages. [2] HDF5 file format also offers storing data in chunked datasets. Chunked datasets are split into multiple chunks that are all stored separately in the file. Chunks can then be read and written individually, improving performance when operating on a subset of the dataset.

2.2 Volume Rendering

Volume rendering is used in scientific visualization to create 2D projections from a 3D data set. For instance, a series of 2D slice images of astronomical objects can be assembled to render 3D volume rendered images using a volume rendering algorithm.

2.2.1 Direct Volume Rendering. Direct Volume rendering is creating a projected image from multi-dimensional data. Only simple algorithms and no approximations are applied to the data. This type of rendering is best suited to visualize astronomical data. Geometric structures are not created in this type of rendering which results in renders as natural and accurate as possible. Volume ray casting, splatting and shear warp, are the well-known techniques used in direct volume rendering. Ray casting is preferred over other techniques in astronomy because of its higher quality and more accurate render results.

2.2.2 Volume Ray Casting. Volume ray casting is widely used when volume rendering scientific data because of its high-quality results. This technique of rendering also allows parallelization and good user interactivity. [16] Volume ray casting is an intensive process and a single general-purpose CPU is not sufficient to achieve interactivity or even real-time for large data sets. [9] Certain optimization algorithms are used to achieve better speeds like empty space skipping, early ray termination. Visualizing big multi-dimensional data sets with the help of a high-end GPU and also using parallel algorithms may achieve fast render speeds.

3 RELATED WORK

3.1 KARMA

KARMA is firstly designed to visualize data sets in 2D, slice by slice. It is widely used by astronomers today. Some packages allow KARMA to read different astronomical data formats like FITS and Miriad. This tool lacks a good user interface, even doing small tasks on this tool may require high technical knowledge. KARMA developers have introduced a 3D visualization package called KARMA XRAY. This package was not astronomers' preference because its rendering speed limited the tool's interactiveness and had memory management problems with big data sets. [3]

3.2 SAO Image DS9

This is another widely used tool by astronomers around the world. This tool has a better user interface compared to KARMA. SAOImage DS9 supports FITS and many other data formats. [8] This tool has 2D visualization that renders data sets slice by slice and has 3D visualization that allows users to interact with the 3D render. However, this tool suffers performance problems when it comes to rendering 3D visualization because rendering with Graphics Processing Unit (GPU) acceleration is not supported. [8]

3.3 CARTA

CARTA is a modern 2D visualization tool that offers an user friendly interface and fast rendering speeds. CARTA functions as a server-client system where CARTA is installed on a high-end server and astronomers connect to this server from their typical workstations. After all the processing on the server is completed, render data is sent to the client where the render gets processed preferably by GPU and is displayed to the user. This means having a GPU on the client's workstation increases the rendering performance of this tool. Server-client interaction created a lot of advantages on this tool and this same type of technique could be used for 3D visualization.

Astronomers do not always have good access to sufficient computational power or data-storage capacity to deal with large data sets. Dealing with large data sets effectively and efficiently requires a higher level of computing knowledge relating to the choice and use of appropriate data structures, techniques for scheduling, and so on. Therefore using server-client presents an opportunity to provide the wider astronomy community with a reliable, efficient visualization service with potentially lower cost, less administrative effort, and a reduced need to transfer data. [11, 17]

3.4 VTK

Standing for The Visualisation Toolkit. It is maintained by Kitware and is open source under BSD license. It was created for the purpose of displaying and manipulating scientific data in a browser. It supports volume rendering, scientific visualisation and two dimensional plotting. VTK.js is VTK's accompanying JavaScript library for rendering graphics in a browser and is available as packages on NPM for easy integration with a JavaScript framework. VTK.js is supported on Google Chrome and Firefox but not yet on Safari and Microsoft Edge. Having VTK as C++ library on server and VTK.js on client side brings a huge advantage for this project's purpose since it will make server and client interaction simpler.

3.5 Requirement Gathering and Analysis

3.6 Requirement Gathering

Throughout the project, we worked with the lead developer of CARTA, Dr. Angus Comrie. He had a particular interest in 3D visualisation using the server-client hybrid rendering approach, which is the main motive behind our project. The research questions we wish to answer are:

(1) How can the data be pre-processed using methods such as mipmapping on the server-side of the system to increase the speed of data processing and transfer?
(2) Does the graphical rendering library VTK produce the best results in terms of speed and accuracy when rendering?
(3) How can the client and server approaches be combined to facilitate high-resolution data rendering over a network while maintaining usability?

(4) How does network latency between server and client affect when rendering large volumes of data using the hybrid rendering model?
(5) How does the hybrid model approach scale in terms of rendering time, transfer latency, and time from user interaction and feedback over increasingly larger data sets?

These research questions suggest that speed and efficiency are the key aspects to consider when designing and developing the system. This system needs to provide a smooth experience for a user whilst dealing with large astronomical data. Figure 1 proposes an initial sequence diagram on how the system should function at deployment. This basic system flow was changed and improved to develop a final product.

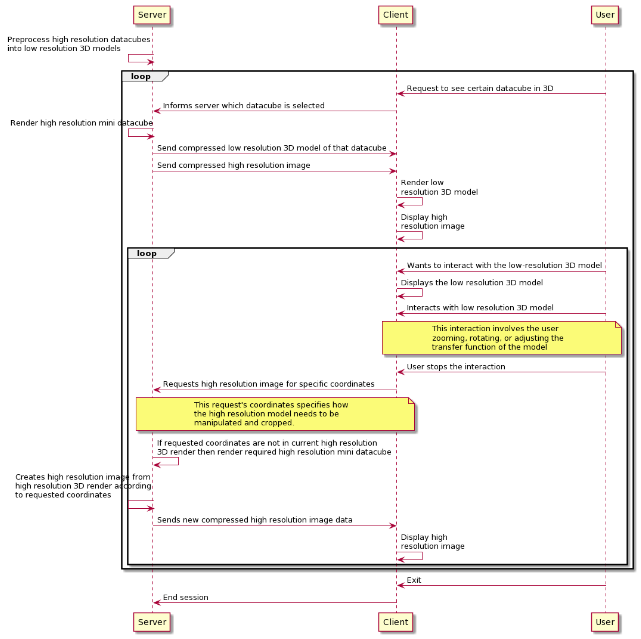
Server
Client
User
Preprocess high resolution datacubes into low resolution 3D models
loop
Request to see certain datacube in 3D
Informs server which datacube is selected
Render high resolution mini datacube
Send compressed low resolution 3D model of that datacube
Send compressed high resolution image
Render low resolution 3D model
Display high resolution image
loop
Wants to interact with the low-resolution 3D model
Displays the low resolution 3D model
Interacts with low resolution 3D model
This interaction involves the user zooming, rotating, or adjusting the transfer function of the model
User stops the interaction
Requests high resolution image for specific coordinates
This request's coordinates specifies how the high resolution model needs to be manipulated and cropped.
If requested coordinates are not in current high resolution 3D render then render required high resolution mini datacube
Creates high resolution image from high resolution 3D render according to requested coordinates
Sends new compressed high resolution image data
Display high resolution image
Exit
End session
Server
Client
User

Figure 1: Proposed sequence diagram for initial system design.

3.7 Design Methodology

An incremental prototyping model approach was implemented, where each team member builds their own prototype for their section. Once each member's developed their prototypes, it was merged and tested again before being deployed. Figure 2 illustrates this process. The project was split into three independent sections, one for each team member, which made this the most appropriate approach since it provided freedom of individual coding.

The server was developed using a feature-driven approach, where each feature was developed one after the other was complete. Once a feature was completed it was merged into the prototype and tested for bugs. This made testing process very easy and effective for determining anomalies in the code.

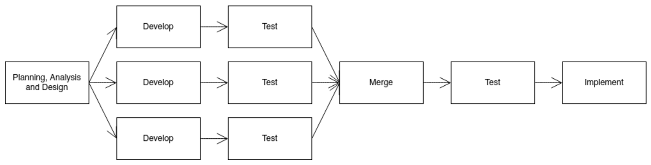

Figure 2: Incremental prototyping model.

4 BACKEND IMPLEMENTATION

Astronomical data are huge in size. When working with such large data sets, it is crucial that memory management is prioritized for scalability as well as obtaining fast process speed for better server response time. For this reasons, This is why C++ was chosen to implement the server; it provides good memory management and is computationally efficient. C++ is also widely utilized in computer graphics applications, scientific or, otherwise and also web applications' servers that require fast response times. One of our primary motives for this project is to build a proof of concept for CARTA which is also built in C++ language.

The Server will be using VTK (Visualisation toolkit), which is a C++ library, for volumetric rendering high resolution data on the server side and encoding it to JPEG (Joint Photographic Experts Group) picture format that will be sent to client side. This library was chosen because it is widely used in scientific volume rendering.

4.1 File Preprocessing

Disk reading is a slow process in computers. Reducing this bottleneck created from disk reading will increase the speed of the server dramatically. FITS file format only provides sequential reading from disk while HDF5 file format provides parallel reading and chunking features. Reading data through HDF5 file format will increase servers' response speed resulting in a more efficient astronomical rendering system.

Mipmaps are different levels of low-resolution cubes of the data. Another way to speed up server performance is by pre-processing these mipmaps before the server and client renders it. HDF5 file format can store different datasets in its file format, allowing it to store pre-processed mipmap datasets. These mipmap datasets are stored in chunks, each of which contains one cubelet. These cubelets dimension sizes are 64x64x64 which makes them 1MB in size. This could be easily changed in the code if different chunk sizes are required.

Since most of the astronomical data is stored in FITS format, a Converter is needed to convert FITS files to the desired HDF5 file format. Figure 3 illustrates the process of the converter which converts a FITS file format into a HDF5 file format.

4.2 Back-end disk reading

To achieve maximum potential reading speeds from a disk, especially SSD, we need to read data from disk in small blocks such as 512Kb, 1MB and 2MB. An experiment was conducted to test and analyze what size data will achieve maximum reading speed from the disk. The system was built to allow cubelet sizes and chunk sizes to be easily changed, according to the chosen cubelet size during pre-processing FITS files to HDf5 format.

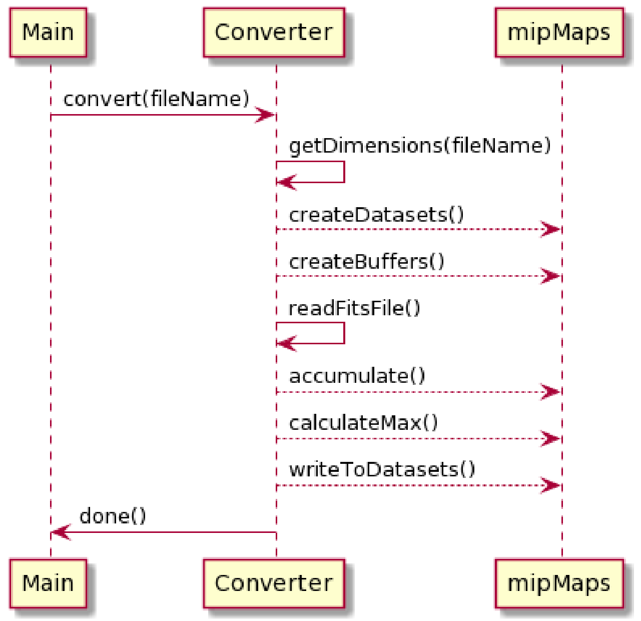

Figure 3: FITS to HDF5 Converter sequence diagram.

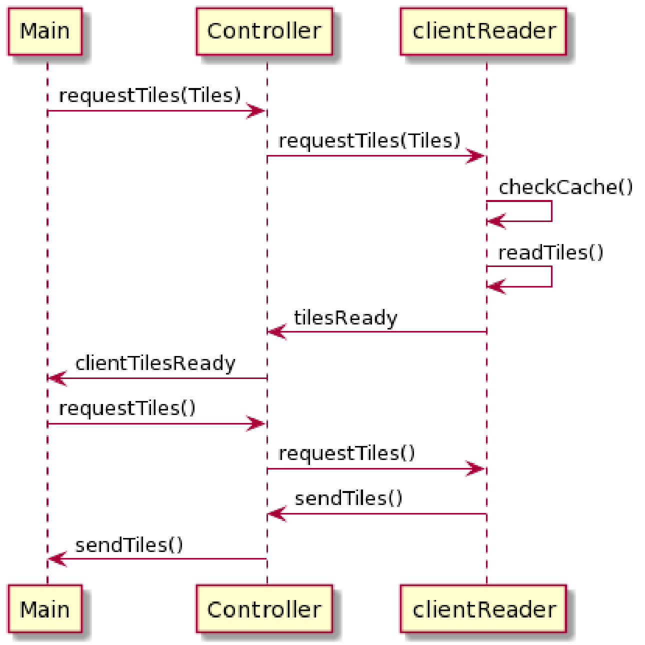

Figure 4: Sending requested cubelets to client.

When the server needs to read from the disk, it will first check the memory cache and read cubelets that are not cached but required. All cubelets that are read will have a unique ID representing their position in the mipmap dataset. These cubelet's dimensions and also their content are stored in cache memory. If any cubelet is requested for rendering again, it is quickly retrieved from cache memory, which speeds up the server response.

4.2.1 Reconstructing cubelets. While only up to maximum of 4 cubelets are sent to client side for client to render, the server deals with more than 1000 cubelets. These cubelets needs to be combined together to create one big cube. Right now these cubelets are combined row by row but for future work these cubes needs to be combined as a whole to achieve best speed results.

4.2.2 Sending cubelets to client. When the User crops a cube visualization on the client-side, the client requests new cubelets from the server to render a higher resolution of the cropped section. The server first checks the memory cache and reads missing cubelets from the disk. After all cubelets are in memory it is compressed and sent to clients one by one, as illustrated in figure 4. These cubelets are decompressed on the client-side, reconstructed to a larger cube, and then this cube gets rendered.

4.2.3 Rendering new volume from crop coordinates. When the user crops the cube on the client-side, the client sends crop points to the server. Once the cube is cropped, the server can render a higher resolution of the cropped section. The server calculates new cubelets needed from these crop points, reads if these cubelets are not cached already, reconstructs cubelets to get a single large cube, and then renders this higher resolution cube, as illustrated in figure 5.

4.2.4 Encoding Image. The high-resolution images on the server side are encoded using VTK library. VTK library uses FFMPEG to

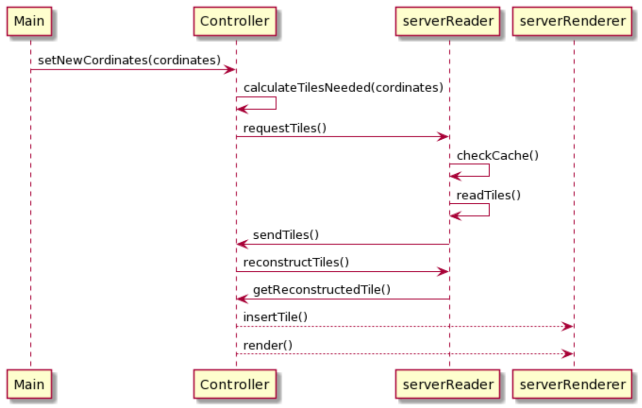

Figure 5: Rendering new volume from crop coordinates.

encode these images. FFMPEG is a free and open source software project consisting of a large suite of libraries and programs for handling video, audio, and other multimedia files and streams. The image is encoded directly to servers memory and then it is directly sent to client from the memory (figure 6).

FFMPEG is not the fastest encoder library. FFMPEG is limited to encoding on CPU whilst Nvidia's NVENC library can do encoding on GPU. But VTK does not support NVENC which makes it hard and time consuming to implement. Due to time constraints I was not able to implement this library. However results gathered from testing suggest using NVENC instead of FFMPEG is not going to results very noticeable performance difference in terms of servers response speed.

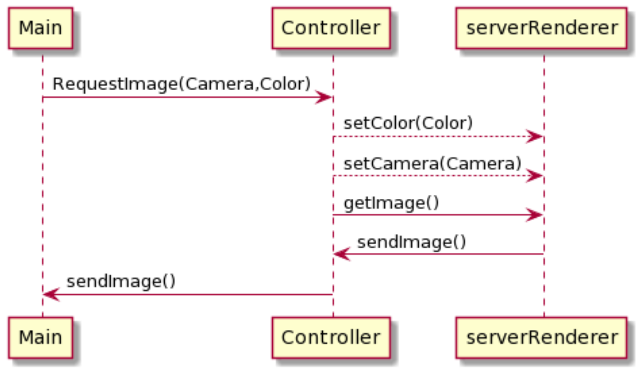

Figure 6: Encoding and sending image.

4.2.5 User choosing a new astronomical file. When a user chooses a new file to be rendered from the server's file list, a client sends the selected file to the server. The server determines which cubelets need to be read from the disk and sends them to a client. After client cubelets are sent the server determines which cubelets need to be read from the disk to produce a higher resolution render. Once the reading is complete, these cubelets are reconstructed into a single large cube. This large cube is rendered in volume with the help of the VTK library. As soon as rendering is complete, the front view of the rendered volume is encoded to image format and sent to the client. This process is illustrated in figure 7.

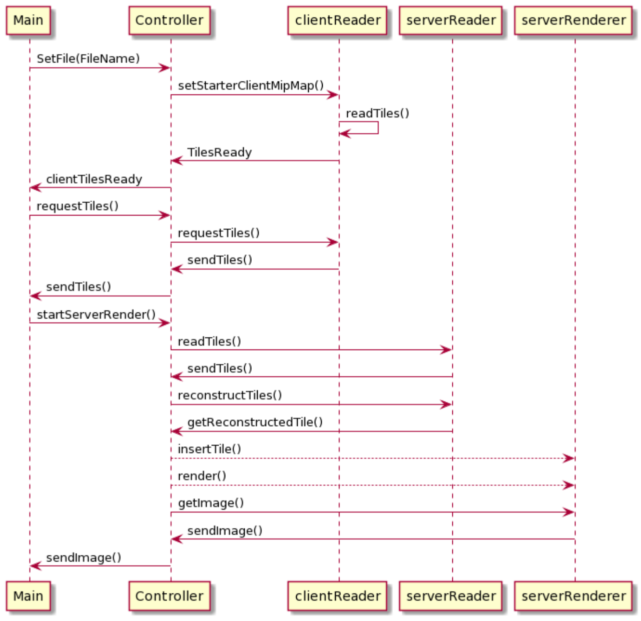

Figure 7: User choosing a new astronomical file.

5 TESTING THE SYSTEM

The aim of this project is to get maximum speed server response times to client requests. The performance tests were conducted on different parts of the code to discover bottlenecks. Finding bottlenecks will help further improve the code design. The code was tested on both HDD (Hard disk drive) and SSD (Solid State Drive: the speeds were recorded to determine the maximum read speeds of each. Thereafter, the speed of the reconstruction of cubelets into one large cube was tested. The rendering speed of the VTK library was tested on different volume sizes and finally speed of encoding this volume to image.

The specifications of the laptop used to conduct these tests are Intel Core i5-7300HQ, GTX 1050 mobile. Benchmark read speeds for HDD that have been used in testing were 54.3 MB/s for 512K block reads, 56.0 MB/s for 1MB block reads and 55.5 MB/s for 2M block reads. Benchmark read speeds for SSD that have been used in testing were 470 MB/s for 512K block reads, 475 MB/s for 1MB block reads, and 471 MB/s for 2MB block reads. All these tests were conducted 10 times and an average of the results was taken. Before each test system cache was cleaned.

6 RESULTS AND DISCUSSIONS

6.1 Reading astronomical data from disk

Before doing disk reading tests, various sizes of FITS files were converted to HDF5 files. During conversion, the mipmaps were chunked in different cubelet sizes. For this testing all 250MB, 500MB, 750MB, 1000MB and 2000MB FITS files were converted to HDF5 Files 3 times, one time chunking in 512K, one time chunking in 1MB and last time chunking in 2MB cubelets.

Figure 8 shows test results when cubelets are read from HDD. Reading in a block size of 1MB gave the best speed result but the time difference between other cubelet size reads is unnoticeable. Let's take the speed of reading 1000MB file in 1MB cubelets because the server will be set to render 1000MB data. From benchmark results, with a speed of 56.0 MB/s, it will take 17.85 seconds to read this file. The test result for servers' read speed is 17.89 seconds (Appendix 1). This means we are using HDD's reading speed to its full potential.

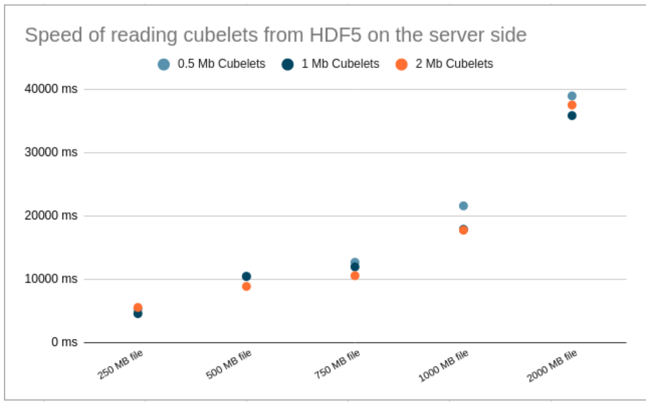

Figure 8: Test results for reading server cubelets from HDD.

Figure 9 shows test results when cubelets are read from SSD. The best reading speed was achieved when reading cubelets in the size of 1MB. This time reading speed difference of 1MB compared to other cubelets sizes was more noticeable. From benchmark results, SSD reads 475 MB/s the block sizes of 1MB. It will take 2.1 seconds to read 1000MB with a speed of 475 MB/s. The test result for servers' read speed is 2.6 seconds. This means the server is using SSD's reading speed to its full potential. Server reading speed is slower since the server is calculating cubelets' coordinates in the file and allocating memory for it.

If the server runs on SSD, the HDF5 file should be chunked into 1 MB chunks, and datasets should be read in 1MB cubelet sizes to get the most from the SSD.

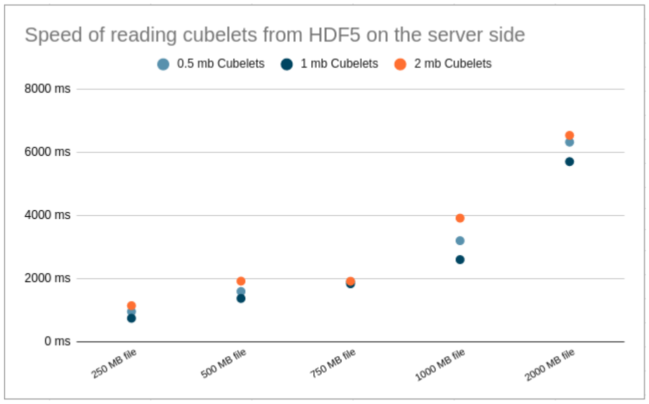

Figure 9: Test results for reading server cubelets from SSD.

The servers' response time when a client requests new cubelets was tested, on an HDD and SDD machine. Results for these tests are shown in figure 10. The tests were done up to a maximum of eight cubelets, even though a client should be requesting a maximum of four cubelets. This was done to experiment if eight cubelets are viable for this system. Reading eight cubelets from HDD took 125 ms, reading four cubelets took 62 ms (appendix 1). 122 ms will result in slow response speed from the server, making four cubelets a better option for the system if a file is stored on HDD on the server-side. Reading eight cubelets from SSD took 27 ms, and reading four cubelets took 14 ms (Appendix 1). 27 ms is an acceptable time so if there are no other bottlenecks caused for using eight cubelets, eight cubelets could be sent to the client to get a better-resulting to render on the client-side if a file is stored on SSD on the server-side.

6.2 Reconstructing cubelets to one big cube

Drawback of reading small cubelets to achieve maximum read speed from the disk is these cubelets needs to be reconstructed to one big cube for VTK library to be able to do rendering. This process delays servers response speed to client when sending high resolution image if new cube needs to rendered.

From figure 11 we can see that it takes longer to reconstruct when we have more cubelets. Let's focus on reconstructing 1000MB data because that's what has been focused on for this project. There is no huge time difference between when using 2MB cubelets and 1MB cubelets. The reasoning behind this is cubelets are not always the desired size. 1MB cubelets dimensions are set to be a maximum of 64x64x64, but when it comes to edges of the data, these cubelets are

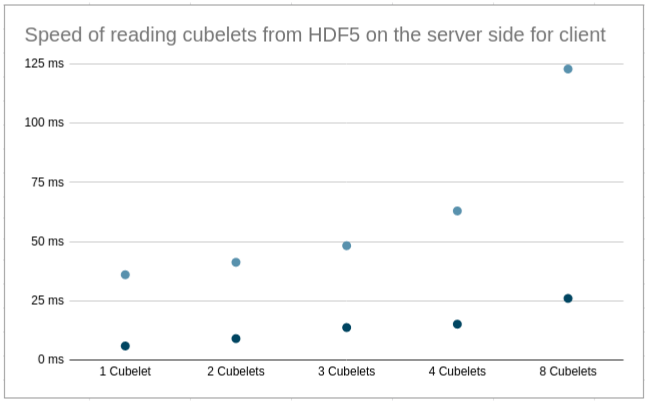

Figure 10: Test results for reading client cubelets from HDD.

getting smaller, so we end up with more cubelets than theoretical calculations. For 1000MB data, we expect to see 1000 cubelets in sizes of 1MB, but from the testing results in appendix 3 there are 1125 cubelets, and for 2MB there are 675 cubelets. This is the reason speed results of 1MB are not twice of 2MB cubelets.

Reconstructing cubelets for 1MB in 1000MB data in testing took 10.6 seconds. This suggests that reconstructing the algorithm that has been used might bottleneck the server, so a better algorithm needs to be designed and implemented. The code design chosen for this part of the system was not optimized enough because of the time constraint and complexity of this algorithm.

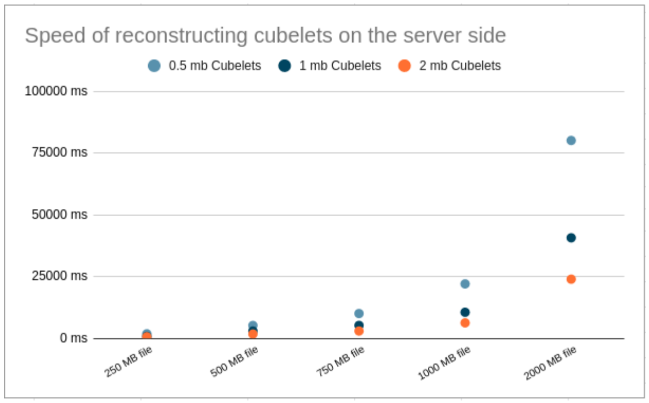

Figure 11: Reconstructing a big cube from small cubelets.

6.3 Rendering with VTK library

Rendering volume is a computationally expensive process. From the figure 12, the test results provide that rendering time increases linearly to data size. Rendering 1000MB data in volumetric took 12.3 seconds. This is slower than what was expected but the issue on the server could not found.

Rendering is only performed once a client establishes a connection and the user crops the cube. This is unlikely to happen often and time consumed on the client-side for receiving new cubelets and rendering them on the client-side should cover some part of it. Users will be able to still use low-resolution render on the client-side to get a better camera position, and the server will send high-resolution images according to the new camera view as soon as rendering is complete.

This part of the server should not be overlooked. The reasoning for slow rendering speed must be found in the design and needs to be optimized.

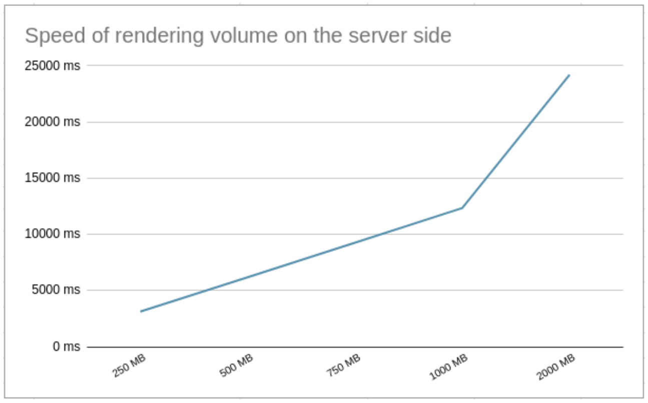

Figure 12: Volumetric rendering using VTK library.

6.4 Encoding volume render to image

Once the volumetric rendering is done on the server side, the server sends a high-resolution image to a client. The resolution chosen for the testing was 1920x1080 to give the user a better experience in terms of quality. The time taken to encode the images from volumetric data is constant over all different render sizes, as seen in figure 13. The average time it took for the server to encode the image is 71.9 ms (Appendix 5). This is in an acceptable time range, but further improvement is needed by implementing hardware-accelerated encoding.

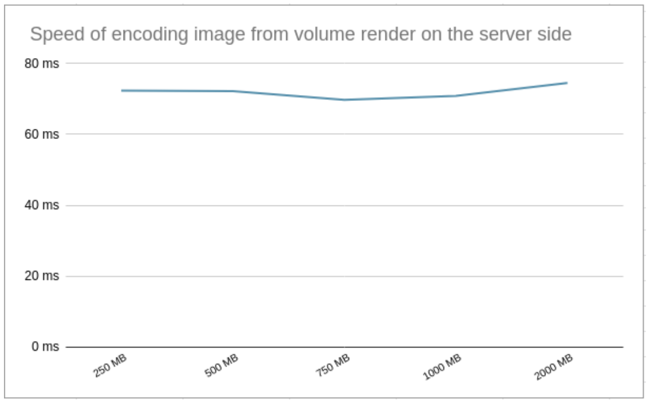

Figure 13: Encoding Image from volumetric render.

7 SCALABILITY

This server is designed to always render data in the initialized memory size. After sending cubelets to a client to render, the server scales up a clients' render dimensions according to initialized size. It then finds the correct mipmap dataset to render from, reads the data, and renders it. This allows the server to render any astronomical data size. Converting bigger astronomical data sizes in FITS format to HDF5 file format is a slow process. Our converter does not scale quite well and it is restricted to the PC's memory size. Also when converting new HDF5 file format increases the storage size of the Astronomical data by more than 100%. A better converter design needs to be implemented if ever this proof of concept needed to be taken further in development.

8 LIMITATIONS

Despite the beneFITS of adopting an incremental prototyping model approach, there were noticeable constraints. Merging individual prototypes was time-consuming. Once merged, new bugs and inconsistencies were introduced into the system and due to time constraints, these could not be fixed.

9 CONCLUSION

Our proof of concept application, 3DAVis, allows users to interact with 3-dimensional astronomical data cubes in an intuitive manner that would otherwise be too large to be stored or rendered on every PC or transported over a local network in an acceptable amount of time. This was achieved by a hybrid-rendering technique vastly unexplored in the widely available 3-dimensional astronomical data visualization applications, by using a client-server architecture where the computationally expensive tasks are handled by the server and the data exploration and interaction are handled by the client. We conclude that future astronomical visualization applications, which wish to allow the visualization of and interaction with the large astronomical data cubes currently being produced by modern radio astronomy telescopes in 3D, should employ a client-server architecture and a similar hybrid-rendering technique.

Test results showed that some parts of this server design are bottlenecking other parts of the system. This server design achieved very high disk reading speeds, acceptable image encoding speeds, and reconstruction of cubes and rendering of the cube needs to be investigated further.

Pre-processing increased servers' response time in some parts of the system. Complete disk read capacity was reached, and creating a lower resolution render for the client-side was done beforehand. Load on the server decreased, extensively improving servers time.

Rendering with the VTK library resulted in very accurate and good renders, but in terms of speed it did not live up to its potential. This could be because of a fault in code design and needs to be investigated further.

Encoding full HD images only took around 70 ms. This speed is good enough and maintains good usability on the client-side. This could be improved further using Nvidia's NVENC library instead of using the FFMPEG library.

Network latency with the addition of servers response time is quite high in some parts of the design. Currently, the system is usable, however, bottlenecks need to be optimized to get better usability from the server-client hybrid approach.

Rendering time and sending pictures scale well with this system. Server always renders required amount of data size not exceeding initialized memory size usage by scaling up and scaling down the render.

Several features which were presumed to be included but were not to due to time constraints were: better FITS to HDF5 converter, better cubelet reconstruction algorithm, hardware-accelerated encoding of images on the GPU.

ACKNOWLEDGMENTS

I sincerely appreciate Prof. Rob Simmonds for being our supervisor and guiding us throughout the project. I would also like to extend my deepest gratitude to Dr. Angus Comrie, the lead developer of CARTA, who assisted in answering our day-to-day questions and directing us to related educational material to further our understanding of topics we were covering.

REFERENCES

[1] James Abello, Panos M Pardalos, and Mauricio GC Resende. 2013. *Handbook of massive data sets*. Vol. 4. Springer.

[2] Anastasia Alexov, Pim Schellart, Sander ter Veen, M Van den Akker, L Bähren, Jean-Mathias Grießmeier, JWT Hessels, JD Mol, GA Renting, J Swinbank, et al. 2012. Status of LOFAR data in HDF5 format. *Astronomical Data Analysis Software and Systems XXI* 461 (2012), 283.

[3] David G Barnes, Christopher J Fluke, Paul D Bourke, and Owen T Parry. 2006. An advanced, three-dimensional plotting library for astronomy. *Publications of the Astronomical Society of Australia* 23, 2 (2006), 82–93.

[4] Angus Comrie, Kuo-Song Wang, Shou-Chieh Hsu, Anthony Moraghan, Pamela Harris, Qi Pang, Adrianna Pińska, Cheng-Chin Chiang, Rob Simmonds, Tien-Hao Chang, et al. 2021. CARTA: Cube Analysis and Rendering Tool for Astronomy. *Astrophysics Source Code Library* (2021), ascl–2103.

[5] Tim Dykes, Amr Hassan, Claudio Gheller, Darren Croton, and Mel Krokos. 2018. Interactive 3D visualization for theoretical virtual observatories. *Monthly Notices of the Royal Astronomical Society* 477, 2 (2018), 1495–1507.

[6] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. 36–47.

[7] Amr Hassan and Christopher J Fluke. 2011. Scientific visualization in astronomy: Towards the petascale astronomy era. *Publications of the Astronomical Society of Australia* 28, 2 (2011), 150–170.

[8] W A Joye and E Mandel. 2003. New features of SAOImage DS9. In *Astronomical data analysis software and systems XII*, Vol. 295. 489.

[9] Jens Kruger and Rüdiger Westermann. 2003. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.* IEEE, 287–292.

[10] Jessica Mink, Robert G Mann, Robert Hanisch, Arnold Rots, Rob Seaman, Tim Jenness, Brian Thomas, and William O'Mullane. 2014. The past, present and future of astronomical data formats. *arXiv preprint arXiv:1411.0996* (2014).

[11] Peter J Quinn, David G Barnes, Istvan Csabai, Chenzhou Cui, Francoise Genova, Bob Hanisch, Ajit Kembhavi, Sang Chul Kim, Andrew Lawrence, Oleg Malkov, et al. 2004. The International Virtual Observatory Alliance: recent technical developments and the road ahead. In *Optimizing scientific return for astronomy through information technologies*, Vol. 5493. International Society for Optics and Photonics, 137–145.

[12] E Rosolowsky, J Kern, P Federl, J Jacobs, S Loveland, J Taylor, G Sivakoff, and R Taylor. 2015. The cube analysis and rendering tool for astronomy. *Astronomical Data Analysis Software an Systems XXIV (ADASS XXIV)* 495 (2015), 121.

[13] Alexander Szalay and Jim Gray. 2001. The world-wide telescope. *Science* 293, 5537 (2001), 2037–2040.

[14] Brian Thomas, Tim Jenness, Frossie Economou, Perry Greenfield, Paul Hirst, David S Berry, Erik Bray, Norman Gray, Demitri Muna, James Turner, et al. 2015. Learning from FITS: Limitations in use in modern astronomical research. *Astronomy and Computing* 12 (2015), 133–145.

[15] Ivan Viola and Helwig Hauser. [n.d.]. Interactive visual analysis and exploration of astrophysical data. ([n. d.]).

[16] Lee Westover. 1989. Interactive volume rendering. In *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*. 9–16.

[17] Roy Williams and Dave De Young. 2009. The role of the virtual observatory in the next decade. *astro2010, The Astronomy and Astrophysics Decadal Survey* 201 (2009).

Disk speed for reading client cubelets from HDD	
1MB Cubelets	
Tile Number	**Average**
1 Cubelet	**40.16 ms**
2 Cubelets	**39.93 ms**
3 Cubelets	**51.14 ms**
4 Cubelets	**62.13 ms**
8 Cubelets	**122.20 ms**

Disk speed for reading client cubelets from SSD	
1MB Cubelets	
Tile Number	**Average**
1 Cubelet	**5.13 ms**
2 Cubelets	**8.39 ms**
3 Cubelets	**11.29 ms**
4 Cubelets	**14.25 ms**
8 Cubelets	**27.24 ms**

Appendix 1. Disk reading speed results for client's cubelets.

Test speed of reading for server cubelets from HDD						
File Sizes	0.5MB Cubelets		1MB Cubelets		2MB Cubelets	
	Tile Size and Number	**Average**	Tile Size and Number	**Average**	Tile Size and Number	**Average**
250 MB file	640 tiles	**5257.28 ms**	320 tiles	**4560.91 ms**	192 tiles	**5537.82 ms**
500 MB file	1100 tiles	**10497.12 ms**	600 tiles	**10428.10 ms**	300 tiles	**8865.57 ms**
750 MB file	1452 tiles	**12685.31 ms**	726 tiles	**11955.36 ms**	363 tiles	**10561.41 ms**
1000 MB file	2250 tiles	**21613.61 ms**	1125 tiles	**17895.60 ms**	675 tiles	**17767.80 ms**
2000 MB file	4500 tiles	**39009.52 ms**	2250 tiles	**35894.68 ms**	1125 tiles	**37572.48 ms**

Test speed of reading for server cubelets from SSD						
File Sizes	0.5MB Cubelets		1MB Cubelets		2MB Cubelets	
	Tile Size and Number	**Average**	Tile Size and Number	**Average**	Tile Size and Number	**Average**
250 MB file	640 tiles	**951.29 ms**	320 tiles	**740.91 ms**	192 tiles	**1140.80 ms**
500 MB file	1100 tiles	**1589.10 ms**	600 tiles	**1370.17 ms**	300 tiles	**1918.19 ms**
750 MB file	1452 tiles	**1900.38 ms**	726 tiles	**1832.90 ms**	363 tiles	**1914.83 ms**
1000 MB file	2250 tiles	**3201.11 ms**	1125 tiles	**2600.07 ms**	675 tiles	**3917.68 ms**
2000 MB file	4500 tiles	**6329.05 ms**	2250 tiles	**5709.31 ms**	1125 tiles	**6543.67 ms**

Appendix 2. Disk reading speed results for server's cubelets.

Test speed of reconstructing for server cubelets						
File Sizes	0.5MB Cubelets		1MB Cubelets		2MB Cubelets	
	Tile Size and Number	**Average**	Tile Size and Number	**Average**	Tile Size and Number	**Average**
250 MB file	640 tiles	**1906.21 ms**	320 tiles	**867.74 ms**	192 tiles	**692.14 ms**
500 MB file	1100 tiles	**5284.28 ms**	600 tiles	**3029.89 ms**	300 tiles	**1813.04 ms**
750 MB file	1452 tiles	**10112.84 ms**	726 tiles	**5311.50 ms**	363 tiles	**3044.17 ms**
1000 MB file	2250 tiles	**22108.61 ms**	1125 tiles	**10624.84 ms**	675 tiles	**6346.53 ms**
2000 MB file	4500 tiles	**80155.11 ms**	2250 tiles	**40763.33 ms**	1125 tiles	**24016.02 ms**

Appendix 3. Test results for reconstruction of the cubelets.

Test speed of rendering server	
Render Size	**Average**
250 MB	**3135.44 ms**
500 MB	**6190.75 ms**
750 MB	**9267.31 ms**
1000 MB	**12344.16 ms**
2000 MB	**24211.03 ms**

Appendix 4. Rendering speed results on the server side.

Test speed encoding Image	
Render Size	**Average**
250 MB	**72.37 ms**
500 MB	**72.19 ms**
750 MB	**69.78 ms**
1000 MB	**70.91 ms**
2000 MB	**74.49 ms**

Appendix 5. Speed of encoding render to image.