3D Astronomy Visualization

Project Proposal

MICHAELA VAN ZYL VZYMIC015, BILAL HASAN ASLAN ASLBIL001, and BHAVISH MOHEE MHXBHA001, Department of Computer Science, University of Cape Town, South Africa

1 PROJECT DESCRIPTION

Astronomy is a visual science that relies heavily on visualisation and interpretation of data, as claimed by Rolowsky et al [11]. Modern telescopes like MeerKat, the Australian Square Kilometre Array Pathfinder (ASKAP), the Large Synoptic Survey Telescope (LSST), and the Square Kilometre Array (SKA) can all collect around gigabytes to terabytes worth of information. The actual dilemma faced by astronomers during the past few decades is constructing the most efficient and interactive model of visualising this data [3, 9]. As the data sets become larger it becomes exponentially more difficult to visualise the data and becomes less interactive as the data has to be rendered in real-time. Through the research paper by Borkin et al. [2], the advantages of utilizing 3D visualisation of astronomical data was outlined and how this is nowadays accessible through the advent of technology, especially because of the evolution through multi-cores computers and highly performance Graphics Processing Units (GPU). These technological tools will thereby allow graphing data by rendering slices through data cubes along the different multidimensional axes.

Consequently, through this project, we aim to achieve the development of an interactive 3D visualisation model using CARTA in order to effectively visualise large data cubes and interact with them in real-time.

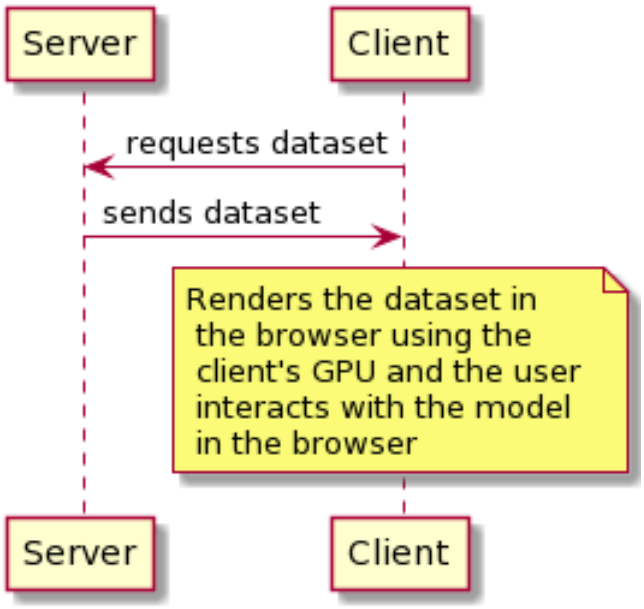

Fig. 1. The client based approach for the remote rendering of a data set.

2 PROBLEM STATEMENT

Although there are some 3D visualization tools that can be used for the visualisation of astronomy data, these tools often struggle to facilitate interaction with data sets whose size can easily exceed multiple client machines' memory capacities. Due to the size of the data sets the time period for processing and rendering the data in real time often exceed an acceptable threshold for user interaction. The majority of these tools use the client approach where whole data sets are loaded into client memory, using only the client's CPU's and available GPU's for rendering and processing data which severely limits the size of the data set that can be visualised. This approach requires the larger data sets to be broken up into smaller ones to be processed. An alternative to the client based approach (Figure 1) there is the server based approach (Figure 2) where the data set is processed and rendered on the server and the frames from the rendering are streamed to the client over a network. While this approaches takes processing pressure off of the client's machine the main disadvantages that stem from this approach are that as the data set become larger the longer the server takes to render the data and the longer it takes for the user to receive feedback after interacting with the data. This latency is compounded when considering the additional time it take for the data to be transferred over a network. The goal of this project is to determine which available libraries can be used to implement a hybrid of the client and server based

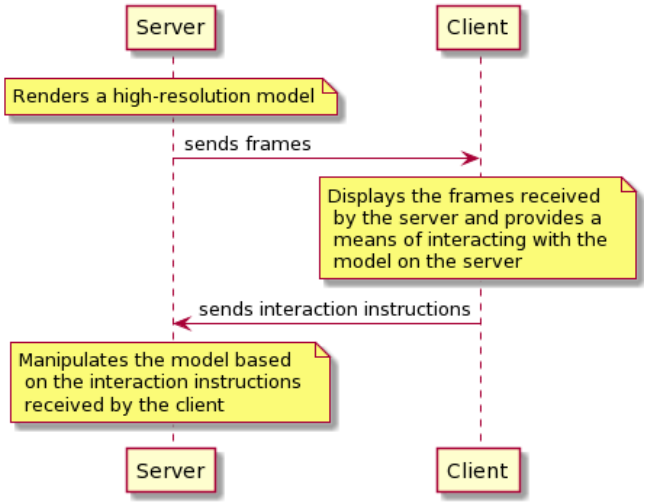

Fig. 2. The server based approach for the remote rendering of a data set.

approaches as well as the feasibility of the server-client approach for 3D visualisation of astronomical data sets. In order to achieve these aims, the following research questions are proposed:

(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) Which compression algorithms such as ZFP and SZ compress the data the best in terms of speed and size for transfer over a network form a server to a client?
(3) Which graphical rendering libraries such as VTK and XTK libraries produce the best results in terms of speed of rendering?
(4) How can the client and server approaches be combined to facilitate high-resolution data rendering over a network while maintaining usability?
(5) How does network latency when rendering large volumes of data using the hybrid model compare to the server-client model such as implemented in the CARTA framework?
(6) Hoe 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?

3 PROCEDURES AND METHODS

3.1 Implementation Strategy

The combination of the client and server based approaches can be combined in a hybrid approach where characteristics of each approach is combined in a manner that mitigate the disadvantages from each. At the beginning of the interaction the server streams a compressed image of a high resolution model constructed on the server as well as compressed downsized pre-processed volume data to the client. When the client initiated an interaction with the model the high resolution image is switched with a volume model that is rendered in the browser using the volume data sent with the image. The placeholder model remains in the window until the interaction stops, the transformations of the model are sent to the server where the full resolution model is rendered and the high resolution is sent to the client, this process is repeated for every interaction. As soon as the the interaction ceases the server streams high resolution image data once again.

3.2 Methods

Based on the research questions mentioned in the previous section, an adequate amount of planning is required to ensure the success of the project. Through periodic virtual meetings, we were able to analyse and discuss the most appropriate methods and procedures that we will adopt throughout the development of this project. Our goal is to develop interactive 3D astronomy visualisation using the CARTA client-server architecture for processing and rendering the data cubes. Due to numerous limitations of a fully server-side rendering system and that of a fully client-side rendering model, we decided to adopt a more hybrid system approach in order to nullify the drawbacks of having a completely one-sided structure. The server side uses images on a Portable Operating System Interface (POSIX) compatible file system in FITS [12] and HDF5 [5] (IDIA schema) format for faster access to the data as the client does not need to upload the file for each visualisation procedure. The entire process begins when the client requests for a particular image on the browser interface. The server is made aware of this request via the

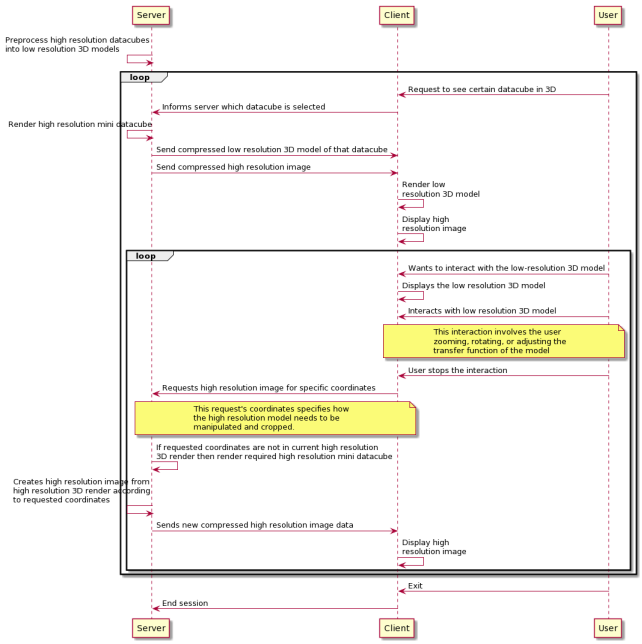

Fig. 3. Hybrid rendering on a client-server model.

communication layer API (Application Programming Interface) [1]. The server then fetches the image for pre-processing and compression. From an overall perspective, the server mainly performs two tasks. Firstly, it manipulates the data file in such a way that the high-performance server system does the heavy lifting work on the subset of the large data cube, renders it into a high-resolution image and sends the latter over for display on the client side. Consequently, this method does not require the server to send over gigabytes worth of data to the client for rendering. Now that the user is able to visualise the data, interaction using intuitive moves such as wheel scroll for zooming and click and dragging for rotation, will be enabled for the user to take advantage of . At that point, the server sends over a compressed version of the data cube to the client which then renders the data using the GPU on the machine. As this procedure is performed locally, interaction is expected to be smooth as it does not require transferring chunks of data during the interaction process for the server to render, as it would have been in a fully server-side rendering system. This is an overview of the entire visualisation process, but a more descriptive explanation of our method and procedure is provided in the following sections below.

3.2.1 Back-end. The back end is the server side of the CARTA system where all the data processing occurs for visualisation. Since CARTA was originally constructed in C++ programming language, the latter will be used for further development to ensure high performance and control over memory. C++ also supports many libraries like the Visualisation toolkit (VTK) [7] and the Compute Unified Device Architecture (CUDA) [8] which are crucial in this project as they are very often associated with astronomy and visualisation. After the user selects a data cube for visualisation on the client

side, the server is notified and starts processing the data by down-sampling using lossy compression libraries like ZFP and SZ. This compressed data is sent to the client for rendering and so, the user will see a low-resolution version of the data cubes at first until the high-quality image can be completely rendered by the server and transferred through the network to the client side. Then, the client will be allowed to interact with the model displayed on the client's screen using intuitive mouse scroll and click and drags or the widgets found on CARTA's browser interface at which point the local low -resolution version of the data produced locally, takes over. The user can also select the Region of Interest (ROI) which is a smaller subset of the data which is targeted for observation. Whenever, the user attempts to interact with the data cube and exceeds the current screen parameters , the high-resolution model is replaced with a low-resolution level of detail (LOD) model which is rendered by the client's GPU on the front-end browser. This method was chosen as it enables the user to see the changes caused by the interaction almost as soon as it is performed since data transfer through a network to the server is not required for the rendering process for this step. Shortly after the user stops interacting with the system for about 200 milliseconds, an update notification is sent to the server to acknowledge this step and proceeds to generating and streaming across the high resolution image to the front-end browser. The resolution sent over from the server will be down-sampled using mipmapping [6] and sent to the client's screen in order to have an effective frame per second (FPS) as using the maximum resolution of the picture would be considered as a waste of data as well as time spent for transferring over the network as this resolution cannot be displayed to its full potential due to the fixed level of detail of the client's screen. On the other hand, the minimalised compressed version of data cube, which is displayed upon user interaction, has to be comprehensible to the user in order to get the requested results and also has to project an interactive performance whereby the display is updated almost as soon as the interaction is processed. Therefore, experiments need to be performed to establish a balance between the two sides and obtain reasonable latency hiding and the understandable low-resolution level of detail.

3.2.2 Communication Layer API. In order to allow efficient communication between the server and the client, a communication layer comprising of an API has to be set up to send and receive data between the two parties. Therefore, a compatible API framework needs to be implemented that results in smooth data transfer between the server and the client. Some popular examples of web application API framework that can be applied in this project are REST and gRPC. REST is a relatively easy-to-use framework that is usually used in building web applications which is compatible with the C++ environment of the CARTA back-end. On the other hand, gRPC is also language agnostic and can therefore merge with the back-end to communicate with the front end. However, according to Chamas et al. [4], gRPC is approximately 7 times faster than REST for receiving data and approximately 10 times faster than REST for sending data. Ideally, gRPC seems to be amply sufficient to manage he back-end and front-end communication, however throughout the development of this project, other technological tools may be employed to achieve the functionalities required, especially when carrying data for streaming the data cubes and to handle user interaction seamlessly.

3.2.3 Front-end. The front end is the client side where the user can see and interact with the data generated. Several front-end libraries such as React, Vue and Angular exist which fits our purpose. However, upon discussion and analysis, Vue was chosen to be used for building the front-end framework as it is commonly used for creating interactive user interface and naturally combines user interaction and behaviour functionality into its components. Concerning the rendering for the visualisation of the compressed data cubes, tool-kits such as VTK and XTK can be deployed. VTK is better supported by the whole system for rendering astronomy data cubes whereas XTK is more lightweight and is able to use VTK file as input for the rendering process. During the implementation phase, the most efficient rendering method will be implemented so as not to hinder the interactive flow of the system. Similar to CARTA's original design, our system will also contain several features and widgets to interact with the data. These includes using the scroll wheel for zooming in and out and click and dragging for rotating around the data cube. As mentioned in the serve side section, upon user interaction, the low-resolution model will take over for latency hiding and will allow change to high resolution when the user stops interacting for a short period of time. After selecting a particular file as input, the user will be able to select a region of interest (ROI) to examine that smaller subset in more detail. The server will be notified of this update and will focus on processing the data for that subset and render more rapidly and efficiently due to a smaller model under inspection. Some experiments and testing will be performed at this point to find out of the most human intuitive manner to select the intended volume of data. As in CARTA, our model will also provide the same colour mapping to vary the transfer function for astronomers to utilize for searching for specific matter. Likewise, histograms and wavelength graph will also be presented to the user. The user will be able to interact with the diagrams as well to results at fixed coordinates.

3.3 System Evaluation

There are three main challenges that we expect to encounter during the development of this project and are as follows:

3.3.1 Network Latency. The latency depends on the distance between the server and the client. Latency is a major issue on long distances and cannot result into an unusable product due to lagging when interacting with the data. Interaction has to be smoothing order to encourage usage of 3Davis as an astronomy visualisation tool. Moreover, the transition from low-resolution image rendered by the front-end to the high-resolution image rendered by the server has to be tackled accordingly to match the corresponding pixels and frames. Therefore, we intend on implementing latency hiding techniques and test the system by varying the latency factor. If interaction is considered to be an adequately smooth flow, the system will be deemed as a success.

3.3.2 Bandwidth Usage. As this project involves a client-server architecture, we also need to consider the network over which data is

transferred. Therefore, we will also test the system on different network bandwidths to experiment the limit of the system's usability.

3.3.3 Data Size. As modern telescopes continue to gather images of increasing quality, our system needs to be able to adapt to those enormous size as input for processing. Consequently, as a testing objective, we will vary the image size to experiment how the server handles the data processing and how the client manages the rendering and the interactions.

In addition to these benchmarks testing, we also intend to compare our system to other remote visualisation tools used in the astronomy industry and try to conform to the recent trends and functionalities.

4 ETHICAL, PROFESSIONAL, AND LEGAL ISSUES

There will not be tests conducted on users as the construction of this prototype is a proof of concept test and therefore we forgo any ethical issues involved in trials with a user group. However, if we deem later in development that user tests are required, we will reevaluate this section.

3DAVis will be developed under the MIT license for open source software and all the packages and libraries used in the construction and implementation of the system are confirmed to be open source as well. It is in the interests of the communities that helped to create these libraries and continually support the software free of charge that we keep our code open source as well.

5 RELATED WORK

The key works that were discussed are centered around the individual parts of the process of remotely rendering and visualising radio astronomy data. These parts relate to the structure of server client model, processing radio astronomy data files, compression of data for transfer over a network and the visualisation of scientific data in the context of a web browser.

To implement this system there are many existing packages and libraries available for each section, each with their own characteristics therefore the libraries that are chosen must be measured against each other using variables such as speed, size and scalability without sacrificing quality and accuracy of the visualisation.

There are various packages dedicated to rendering three dimensional graphics in a browser. Volume rendering is the preferred method for rendering astronomical data, where data points are rendered as three dimensional pixels (voxels) within a space and each point is given visual characteristics such as colour and opacity according to data parameters attached to the point. The technique is well suited for visualisations which lack defined boundaries and is thus appropriate for the diffuse nature of astronomical phenomena. This method might not be the most efficient because of how the rendering time for a dataset grows linearly with the size of the dataset. The best lighting technique to complement the structure of volume data is ray-casting, although it is computationally expensive and will take up more time and processing power. The libraries available for three dimensional visualisation that we are considering are all cross platform, compatible with all major browsers and are open source.

5.1 Visualisation Libraries

5.1.1 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.

5.1.2 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. These libraries are available as packages on NPM for easy integration with the chosen JavaScript framework.

The next key aspect of the system that has to be considered is the file format of the astronomical data. The file formats that are commonly used to store astronomical data are FITS and HDF5. FITS stores data in a sequential format where the data can only be accessed as a single stream which could affect the processing time of the data where all the data has to be processed in order to comprehend the model and there is no way to access the data in a parallel way. HDF5 addresses this by organising data into a hierarchical structure. This addresses the shortcoming of FITS however FITS is still the most commonly used and preferred format.

There is currently a selection of tools for the visualisation of radio astronomy data such as CARTA, SAOImage DS9, KARMA for two dimensional visualisation, and SlicerAstro for three dimensional visualisation. However, none of these current tools support remote visualisation and all their rendering is done on the client's system.

For the visualisation data to be sent from the server to the client in the fastest and most efficient manner some sort of data compression needs to take place. The compression of visual data can be done during rendering in the form of mipmaps where sections of the data are down-sampled to a single pixel, thus minimising the rendering of unnecessary detail and reducing the amount of data that needs to be rendered. Mipmaps are suitable for two and three dimensional visual data [10]. The data also need to be compressed for transfer over a network, lossy compressing is best suited for the hybrid approach because losing a few bits of information in a continuous stream will not have a noticeable impact on system performance. There are also various compression libraries available to add to the implementation.

5.2 Compression Libraries

5.2.1 ZFP. Is a lossy compression format which is used for multi-dimensional floating point arrays which achieves high compression ratios and facilitates high data throughput.

5.2.2 SZ. Is a compression method for scientific data in the form of floating point values.

The next aspect that had to be considered is how to showcase the visualisation on the client's browser. There are three major frameworks present in the front end development sphere where they all perform relatively the same functions and all using component based approaches for constructing cross platform web applications. Components are used to obtain input from the user and change their behavior based on that input, this behaviour manifests as visual feedback to the user. They also make it easy to reuse components within the web application without repeating code.

5.3 Front-end JavaScript Libraries

5.3.1 React. A framework for creating interactive user interfaces for web applications developed by Facebook. It combines the UI and behavioural functionality in its components and It operates under the MIT open source license.

5.3.2 Vue. Is an open source JavaScript framework under the MIT license. It also combines the user interaction and behaviour functionality into its components. Functional parts of pages like buttons and certain sections of a web page are implemented as separated into self contained components and can be combined in new components, this allows for components to be reused in various parts of the web page therefore reducing code repetition. Vue can make use of visualisation packages like VTK.js and WebGL by supporting packages that integrate the packages with the framework structure.

5.3.3 Angular. A JavaScript framework developed by Google under the MIT open source license and is much like Vue and React. Angular however, separates the UI and behavioural elements in its components.

The final aspect of the system that must be considered is how to implement the server where the majority of the processing will take place. There are also many libraries in various languages to consider, most of these libraries are open source and support cross platform development.

5.4 Web Application Libraries

5.4.1 Node.js. An open source Javascript library for accelerated building of web applications and is characterised by its ease of use and speed of implementation. Additional libraries and functionality can be added through the NPM package manager.

5.4.2 Drogon. A C++ HTTP application framework under the open source MIT license. Has much of the same functionality and structure of Node.js and has a large community for development support.

5.4.3 Crow. This micro-framework is built on C++ and functions the same as Flask for Python where it does not require particular tools or libraries to be included to operate, it also has no database abstraction layer, form validation or any other components that require third party libraries to provide common functions. It is also extremely fast due to its pared back architecture.

5.4.4 Django. Is a web application framework based on python and follows the model-template-view architecture pattern for rapid and easy application construction. It characterises itself as being fast, secure, and scalable and is under the open source 3-clause BSD license.

6 ANTICIPATED OUTCOMES

We expect to create a proof of concept for 3DAVis and showcase how it is an efficient method for remote rendering of three dimensional radio astronomy data, implemented on the server-client architecture. It is also anticipated that it will be an effective tool for the visualisation of scientific data. The initial implementation will be tested on small data sets, the sizes of which will be increased to observe how the system scales for larger data sets. Extremely large data sets are commonplace in the radio astronomy field and so it is imperative that the system maintains its performance as the data sets become larger.

This system addresses a need prevalent in current astronomy data exploration tools that are largely standalone client-based software applications. The performance of these systems is dictated by the client's hardware capabilities. Taking the processing and storage of the data away from the client's system and pushing it to a powerful server machine allows for much larger data cubes to be processed and explored. This also opens up the larger astronomy community to easier data cube exploration. Most available software also only visualises the data in a two dimensional format whereas 3DAVis will showcase the data in a three dimensional manner which is the preferred manner in which to visualise astronomy data.

The primary factor by which we will be able to determine if the implementation is a success is if the system scales efficiently and maintains its performance with a large data cube. This example data cube must be one that astronomers would actually want to use to explore and interact with for knowledge discovery. The ultimate goal is for 3DAVis to become a comprehensive data exploration tool for astronomers. The risks for this project are outlined in a risk matrix (Figure 2) which also includes how to monitor, mitigate, and manage these risks.

7 PROJECT PLAN

7.1 Risks

The risks for this project are outlined in a risk matrix (Figure 4) which also includes how to monitor, mitigate, and manage these risks.

7.2 Timeline

Our project runs from 17th May up until 18th October. Our Gantt chart (Figure 5) lists our project's timeline in more detail.

7.3 Resources Required

Software :

- Javascript libraries (nodejs, threejs, vtkjs , etc.)
- C++
- Vue
- Draco, LZ4
- Github

Hardware :

- High-end computers with Graphics processing unit (GPU)
- Stable internet connection

Human resources :

#	Risk	Probability	Impact	Monitoring	Mitigation	Management
1	Unclear Scope of the project.	3	8	Update project supervisor frequently on the works that is being focused.	Change work that has been done and try to keep as much as old work to reduce time lost.	Have a meeting with project supervisor, talk about desired scope and update current
2	Spend too much time on out of scope functionalities. Gold Plating.	6	4	Have more strict scopes.	Make sure only required work done if deadlines are not met by group.	Group leader should be more strict on work done by group members.
3	Workload is unevenly distributed among the group.	4	6	Regular meetings in group. Group leader communicates with every group member about their work.	Reduce the scope.	Redistribute workload between members to equalize the workload
4	Project Code is not meeting desired speed performance.	4	7	Throughout the project look for various bottlenecks and highlight them.	Improve algorithms used, Use different libraries provide better efficiency.	Make more research on different libraries. Update algorithms used.
5	Not meeting deadlines on Gantt chart.	3	7	Have regular meetings and figure out delays early.	If deadline is not met, update Gantt chart accordingly.	Increase daily work time until right targeted deadlines reached.
6	We are unable to develop a solution that satisfies	2	9	Regularly check how program functions as a whole and meets to all performance goals	Explain why project goals is not achievable by the algorithms/libraries used. Explain what could be changed or improved so in future same mistakes won't happen.	Closely follow why algorithms/libraries used over other algorithms/libraries and also test different libraries if possible.
7	Team member drops out of project	1	9	Maintain good communication and relationship among team members through out the project.	Ensure that each member properly understands other members' work.	Let the supervisor know about the situation,reduce the scope and redistribute the workload

Fig. 4. Anticipated risks that could possibly be encountered during the time of the project's development

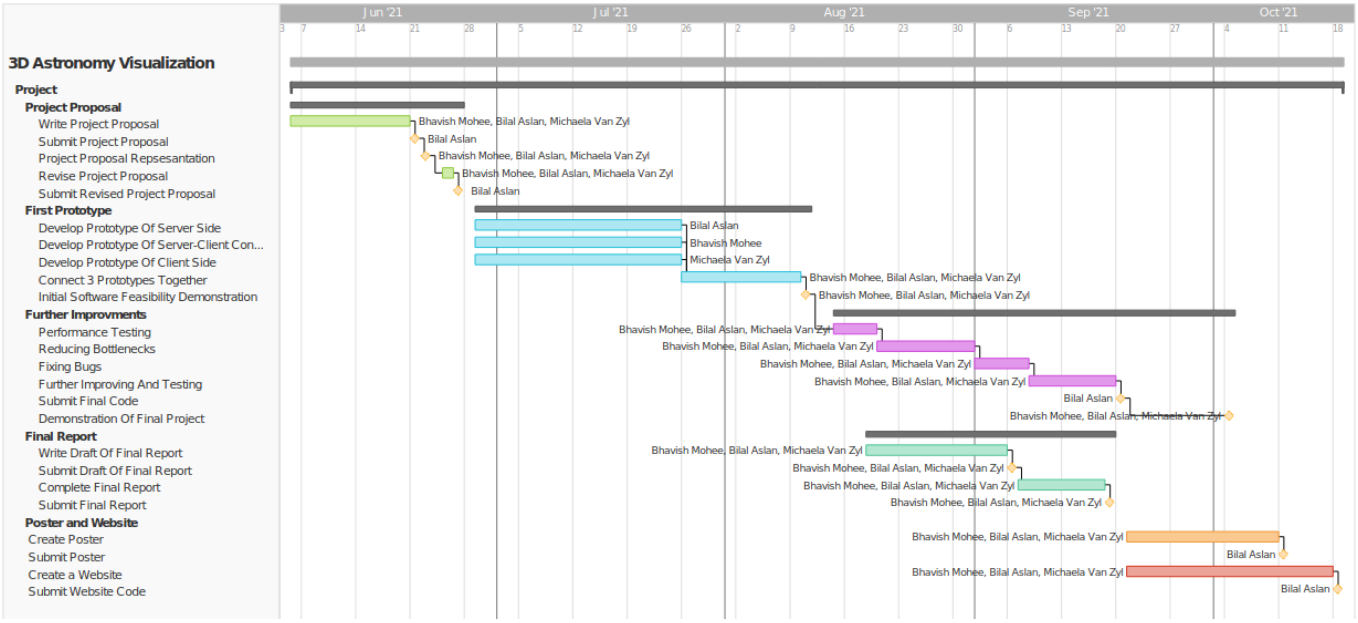

Fig. 5. The timeline outlining the sequence of the deliverables and milestones as well as the amount of time that will be dedicate to each.

Various advisers are required to ensure progress is being made on the project at all times. These include the project supervisor, Rob Simmonds, to guide the project development, project advisers, Angus Comrie to provide advice and information on tools, algorithms, libraries, etc.
Other resources:

- Astronomical data cubes

7.4 Deliverables and Milestones

- Project proposal - 4th till 21st June
- Initial software feasibility demonstration - 10th till 13th August

- Submission of draft final project paper - 6th September
- Submission of final project paper - 17th September
- Submission of final project code - 20st September
- Demonstration of the final project - 4th till 8th October
- Completion of the poster - 11th October
- Completion of the web page - 18th October

7.5 Project Allocation

As it is mentioned in gantt chart, work will be split amongst the team members in the following way:
Bilal Hasan Aslan will focus on the server-side of the project. He will be responsible for preprocessing astronomical data to make processing easier and processing data so that the front-end can render the data. He also will be responsible for preparing requested data when the interaction occurred on the render.
Bhavish Mohee will focus on the client-server interaction. He will be responsible for sending/receiving data between client-server and compression/decompression of the data.
Michaela Van Zyl will focus on the client side of the project. She will be responsible for rendering received data from the server, interactiveness of the render, User-friendly GUI, and other GUI parts of the render to display extra information.The risks for this project are outlined in a risk matrix (Figure 2) which also includes how to monitor, mitigate, and manage these risks.

REFERENCES

[1] Joshua Bloch. 2006. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 506–507.

[2] Michelle Borkin, Alyssa Goodman, Douglas Alan, Jens Kauffmann, and Michael Halle. 2007. Application of medical imaging to the 3d visualization of astronomy data. In *Proceedings of IEEE Visualization Conference*.

[3] Fernando Camilo. 2018. African star joins the radio astronomy firmament. *Nature Astronomy* 2, 7 (2018), 594–594.

[4] Carolina Luiza Chamas, Daniel Cordeiro, and Marcelo Medeiros Eler. 2017. Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: An energy cost analysis. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*. IEEE, 1–6.

[5] 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.

[6] Willem H De Boer. 2000. Fast terrain rendering using geometrical mipmapping. *Unpublished paper, available at http://www. flipcode. com/articles/article geomipmaps. pdf* (2000).

[7] Marcus D Hanwell, Kenneth M Martin, Aashish Chaudhary, and Lisa S Avila. 2015. The Visualization Toolkit (VTK): Rewriting the rendering code for modern graphics cards. *SoftwareX* 1 (2015), 9–12.

[8] Pawan Harish and Petter J Narayanan. 2007. Accelerating large graph algorithms on the GPU using CUDA. In *International conference on high-performance computing*. Springer, 197–208.

[9] Justin Jonas and MeerKAT Team. 2018. The MeerKAT radio telescope. *Proceedings of MeerKAT Science: On the Pathway to the SKA* (2018), 25–27.

[10] 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

[11] 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.

[12] Donald Carson Wells and Eric W Greisen. 1979. FITS-a flexible image transport system. In *Image Processing in Astronomy*. 445.