3DAVis: Remote Visualisation of 3D Astronomical Data

Mivhaela van Zyl - VZYMIC015
VZYMIC015@myuct.ac.za
Department of Computer Science, University of Cape Town
Cape Town, South Africa

ABSTRACT

This paper focuses on the implementation of the 3DAVis system, the constructed system is a proof of concept for a hybrid approach to a server-client rendering system. It was made to be able to render and facilitate interaction with astronomical data of increasingly larger sizes in real time and to overcome the challenges of visualising very large data sets. The final 3DAVis system is made up out of a client application that runs in a web browser and a server system that will facilitate file storage and high-resolution image production. As the amount of data that the client is required to process and render increases the time the client requires to do so increases in proportion to the amount of data in a O(n) manner. Also, as the amount of data cubes that need to be combined by the client increases the time that is required to perform the combination in creases in an exponential manner with each additional data cube. The implementation is promising and can be further developed in the future, however, the amount of data that the client receives at any time must be limited to maintain performance. The clients system's performance is consistent across interactions and the primary factor that will affect the speed of user interaction is the time that the server take to complete its function and communicate the results to the client.

1 INTRODUCTION

Astronomical data presents certain challenges that affect the approaches we take when developing a system that can visualise this kind of data [5]. Firstly, there is not a single dominant format which means attempting to make a generic system would be difficult, a single format must be selected to base the system on. There is also the aspect that astronomical data has a low signal to noise ratio as well as a high dynamic range. It becomes difficult to separate the data from the noise and to normalise the data to within a range that can be visualised. Astronomical data uses dimensions in a different way to typical two and three dimensional spatial datasets, and maps different data types to axes and correctly mapping these axes is imperative to accurate visualisation. Possibly the greatest challenge and the challenge we will be mainly addressing is how to effectively handle the size of the data that is produced by astronomy recording instruments. High resolution data often contains millions of data points and could easily take up the entire storage space of several hundred home computers.

Systems attempting to addressed this problem have been proposed and developed for example SlicerAstro [13], iDaVIE-v [8], Frelled [14], Fips [6] and others that touch on real time visualisation [12] [4] [5] [10].

1.1 Motivation

There is a need within the astronomy community for a system that can reliably produce and accurately portray data while accommodating the complexities associated with processing and rendering astronomical data.

The main portion of data processing and visualisation would be done by a remote server system which would give a wider range of people access to the data regardless of the computational power of their local computers. It also reduces the need to transfer the data between client computers and would remove obstacles from the scientists path in being able to explore data and study it effectively.

The motivation behind implementing 3DAVis is to address certain challenges around visualising and interaction with astronomical data cubes in real time, these challenges derive from certain characteristics of astronomical data. The extremely large size of the astronomical data cubes which makes it incredibly difficult to render them in a timely manner even when making use of a dedicated powerful computer. This also limits how astronomers can work with the data, they are limited to using these dedicated computers or attempting to use their own computer which can not render the entire data cube.

1.2 Research Questions

The key questions for this research pertain to the design and implementation of visualisation software within the context of astronomy, these were:

(1) How can the computational overhead of rendering large amounts of data be offloaded from the client system?
(2) What is the limit for the size of data cubes that are rendered by the client system before impacting usability?
(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 the hybrid model approach scale in terms of rendering time, transfer latency, and time from user interaction and feedback over increasingly larger data sets?

1.3 Contributions

During the course of this research a proof of concept system was implemented in order to test whether it is a feasible model for future remote visualistion software to be based on. The 3Davis system is comprised out of a client component and a server component, this paper will focus on the client component which facilitates user interaction with the client application as well as the server. The client is a browser application aimed at presenting information to the user in either as a realtime rendering within the browser or as an image transmitted by the server.

1.4 Overview

The current technology present in the field of astronomical data visualisation and rendering will be discussed in section 2. and how they deal with the challenges that astronomical data presents. As well as exploring the various parts that make up such systems and packages that would potentially be used to implement these these parts.

Section 3 discusses the various parts that make up the system as well as the various algorithms will be discussed to showcase the internal structure. Particular focus will aimed at the front-end or client side sub-system, discussing its functions and features and depicting how it contributes to the overall system.

In section 4 the experiments used to test the final system are formulated, as well as how each experiment or test relates to one of the research questions. It will also be covering the results obtained from the experiments.

Within section 5 each set of results will be discussed and evaluated in terms of how they performed against their experiment criteria and how they assist in answering the research questions 1.2.

The conclusions will be derived from the results will be presented in this section 6.

2 RELATED WORK

The remote visualisation of three dimensional astronomy data has many parts that require careful consideration taking into account the nature and challenges presented by astronomical data especially the size of the data artefact.

The astronomy field conducts their research by observing the visible universe and collects data through telescopes and satellites. These data collection methods generate extremely large amounts amounts of data. Scientists wanting to study the data need to interact with and explore the collected data which is done through the visualisation of the different parameters contained within the data. Through the interaction and manipulation of the data scientists are able to make discoveries and gain insights into our universe.

In this section we will be discussing the various functional parts that make up the system for the remote rendering of three-dimensional astronomical data, which will be implemented as a server-client system. These functional parts include, data visualisation methods, data rendering, and user interaction with the data.

As well as investigating libraries and packages that would assist in implementing this proof of concept system.

2.1 Data Visualisation

Data needs to be visualised in order for it to be interacted with in an intuitive way. There are a multitude of ways in which to represent and render volume data. Similar techniques for data visualisation for astronomy are used in the visualisation of medical volume data because both fields have to contend with how to represent large amounts of data [3] [7]. The following subsections will elaborate on different visualisation techniques and their ability to render volume data.

Volume rendering represents three dimensional data as voxels which can be translated from data points in the dataset and projects them into the picture plane [3] and does not directly deal with surfaces [9]. The colour and opacity is computed for each voxel and classifies them in terms of what proportion they are of each object class. It does not require the explicit classification of binary surfaces to define the data as objects and non objects like in isosurfaces. Meaning is added to the data through colouring the rendered points based on a parameter of the data like temperature [1]. This rendering method produces a model which has a semi transparent gel appearance and is suited for displaying weak or fuzzy surfaces which is a much better way of representing astronomical data. This gives volume rendering the ability to render both the external surfaces and the interior three dimensional structures 1 which would have otherwise gone unnoticed [4] [11].

However, the rendering time grows linearly with the size of the dataset. We encounter problems when the dataset reaches a size where the computational overhead for rendering all the voxels in the scene becomes too great as all the voxels in the scene participate in the rendering.

2.1.1 Visualisation Libraries.

VTK and VTK.js. Standing for The Visualisation Toolkit, 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.

OpenGL and WebGL. A three dimensional graphics API based on OpenGL, made and maintained by Kronos and has an open source license under the copyright of The Khronos® Group Inc. It renders three dimensional graphics in a browser by making use of the HTML 5 canvas element.

2.2 Rendering

Volume rendering of voxel scenes is computationally expensive and makes it difficult to render large datasets in real time and have the data be interactive with user input [14] [13] [7]. Real time interaction with large datasets over the internet is a challenge because one not only has to account for processing time but also the latency that comes with transmitting data over the internet.

Remote rendering of data cubes as opposed to visualisation on local system has to be put into consideration due to the fact that the data easily exceeds the size of a typical local system's memory [12]. It would be desirable to do the computing of the data on a remote system that has sufficient power and capacity and would provide the wider astronomy community with a low cost visualisation service [2] as well as reducing the need to transfer data between systems. The user would request a visualisation of a dataset from the remote host and the remote host would send the visualisation to the clients computer [4].

Client Side rendering. Where the client take on the computational overhead to render the data. The server only sends the data and is not involved in the rendering and visualisation of the data. This approach limits the amount of data that the server can send the

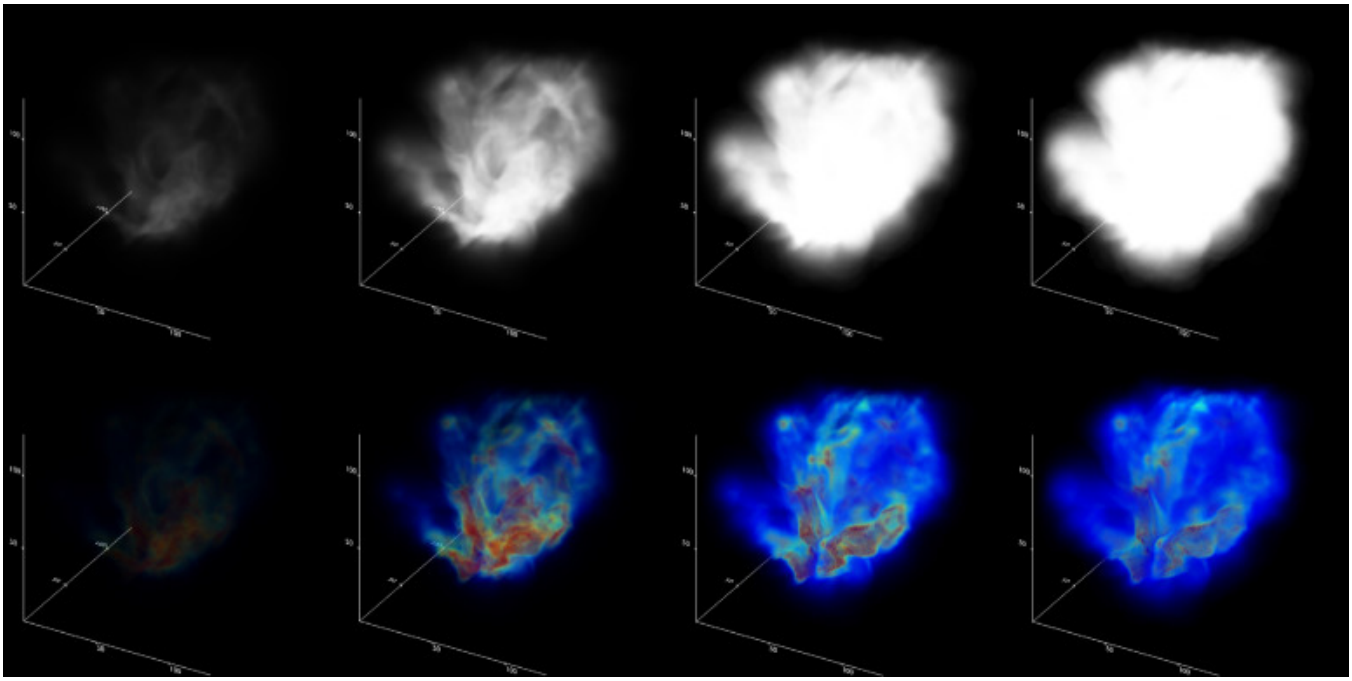

Figure 1: Astronomical data visualised using volume rendering shows how the classification and colour coding of the individual data points uncover emergent data structures that could otherwise not be seen [14]

client and the rendering is limited to the clients computational power. However, if interaction takes place and the client computer is not overwhelmed with data interaction should be smooth and above a framerate threshold.

Server Side rendering. Is where the server renders the visualisation from the data and then streams the frames to the client. The server has enough memory and computational power to store and render large amounts of data. However, with this approach if there is interaction with the data from the client's side the interaction has to be communicated to the server and a new rendering has to be produced and sent back to the client. This can produce undesirable latency between interaction and feedback which can be exacerbated if the bandwidth of the communication network is too little [13].

2.3 Data Interaction

The visualisation of astronomical data creates a need for interaction, interaction with the data is critical for knowledge discovery. It is necessary to allow the user to manipulate data to obtain an intuitive understanding of the data as well as quantitative results [11] [3].

Frelled [14] uses the software Blender that has all the desired characteristics for smooth and effective interaction because it is designed to manipulate and edit detailed three dimensional models and performs better than any current astronomical viewers. It would also be desirable to include the ability to change visualisation parameters and dynamic data filtering especially with radio astronomy data that has many parameters that could be visualised [4]. This would give the user the ability to intuitively and precisely navigate through three dimensional data.

An example of an astronomical data viewer is iDaVIE-v [8] which is used for data cube exploration, makes use of the game engine Unity to visualise the data cube and allows the user to interact with the data in an intuitive manner.

2.4 Discussion

In this section various methods pertaining to how to approach the problem of visualising and interacting with very large astronomical datasets within a web browser were discussed. The size of the datasets indicates that they require a dedicated computer to be able to store, process and render the visualisations. Taking this into account remote rendering becomes the best option to make the data accessible to a wider range of individuals.

Certain visualisation techniques are better suited to the nature of astronomical data, such as volume rendering which is able to reproduce the fuzzy cloudlike nature of astronomical structures and can visualise different parameters of a data points. Both VTK and VTK.js packages are developed by the company Kitware and they use the same coordinate systems. Which allows the client and server to exchange data without the need for any type of conversion between them, and helps to simplify the system. They will be used as the preferred rendering packages for the implementation of the final system.

How effectively the user can interact with the data is directly related to how quickly the model can be rendered and re-rendering while the user is manipulating it from the client side. If the model cannot be rendered in a timely enough fashion then the framerate the user sees drops to below a usable threshold. The model becomes unusable because the user must wait for the model to be

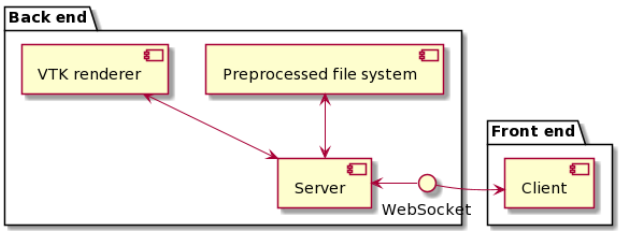

Figure 2: A high level overview of the system and its individual parts. The back end sub-system is comprised out of the VTK renderer, preprocessed file system and the server which connects to the client within the front end sub-system via a WebsSocket.

re-rendered after every adjustment. This would severely hamper the user's ability to extract knowledge. Therefore, rendering data can be partially done on the client computer during user interaction and partially done on the server for a more detailed view of the data.

3 DESIGN AND IMPLEMENTATION

The larger overall system of 3DAVis is comprised out three main components: client/frontend system, the backend system which consists of the server and the pre-processed file system. These separate systems work together implement a hybrid approach to rendering by performing rendering on the client as well as on the server. Where the server produces a high resolution visualisation in the form of a image of the full sized data cube and the client renders tiles from the file system. These tiles are pieces of the full sized data cube at a certain resolution level so that they can be rendered on the client computer without requiring excessive computational power. Interaction with the rendered tiles is performed on the client subsystem and because the rendering is done on the client interaction is instantaneous. The success of the system is determined by how well the individual parts function and interact with each other.

The server pre-processes full size astronomical data cubes from FITS to HDF5 for faster data retrieval while also fulfilling requests from the client. The requests are either for image or volume data, image data comes from the rendered visualisation of the full sized data cube, this is done on the server. Whereas volume data comes from the pre-processed files, the volume data within these files will be rendered by the client subsystem. While the client assists in rendering visualisations of the data its main purpose is facilitate all interaction with the system, allowing the user to view different astronomical data files as well as perform actions on the data. The overall effectiveness of the system relies on how well these components can work together and produce usable data visualisation for the user.

3.1 System Overview

By combining server and client rendering and shifting rendering to both subsystems the full sized data cubes must be pre-process into different levels of data resolution, these levels are further divided up into tiles which can be individually requested. These tile are used by the client to produce a rendering of the data without having to load the full sized data cube.

The system initiates when the client application is started, the server and the client subsystems create connection for sending and receiving data. After this connection is created the client requests the file names of all the available files on the server, these files are the full sized data cubes that have been pre-processed and are stored on the server. The file names are displayed for selection by the user. When the user requests to view a data file by selecting it from the list it informs the server and the server renders and sends an initial high-resolution visualisation of the data cube as well as the lowest resolution level of the down sampled data, this level visualises the the whole cube. When the image is received by the client it is displayed for the user and the volume data is stored for when it is needed. As the user interacts with the image the volume data is renderer and the visualisation is shown to the user for the duration of the interaction. Some examples of the types of the user interaction is manipulation the view of the data as well as the cropping planes.

When the interaction ends and a certain amount of time has passes the camera position and the colour transfer function of the client renderer are sent to the server so that the server can render a high resolution recreation of the rendering on the client system. When the high resolution rendering is then sent to client system the client rendering is switched back to the high resolution image. If the data cube is cropped the resolution of the cropped section needs to be increased by certain factors along the x, y axis and z axis independently. More tile are required by the client and the specific resolution level file and the specific tile or tiles within the file are requested. If multiple tiles are requested by the client each tile is sent individually and are combined into a single volume once all of the requested tiles have been received by the client. The connection between the server and the client is terminated when the user terminates the client system instance.

3.2 Client

The primary purpose of the client/frontend system is to visualise the data received form the server in the forms of image and volume data and to handle and process user interaction with the data. It was constructed using Vue.js, a frontend JavaScript framework for the construction of reactive client-side web applications.

The client Subsystem was constructed using the Vue.js javascript application framework and allows for easy creation of modular components. Additional packages for functionality can easily be added with a package manager like NPM, some of the packages that were used are vtk.js which is used for client-side data visualisation and is the JavaScript extension of VTK library that is implemented on the server-side system. The Bootstrap CSS package was used to improve the overall aesthetic and user experience compared to default HTML elements. To facilitate the connection between the client and the server JavaScript WebSocket are used to send and receive data without having to rely on singular HTTP requests and responses, allowing the client to make a single request and receive multiple responses.

The client will primarily receive images of fixed size in the implementation this size is 1080 pixels and 720 pixels and or one to

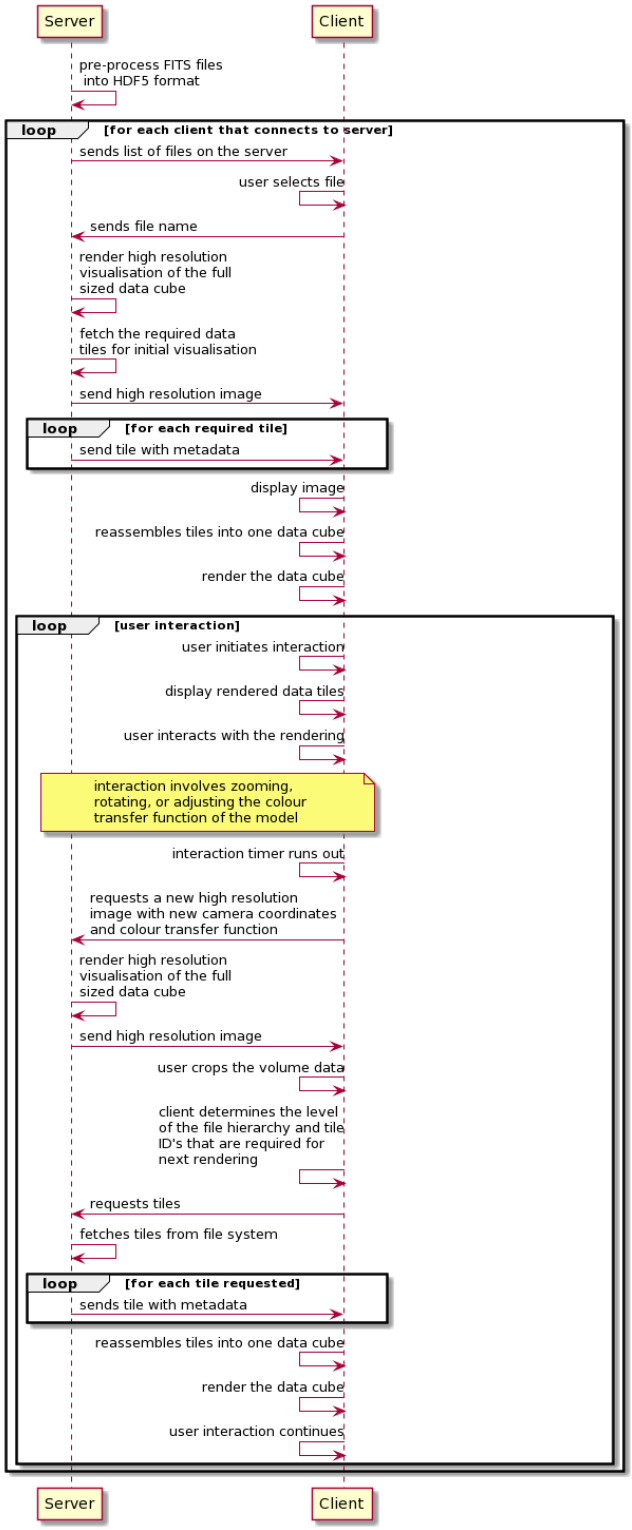

Figure 3: The hybrid server-client rendering approach the system implementation is based on and shows how the server and the client interact with each other while a user is engaged and performing actions.

two data cubes of a fixed which is 64 by 64 by 64 data points per axis. These dimensions limit the amount of data that the user will receive, the dimensions can be increased of decreased and limit the amount of data the client computer will have to process during its use.

3.3 Algorithms

3.3.1 Tiling. This is used in order to determine which tiles need to be requested from the server. Once the volume data has been cropped the client system determines the highest resolution level where the cropped selection would fit with one or two data cubes. It then determines the tile ID's for the level and then requests them. This function is called when the data cube is cropped and uses the x, y, and z planes as input.

```
cropCube() {
    cubeSize = 64

    determine the factor of each dimension
    factor[x,y,z] = serverCubeDimensions[x,y,z]/clientCubeDimensions[x,y,z]
    the factor is what the planes have to be multiplied by to get their
    position in the full sized cube's space

    convert the crop planes coordinates to coordinates in the full sized
    data cube by multiplying the points by their respective factors

    store the lowest resolution level of the file sent by the server
    xy = xyMaxLevel
    z = zMaxLevel

    do {

        if(z == 1) {
            divide the XY level by 2
            xy /= 2
            reset Z level to max Z level
            z = zMaxLevel
        }
        else {
            divide Z level by 20
            z /= 2
        }

        set the level of cube division
        xyDivLevel = xyMax/xy
        zDivLevel = zMax/z

        get the number of cubes that the crop cube covers on each dimension
        xNumTiles = Math.ceil(serverCubeDimensions/xyDivLevel)/cubeSize
        yNumTiles = Math.ceil(serverCubeDimensions/xyDivLevel)/cubeSize
        zNumTiles = Math.ceil(serverCubeDimensions/zDivLevel)/cubeSize

        determine in which block the point dimension falls into for each
        crop plane for x and y planes
            Math.trunc(cropPlane/xyDivLevel)/cubeSize
        for the z planes
            Math.trunc(cropPlane/zDivLevel)/cubeSize

        convert the planes to vertices of the cube
        form [x1,x2,y1,y2,z1,z2] to eight x, y, z coordinates

        for each vertex on the cubes as vertex {
            let tileID = vertex[x] + 1

            if(vertex[y] > 0) {
                tileID += (xNumTiles * vertex[y])
            }

            if(vertex[z] > 0) {
                tileID += (vertex[z] * (xNumTiles * yNumTiles))
            }
            Add tile id to the list of ID's
        }
    }
    while the list of tiles is bigger than 2 or the highest resolution
    level has been reached

    determine if the two tile are adjacent to each other on the x, y,
    or z axis

    send the xyDivLevel, zDivLevel and the tile ID's
```

}

3.3.2 Tile Combination. This algorithm is used to combine the tiles from the server on the client system before rendering interactive volume model. Each data cube that the client requests is send in its sown independent message from the server so as to not burden the client with one large piece od data to download at one time. Upon receiving the data cubes the client system must combine them into a single data cube so that it may be rendered. It is the same algorithm used on the server for the same purpose and is implemented with minor changes to function on the client.

3.4 Data Transfer

The client will make various requests over an active session, the requests the client will make are for volume and image data. The server responses to the data from the client by either producing a rendering according to the specification or sending data tile from a specific resolution level. The server needs to specify what type of data is being sent with the message in order for the client system to distinguish between the responses. This is the data the client system needs to give back to the user as feedback.

3.4.1 File list. The list of file names are sent from the server directly after the connection between the server and the client is made. The file names in the list correlate to different full sized astronomical data cubes that have been preprocessed and stored on the server. The client does not need to make an explicit request for the file names.

3.4.2 Volume data. During the user interaction with the volume data the user has the option to crop the cube in order to take a closer look at a specific section. When the cube is cropped the vertices of the cropping box is recorder as well as which resolution level the the new tiles need to be from and the ID's of the tiles which cover the crop selection box. The algorithm used to determine these values is explained in more depth in the algorithms section. The crop box vertices are converted into the world coordinates of the full sized cube before being sent to the server. These coordinates, the file name that contains the tiles of the desired resolution and the tile ID's are sent to the server when the data is cropped.

The server sends the requested tiles as chunks to the client where each chunk is a data cube of a specified size.

3.4.3 Image Data. The system keeps track when the parameters such as the camera position of the colour transfer function of the client visualisation changes. This is used to orient the camera and colour the model in the server rendering. The data is sent to server when the user has discontinued their interaction and a new high resolution visualisation is needed. The server uses the parameters to recreate what the user sees while they are interacting.

Data representing an image is transferred to the client in a compressed format, the client decompresses the image data.

3.5 User interface

The user Interface of the client system allows the user it access the functionality and interact with the data located within the server. It consists out of elements that allow the user to access the data on the server in the form of files and data cubes, as well as the high resolution visualisations produced by the server. Mechanisms to manipulate the rendering produced by the client such as changing the colour function so that the volume rendering displays differently or to manipulate it position relative to the camera. The user interface and how it appears at various stage of interaction is depicted in figures 4, 5, 6, and 7.

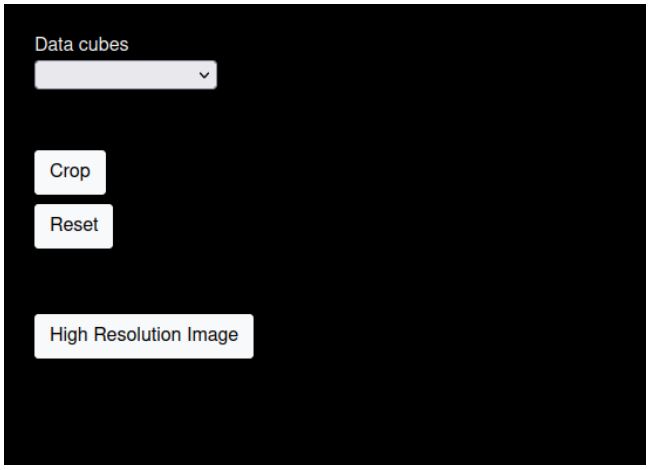

Figure 4: The user interface of the client system before the user has selected a file with a drop down selection element to allow the user to view and select a file as well as the the other function the user can perform on the data cube, such as crop the data cube, reset the page and request a high resolution image if desired. The rest of the screen is blank.

4 EXPERIMENTS AND RESULTS

In this section the experiential methods for testing the system will specified as well as the variables each experiment will be testing. These experiments and results pertain to the performance of the client system. All experiments were conducted on a computer using the Ubuntu 20.04 operating system with 16GB RAM, a AMD Ryzen 5 3600 6-Core Processor and a Palit Nvidia Gefore RTX 3070 graphics card.

4.1 Rendering

This experiment is to test the speed of rendering volume data on the client computer where the limits of the browser rendering can be tested and an optimal size for the data cubes can be determined. The time in which the client computer requires to render a data cubes will be recorded and the data cubes are tested in increasing size. A data cube is a three-dimensional objects with an x, y, and z axis and is comprised out of individual data points. This is to determine the relationship between the size of of the data cube and the time required to render the data by the client computer. The results are shown in table 1 and teh visualisation can be found in graph 8.

4.2 Data cube combination

These set of experiments tests the speed of the tiling algorithm in terms of how it performs when combining data cubes of increasing

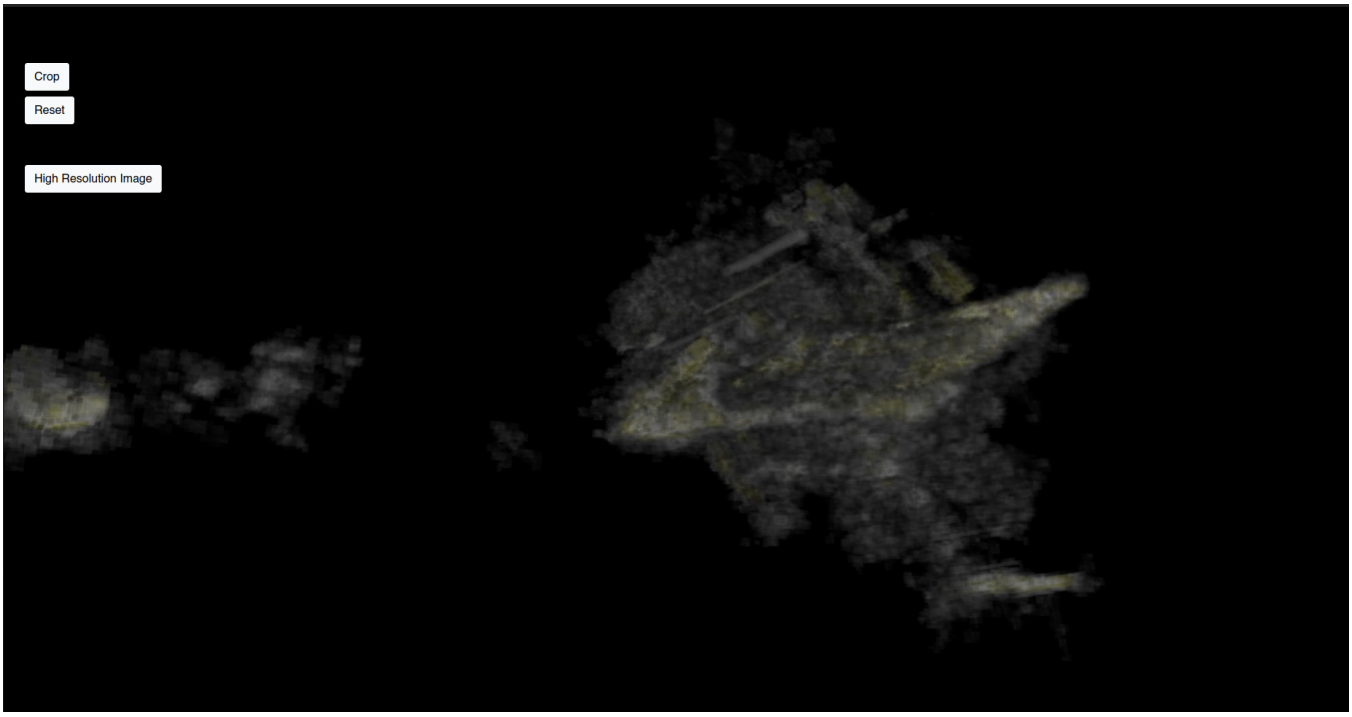

Figure 5: The high resolution visualisation produced by the server of the full sized data cube also located on the server. The image is displayed when the user stops interacting with the window.

Number of data points	Cube dimensions	Average Time (ms)	Average time (s)
262144	64x64x64	468	0.468
2097152	128x128x128	1672	1.672
16777216	256x256x256	11698	11.6976
134217728	512x512x512	89762	89.7618

Table 1: The results from the test to determine the relationship between the size of a data cube and the time that is required to render it by the client. A visualised depiction of the data can be found by graph 8.

sizes as well as how it perform when combining an increasing number of data cubes.

This experiment recorded the amount of time it required to combine two data cubes of increasing sizes. Its purpose is to determine the relationship between the number of cubes being combined and the time it required by the cube combination algorithm adn therefore determining the algorithms performance. All data cubes are of a uniform size of 64 on each axis. The results for the test can be found in table 2 and a visualisation of the data can be found in graph 9.

This experiment recorded the amount of time it took to combine an increasing amount of data cubes. It records the amount of time it takes for the cub combination algorithm to combine cubes of increasing sizes and determine the effect larger data cubes have on the time of execution of the algorithm. The results for the test can be found in table 3 and a visualisation of the data can be found in graph 10.

5 DISCUSSION

The size of the data cube that the client computer can process and render is dependent on the processing power of the client computer as the number of data points increase the rendering time for the data increases in a linear manner. For the client to be able to handle larger local data cube sizes the processing power of the computer must be increased.

The rendering time for data cubes on the client system increases linearly with the amount of data points, where as the data cube dimensions increase so does the time required to render it. This is due to no being able to discard any region of the data while rendering, each data cube must be processed and rendered and therefore the execution time for the rendering of the data will have an O(n) relationship with the rendering time. The largest size the data cube can handle is determined by the individual client computer's specifications however limiting the current dimensions of the data cubes to 64 by 64 by 64 ensures the client computer will be able to render data cubes in a timely manner by minimising computational

Number of data points	Number of cubes	Dimensions of cubes	Average Time (ms)	Average time (s)
65536	2	32x32x32	780.7	0.7807
524288	2	64x64x64	5175.7	5.1757
4194304	2	128x128x128	40813.4	40.8134

Table 2: The results from the test performed to determine the relationship between the size of the cubes being combined in the combination algorithm and the time it take for the algorithm to combine them. A visualised depiction of the data can be found by graph 9.

Number of data points	Number of cubes	Average Time (ms)	Average time (s)
52288	2	5175.7	5.1757
786432	3	9733.9	9.7339
1048576	4	23895.2	23.8952

Table 3: The results from the test performed to determine the relationship between the number of cubes being combined and the time it take for the algorithm to combine them. A visualised depiction of the data can be found by graph 10.

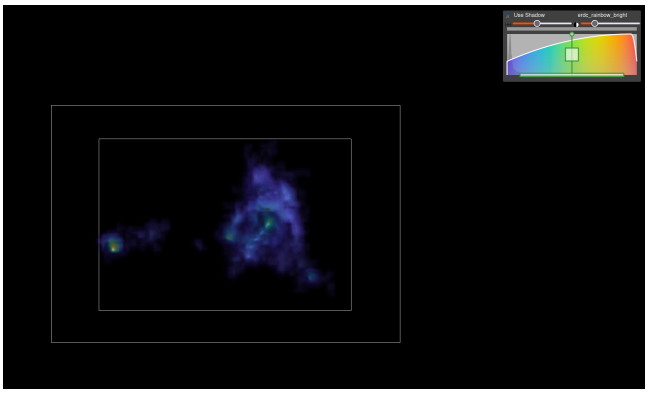

Figure 6: The volume rendering produced by the browser on the client computer. The rendering is interactive and the user can manipulate the volume data by zooming in and out, rotating around x, y, z axes and manipulating the cropping area by clicking and dragging the vertices and planes. How the data is visualised can be manipulate using the widget at the top right corner of the screen.

overhead for the client system. A data cube with the dimensions of 64 render on average under half a second while maintaining a high enough resolution to depict significant structures within the volume data. Whereas data cubes with dimensions of 128 will have a higher resolution of the data and depict the structures with more accuracy the number of data points increased exponentially causing the rendering on a single 128 by 128 by 128 to be 1.6 seconds on average, three times the duration of the 64 by 64 by 64 cube. This time could be decreased with more computational resources but it cannot be assumed the the user would have access to such resources. A 32 by 32 by 32 data cube would take less time to render however it will not have high enough resolution to represent the structures within the data.

A data cube with 64 by 64 by 64 dimensions requires on average half a second to render, however, the time for combining the cubes if there is more than one must be considered and added to the rendering in order to determine how long the user must wait before interaction can begin.

When combining data cubes of increasing sizes the time required to combine data cubes increases linearly with increased number od data points within the cube. It has the same effect of the time as rendering the larger data cubes where the time increases in a linear relationship with the number of data points.

When combining multiple data cubes of the same size the time each additional cube adds plots the beginning of an exponential relationship. This is an undesirable result considering the goal of minimising the waiting time for the user. Therefore, within the implementation the total number of cubes the algorithm will have to combine on the client is two. Combing two cubes of 64 by 64 by 64 dimensions requires on average 5 seconds and adding the rendering time it would produce a visualisation of the data within 6 seconds upon receiving the final data cube. This waiting time will only effect the user when initially loading the data and when cropping the cube where cubes are transfer from the server to the client.

It must be noted that the server and client were executing on the same local computer where time for data transfer was negligible. The time for data transfer would be affected by the bandwidth of the internet connection which can vary depending on the distance from the server and the network speed. This time for data to be transfer would add additional waiting time for the user before the system produces a visualisation. The system could not be judged in terms of how it would perform with different network bandwidths and latencies due to the client and server located on the same computer during testing.

6 CONCLUSIONS

To conclude this paper concerning the remote visualisation of radio astronomical data using the server client model.

The goal of the research was to implement a proof of concept system that combines server and client based rendering in order to overcome the challenges present in rendering astronomy data mainly the overall size of the data.

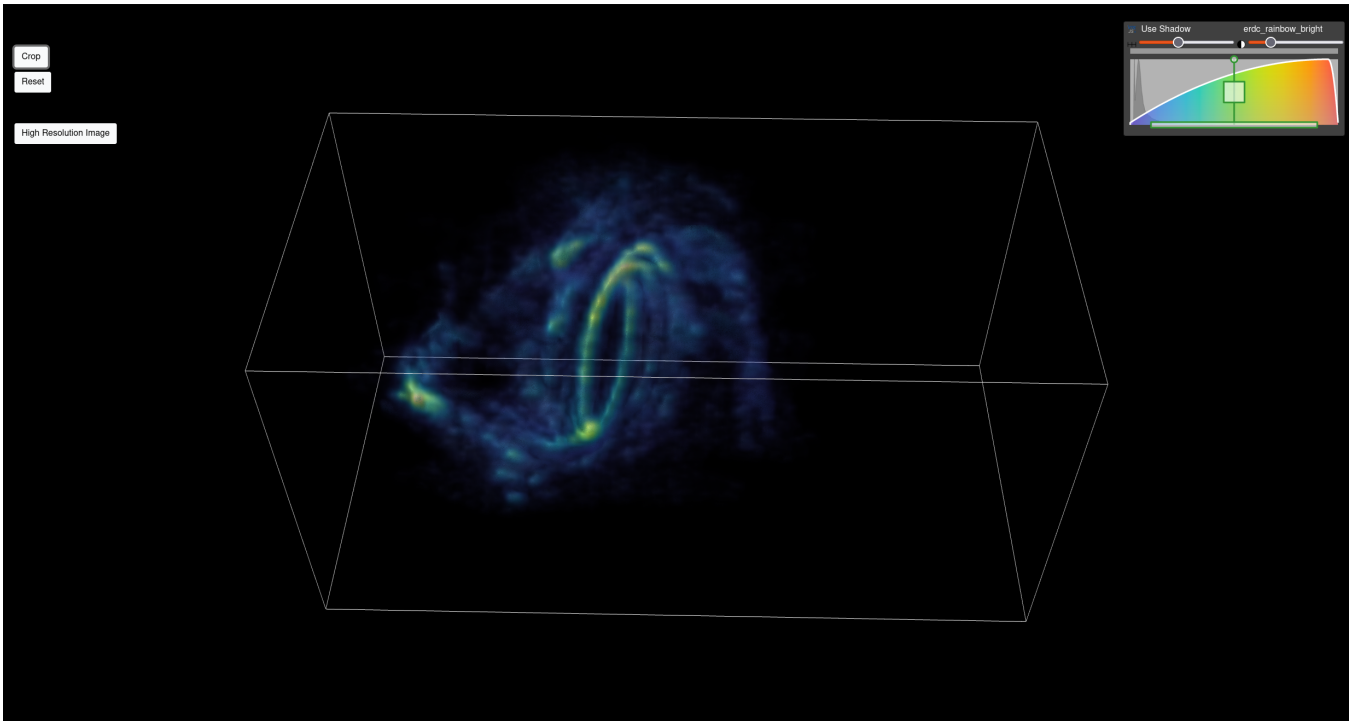

Figure 7: After the cropping area has bee sufficiently manipulated and the crop button has been selected, the server sends data cubes of the selected area at a higher resolution level. This is a view of the data from 6 but at a higher resolution level than before the crop and more detail of the structure can be seen.

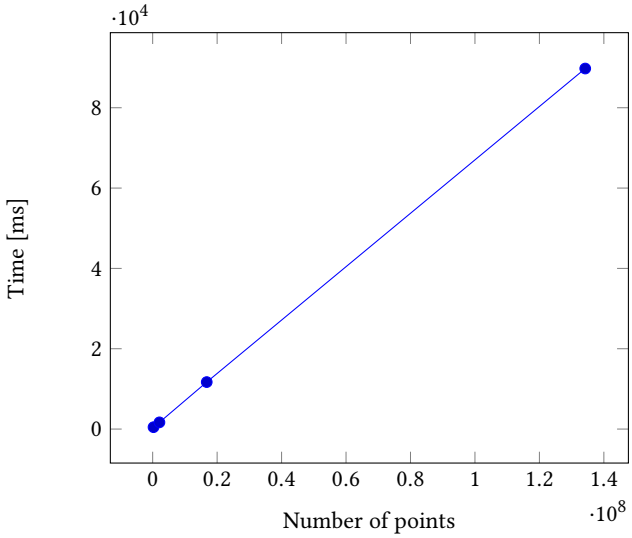

Figure 8: The relationship between the size of the cube being rendered on the client and the time taken by VTK.js to render it. Tabled results can be found in table 1.

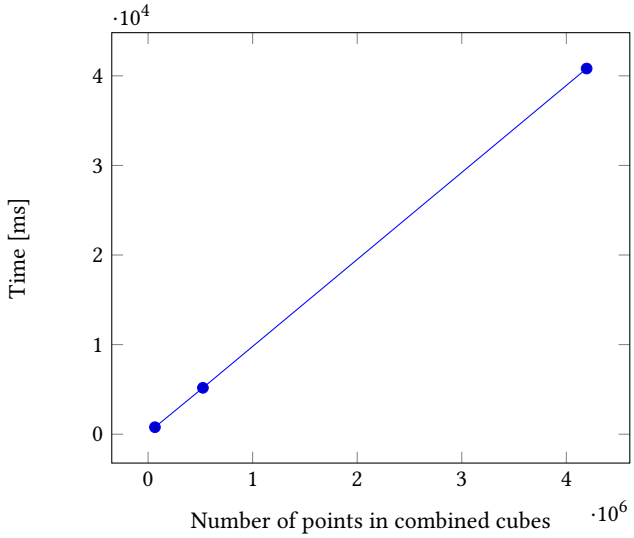

Figure 9: The relationship between the size of the cube being combined on the client and the time required for them to be combined. Tabled results can be found in table 1.

The research questions pertained to the derivation of possible methods to keep the rendering done by the client system to a minimum, what the limitations of client side rendering are and at what point would it impact the usability of the system. As well as how the client system would react to receiving larger data cubes.

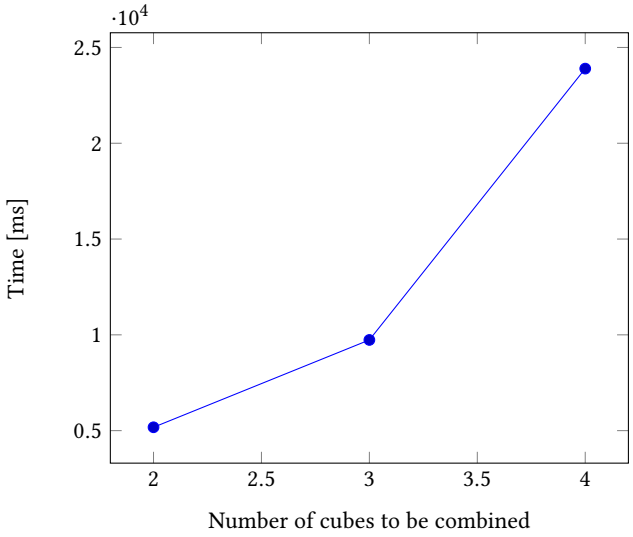

Figure 10: The relationship between of the number of cubes being combined on the client and the time required to combine them. Tabled results can be found in table 3.

The computational overhead for the client was minimised by limiting the amount of data it would receive and process. This was considered from the initial planning of the system (Figure 3). The client would either receive an image of a fixed size or a limited number of data cubes of a fixed size. In the implementation the client is limited to combine and render at most two data cubes of maximum dimension of 64 x 64 x 64, and display and image with dimensions of 1080 by 720 pixels. Data cube dimensions of 64 were found to provide sufficient resolution to depict the data with some detail while rendering at a speed that maintained usability.

It is also important to limit the number of data cubes sent to the client system due to the time required by the cube combination algorithm increases in an exponential manner with each additional data cube. If the client were to combine multiple data cubes it would increase the waiting time for the user before a visualisation of the data can be shown.

In terms of scalability performance of the client system would remain constant across all interactions regardless of the of the size of the data cube being processed by the server. The waiting time of the user is affected by change in size of the data cube and the amount of data cubes sent by the server. This does not take into account the the time the server requires to perform its functions or the amount of time it would require to transfer the data over a network. In this aspect the client system's performance is dependent on the server's performance and the network bandwidth. If the time for server functions and data transfer increases the client system would have to wait a longer period of time before receiving its required data but it would not effect the speed of interaction with the data cube once it is rendered by the client.

As a proof of concept the system shows promise and with a sufficiently powerful dedicated server it is more usable in terms of user interaction than exclusive server side rendering by allowing the user to interact on the client computer, and does not burden the client system with rendering full sized data cubes. The implementation combines the best aspects of both rendering approaches, combining the computational power of a dedicated server withe the speed of user interaction of rendering on a local computer.

REFERENCES

[1] Bennett W. Anderson and Robert P. Burton. 1988. Computer graphics curricula: a survey of PhD granting departments. *ACM SIGGRAPH Computer Graphics* 22, 2 (apr 1988), 94–98. https://doi.org/10.1145/47824.47825

[2] David G. Barnes and Christopher J. Fluke. 2008. Incorporating interactive three-dimensional graphics in astronomy research papers. *New Astronomy* 13, 8 (nov 2008), 599–605. https://doi.org/10.1016/j.newast.2008.03.008 arXiv:0709.2734

[3] H. Fuchs, M. Levoy, and S.M. Pizer. 1989. Interactive visualization of 3D medical data. *Computer* 22, 8 (aug 1989), 46–51. https://doi.org/10.1109/2.35199

[4] 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. https://doi.org/10.1071/AS10031 arXiv:1102.5123

[5] Thomas H. Jarrett, A. Comrie, L. Marchetti, A. Sivitilli, S. Macfarlane, F. Vitello, U. Becciani, A. R. Taylor, J. M. van der Hulst, P. Serra, Neal Katz, and M. E. Cluver. 2020. Exploring and interrogating astrophysical data in virtual reality. *arXiv* (2020). arXiv:2012.10342

[6] Matwey Kornilov and Konstantin Malanchev. 2020. Fips: An OpenGL based FITS viewer. In *Journal of Physics: Conference Series*, Vol. 1525. Institute of Physics Publishing. https://doi.org/10.1088/1742-6596/1525/1/012047

[7] Koojoo Kwon, Eun-Seok Lee, and Byeong-Seok Shin. 2013. GPU-accelerated 3D mipmap for real-time visualization of ultrasound volume data. *Computers in Biology and Medicine* 43, 10 (oct 2013), 1382–1389. https://doi.org/10.1016/j.compbiomed.2013.07.014

[8] Lucia Marchetti, Thomas H. Jarrett, Angus Comrie, Alexander K. Sivitilli, Fabio Vitello, Ugo Becciani, and A. R. Taylor. 2020. iDaVIE-v: Immersive data visualisation interactive explorer for volumetric rendering. *arXiv* (2020), 1–4. arXiv:2012.11553

[9] M Meißner, Hanspeter Pfister, R Westermann, and Craig Wittenbrink. 2000. Volume visualization and volume rendering techniques. (01 2000).

[10] Finian Mwalongo, Michael Krone, Guido Reina, and Thomas Ertl. 2018. Web-based volume rendering using progressive importance-based data transfer. *Vision, Modeling and Visualization, VMV 2018* (2018). https://doi.org/10.2312/vmv.20181264

[11] Ray P. Norris. 1994. The Challenge of Astronomical Visualisation. In *Astronomical Data Analysis Software and Systems III (Astronomical Society of the Pacific Conference Series, Vol. 61)*, D. R. Crabtree, R. J. Hanisch, and J. Barnes (Eds.). 51.

[12] Simon Perkins, Jacques Questiaux, Stephen Finniss, Robin Tyler, Sarah Blyth, and Michelle M. Kuttel. 2014. Scalable desktop visualisation of very large radio astronomy data cubes. *New Astronomy* 30 (jul 2014), 1–7. https://doi.org/10.1016/j.newast.2013.12.007

[13] D. Punzo, J. M. van der Hulst, J. B.T.M. Roerdink, J. C. Fillion-Robin, and L. Yu. 2017. SlicerAstro: A 3-D interactive visual analytics tool for HI data. *Astronomy and Computing* 19 (2017), 45–59. https://doi.org/10.1016/j.ascom.2017.03.004 arXiv:1703.06651

[14] R. Taylor. 2015. Frelled: A realtime volumetric data viewer for astronomers. *Astronomy and Computing* 13 (2015), 67–79. https://doi.org/10.1016/j.ascom.2015.10.002 arXiv:1510.03589