Server-Side Rendering of Large Astronomical Data Cubes

Honours project proposal

Jonathan Weidaman
University of Cape Town
Cape Town, South Africa
jonoweideman1@gmail.com

Shuaib Parker
University of Cape Town
Cape Town, South Africa
prkshu001@myuct.ac.za

1 PROJECT DESCRIPTION

Astronomy as a scientific field relies heavily on visualization. Modern telescopes collect data at an ever-increasing rate [3], and visualization enables astronomers to effectively analyze these large sets of data [6]. The data collected by these telescopes are usually represented as a three-dimensional (3D) array and are commonly referred to as data cubes.

Despite the 3D nature of these cubes, most astronomical visualization tools focus on 2D visualization. This is usually achieved by rendering a 'slice' of the data cube along different axes. While these 2D views are useful in their own right, it is clear that visualizing these data cubes in 3D would provide astronomers with additional insight. [6, 8]

The sheer size of these data cubes poses another problem for visualization tools. Most of the current software offerings are traditional desktop applications that cannot efficiently process these data cubes, due to the limited power of a typical consumser laptop or desktop.

CARTA [1] is an open-source software tool that tackles this problem. It uses a client- server architecture that delegates most of the expensive computation to a remote server, and then streams a portion of the processed data to the end-user's browser for rendering. This makes it possible for the user to visualize and interact with very large data cubes fairly efficiently. However, CARTA only offers 2D visualization, and the CARTA team are keen on adding 3D visualization support.

The aim of this project is to help CARTA achieve this by developing a 3D visualization prototype that is capable of effectively rendering large astronomical data cubes.

2 PROBLEM STATEMENT AND AIMS

Current astronomical visualization tools typically suffer from two major issues :

- The inability to render large data cubes efficiently
- Sub-par or non-existent 3D visualization capabilities

CARTA handles the first problem very well, but it lacks any form of 3D rendering capabilities.

The aim of this project is to provide CARTA with a 3D visualization prototype that can be integrated into their system. Our prototype will not extend CARTA directly, but will employ a similar client-server architecture and use a similar tech-stack in order to ensure that future integration into CARTA is as seamless as possible. Initially, we considered integrating into CARTA directly, but by consulting our supervisor and CARTA's head developer, it was determined that the direct integration into CARTA would involve too many integration problems that wouldn't necessarily be interesting for the purposes of the project, and would drain a lot of our time.

It is also important for our prototype to render the 3D model at a high enough frame-rate and low enough latency to allow the user to easily easily interact with it.

3 PROCEDURES AND METHODS

This section will offer a brief overview of some of the scientific visualization tools relevant to our project and discuss what we can learn from them.Our goal is to develop a software tool that will allow the user to view and interact with a 3D representation of a data cube in his / her web browser. In order to achieve this we will employ a client-server architecture, with a C++ server performing the computationally expensive processing and rendering. The server will then downsample the data and send it to a React based web-application that can be used to view and interact with the processed data.

The volume rendering itself will be written using VTK, as this provides us with a powerful GPU-accelerated rendering method out of the box. We will be using gRPC for the API layer to communicate between the two. These elements are explained in more detail below.

3.1 Back-end

The server will provide most of the heavy lifting when it comes to rendering. C++ is our preferred language for the server, since it provides high performance, control over memory management (which will be required due to our memory intensive tasks), support of many libraries which are widely used in astronomical visualization (e.g. VTK, CUDA) and is widely used by developers in this field which will lead to a higher level of impact. The system will initially support data in the flexible image transport system (FITS) [9] format.

When a data cube is selected by the user for visualization, the server initially generates a high-quality image of the data cube, and this is streamed to the web-based client. In addition, a low-resolution LOD (level-of-detail) model will also be generated by the server and sent to the client.

The client will then be able to interact with the data by rotating, zooming in on areas of interest etc. When the user interacts with the data in this way, the high-resolution view will be replaced by the LOD model which gets rendered and displayed by the front-end. This will allow the user-driven changes to the 3D view to reflect rapidly, and with minimal delay.

However, when the user stops interacting for a short period of time (approximately 200ms), a message is sent to the server and a high-quality view of the current LOD model is generated and streamed to the user. Once this view is received by the front-end, it updates the view of the existing model. If the user interacts with the model after this, the view returns to the LOD model.

The resolution of the LOD model, relative to the resolution of the full model stored on the server will have to be experimented with. Since the LOD model is fully rendered on the user's browser, a resolution which is too high may lead to significant drop in FPS, preventing the interaction with the data cube from being seamless. As outlined by R. Norris, this is considered as an essential feature in a 3D astronomical visualization tool [6]. We may wish to experiment with what level of performance we can achieve on machines with various hardware and customize the level of detail accordingly.

We will also experiment whether the level of detail should be calculated proportionally to the resolution of the full data cube or should be fixed regardless of it. This is because, if the original data cube has a resolution which is "too" high (relative to the specs of the user's PC), a fixed level of detail might produce a model where so much information is lost that the visualisation of it essentially becomes pointless. In these cases, the best solution might be to produce a model with a LOD proportional to the original at the expense of interactive performance, but maintain enough information in the visualisation to allow the user to select a subset of the model to be visualised, at which point interactive performance could be achieved. All of this will have to be experimented with to determine the best solution.

3.2 Front end / client

The main aim of the front-end/client is to allow smooth and interactive exploration of the 3D data.

The user will have access to several features which they can use to interact with the data. They will be able to rotate the LOD model by clicking and dragging it with a mouse, and will be able to zoom in and out using the mouse's scroll wheel. As mentioned before, when the user is exploring the data in this way, it will render a low-res LOD model provided by the server and this low-res view will be the one that the user interacts with. Once the user stops interacting with the view, the high-resolution image will be streamed in and replace the LOD model.

The user should also be able to select a subset of the data cube, or a region of interest (ROI), to be visualized in higher detail. The exact mechanism to perform this function will have to be experimented with in order to provide and intuitive and interactive solution.

One potential solution is to allow the user to click and drag from one point to another on the current view of the model to create a 2-dimensional rectangle. Once this is done, the model can then be rotated so the drawn square becomes orthogonal to the display (so that the rectangle appears as a line), and the user can then click and drag the face of the rectangle to another point in the model, essentially adding in the 3rd dimension of the rectangular cuboid. Another option is that after the 2D rectangle is drawn, the user could be prompted for the length of the 3rd dimension by a dialog box in the GUI . All of these will have to be experimented with to determine which is the most intuitive from the perspective of the user. If some techniques prove to be optimal in certain cases but not in others, the final system could contain multiple options for the user.

Another interactive feature which would be of use to astronomers would be the ability to change the mapping of voxel values to colours and opacity, i.e. the transfer function. This would enhance the user experience and improve the ability to analyse data cubes by increasing the likelihood of visually detecting features within them. An intuitive implementation of this would be to provide a widget, which contains labeled sliders for each of these functions. The colour transfer function will likely be a RGB function and hence three sliders will be provided – one for each value. A separate slider will be provided for opacity. As the user adjusts the slider, the LOD model should be recomputed and updated in real time.

3.3 Communication layer / API

In order to communicate and send data between the front-end and back-end we will make use of gRPC. gRPC is an alternative to traditional API frameworks such as REST and OpenAPI. gRPC is language agnostic,and will allow the client to call methods directly from the back-end.

Although vanilla gRPC does not support web-based clients, gRPC has a web-based package called gRPC-web which we will make use of.

gRPC should be sufficient for most of our client-server calls, but we will explore various other technologies (such as web sockets) when trying to stream the actual image data, since this needs to be done as efficiently as possible. Once we have found the most efficient method, it will be included in the final version of our application.

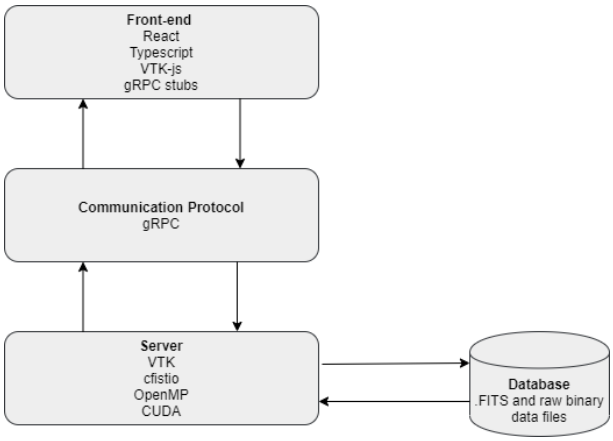

Figure 1: System diagram of our software prototype

4 EVALUATION METRICS

The main evaluation criteria we will be using will concern the speed and efficiency of our software system. Speed is a key aspect we are designing for, since our software system needs to serve as a prototype for a production system that has to realistically handle very large data cubes.

While user interaction will eventually be a key part of a production system, we won't be performing user based testing, since user-based design and testing is a large problem in of itself and focusing on it could detract from what the prototype is trying to achieve: ie. a scalable 3D visualization solution.

Our performance-based metrics are shown below:

4.1 Frames per second

This metric will measure the amount of high-resolution frames per second we can deliver from the backend to the frontend. A decently high FPS is important to provide the user with a smooth enough experience that allows them to interact with the 3D data easily. Higher FPS values also indicate that our solution could be scalable when dealing with larger data cubes. We will consider an FPS above 5 to be satisfactory for user interaction. FPS can be measured easily by VTK's "framerate" function.

4.2 Round trip latency (ms)

A round trip in our case will be defined as the time from the moment the front-end makes a render request to the moment it receives the image data. This time will be measured in milliseconds (ms). A low latency is important because a render request is likely to be sent whenever the user stops interacting with the data. High latency will result in significant lag and will hamper the user experience. We will consider a round trip latency of under 50ms to be acceptable for our prototype. To measure this, we could make use of a front-end library such as Axios to give us the round trip time in milliseconds.

5 RELATED WORK

This section will offer a brief overview of some of the scientific visualization tools relevant to our project and discuss what we can learn from them.

5.1 KARMA and Ds9

Two of the most popular astronomical visualization tools are KARMA [2] and DS9 [4]. However, these tools are traditional desktop applications and struggle to process and render large data cubes. They require the entire cube to be loaded into the main memory of the PC, which is impossible in some cases, since the data cube size can often exceed the size of the main memory [3]. Both KARMA and Ds9 also fail to make use of the graphics processing unit (GPU) for rendering, severely limiting their overall performance.

5.2 CARTA

CARTA is much more efficient at handling large data sets, since it employs a client-server architecture. It delegates most of the computation and storage to enterprise-class servers, and then sends a subset of the processed data to the front-end. The front-end then renders this data and displays the view to the user. We plan on emulating CARTA's client-server approach since it will allow our solution to scale up when dealing with very large data cubes. However, since CARTA is only capable of 2D visualizations, it is able to get away with full client-side rendering. We will not be able to achieve this, since 3D volume rendering is computationally expensive and cannot be efficiently performed on a typical personal computer. Most of our rendering will therefore be performed on the server-side.

5.3 3D slicer and VTK

3D-slicer is an open-source 3D-visualization tool geared towards the medical field. While not directly used for astronomy, 3D-slicer is capable of creating interactive 3D visualizations which makes it especially interesting for our project. 3D-slicer is built on top of The Visualization Toolkit (VTK).

VTK is an open-source graphics library that implements powerful 3D rendering techniques such as ray-casting [7] to produce high quality 3D models. We will be using VTK in this project for both our client-side and server-side rendering. 3D-slicer therefore provides us with a VTK implementation that we can learn from. VTK leverages the power of the graphics processing unit (GPU) by using Nvidia's CUDA [5] under the covers. This allows it to run highly parallel code much more efficiently than pure CPU-based code.

6 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

This study involves the development of a web application where all software, libraries and data files used are free and in the public domain. Hence, there are no legal issues to be addressed. The software will follow the guidelines and rules for developing Open Source Software (OSS), so there will be no intellectual property issues to deal with. Sufficient documentation and user guides will be produced and made available online and these will be written in a professional style. If the researchers are contacted, they will conduct themselves in a professional manner as well as in any environment where the project may be showcased. Since this project does not involve any human subjects, there are no ethical issues to address.

7 ANTICIPATED OUTCOMES

In this section we detail the results we expect to see at the end of the project. This is important to consider prior to the beginning of the project as it will guide many decisions about the scope of the project, the design of the software, which features to prioritise during development and how to test the software.

7.1 System

As mentioned before, we aim to create a 3D visualization tool that enables large astronomical data cubes to be explored. This will be in the form of a client-server-based application, with a C++ backend server and a React-based web client. We consider the following features key to our system:

Server:

- Generation and rendering of a high-resolution 3D model.
- Streaming of a high-resolution view to the frontend.
- Generation and streaming of a low-res LOD model.

Client:

- Rendering of an interactive low-resolution 3D model.
- Interactive features such as panning and zooming in on the data.
- Region of interest selection.
- Customizable transfer functions (optional depending on time).

7.2 Impact

Our project is intended to serve as a proof of concept for 3D visualization of large astronomical data cubes through the use of a client-server architecture, in particular where the rendering of the data cubes is done on the server before being streamed to the user. We expect our impact to be the interest of astronomers and for future 3D astronomical visualisation applications to utilize a similar architecture.

7.3 Key success factors

We will judge the success of our project based on the following criteria:

- The server is able to read in and construct models from FITS files.
- The generation of LOD 3D models on the server which are streamed to the user.
- The LOD model is interactive in the user's browser in the following ways: zoom in and out and rotate using a mouse, ROI selection and customisable transfer functions using sliders.
- Rendering of LOD models as well high quality images done on the GPU of the PC the server is running on.
- The performance on the user's browser maintains interactive speeds.
- The visualisations of data cubes is accurate (i.e. if the same data cube is visualised using another visualisation tool, they both produce similar visualisations).

8 PROJECT PLAN

8.1 Risks

The risks for this project are outlined in the risk matrix which is presented in Appendix A. This matrix describes each risk, how we plan to mitigate and monitor it as well as manage it in the case that it occurs. A probability and impact from one to ten is assigned for each risk, where 1 translates to a low probability and low impact to ten being a high probability and high impact.

8.2 Timeline

Our Gantt chart can be found in Appendix B

8.3 Resources required

Each team member will require a PC with an integrated GPU, a stable internet access in order to access a remote GitHub repository and the necessary tools (such as IDEs and compilers) in order to develop the application in C++ and TypeScript. Astronomical data cubes also need to be provided in order to test our software system.

8.4 Deliverables

- Presentation of feasibility demo - 3rd till 7th August
- Submission of paper draft - 4th September
- Submission of final paper - 14th September
- Submission of final code - 21st September
- Presentation of final demo - 5th till 10th October
- Completion of web page - 12th October
- School of IT showcase - 15th October
- Completion of poster design - 21st October

8.5 Milestones

- Presentation of feasibility demo
- Completion of application (bare minimum before improvements, additional features, etc)
- Testing framework completed
- Project code finished
- Submission of draft of final report
- Submission of finished final report
- Final presentation
- Completion of web page
- Completion of final poster

9 WORK ALLOCATION

Shuaib will be responsible for most of the front-end development as well as the communication protocol between the client and the server. The front-end will comprise of a web application that displays an interactive view of the cube. The front-end also needs to render a low resolution view of the data cube while the user is interacting with it. The communication protocol between the client and the server will need to handle the transfer of LOD data cubes and high quality renders from the server to the client as well as commands and requests from the user to the server.

Jonathan will be responsible for most of the back-end (server) development. This will include the storage and access of the data files used in visualization, the efficient generation of LOD models, the rendering of high quality images, processing of data cubes based on user input, compression of data prior to being sent to the user and the creation of a server which can operate autonomously and handle multiple users at once.

REFERENCES

[1] Angus Comrie, Kuo-Song Wang, Pamela Harris, Anthony Moraghan, Shou-Chieh Hsu, Adrianna Pińska, Cheng-Chin Chiang,

Hengtai Jan, Rob Simmonds, Tien-Hao Chang, and Ming-Yi Lin. CARTA: The Cube Analysis and Rendering Tool for Astronomy, December 2018.

[2] Richard Gooch. Karma: a visualization test-bed. In *Astronomical Data Analysis Software and Systems V*, volume 101, page 80, 1996.

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

[4] WA Joye and E Mandel. New features of saoimage ds9. In *Astronomical data analysis software and systems XII*, volume 295, page 489, 2003.

[5] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[6] Ray P Norris. The challenge of astronomical visualisation. In *Astronomical Data Analysis Software and Systems III*, volume 61, page 51, 1994.

[7] Scott D Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144, 1982.

[8] Melanie Tory. Mental registration of 2d and 3d visualizations (an empirical study). In *IEEE Visualization, 2003. VIS 2003.*, pages 371–378. IEEE, 2003.

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

APPENDIX A: RISK MATRIX

Risk	Probability	Impact	Consequence	Mitigation	Monitoring	Management
Not finishing on time.	3	10	Our work cannot be used as a proof of concept since it's incomplete. This will result in us failing the course.	Use efficent software development methodologies. Have weekly meetings with supervisors and daily meetings between researchers.	Update Gantt chart throughout the project. Ensure deadlines are met.	Ensure literature portion of the project is complete. Ensure the what work has been done is presentable. Reduce the scope.
Team member drops out of degree.	1	9	Increased workload for remaining member.	Maintain mental well being and limit stress as much as posible. Ensure each member understands the other members work.	Maintain good communication throughout project.	Reduce the scope of the project and redistribute workload.
Certain libraries and algorithms are too difficult to implement.	5	4	Time is wasted working on something which won't be used in final software.	Ensure multiple techniques (algorithms) and libraries which accomplish the same task are explored in advance.	Test algorithms and lirrary implementations iteratively.	Use a simpler library or solution at the expense of customization and performance.
Not enough time to run perofrmance tests and explore alternative solutions.	7	2	Work will still be complete, however the impact the project will have on the associated field of research will be reduced.	Add this step as a deliverable for the project. Ensure all deliverables are met on time.	Have weekly meetings. Update Gantt chart throughout project.	Discuss further research and alternative libraries and solutions which could be used in final report.
Workload is unevenly distributed.	6	2	One member will have to do some of the work which was originally allocated to the other member.	Allocate work once a thourough understanding of the workload is atained or allocate work iteratively.	Maintain regular communication (specifically about work being done) between team members.	Reallocate workload.

Figure 2: Risk matrix

APPENDIX B: GANTT CHART

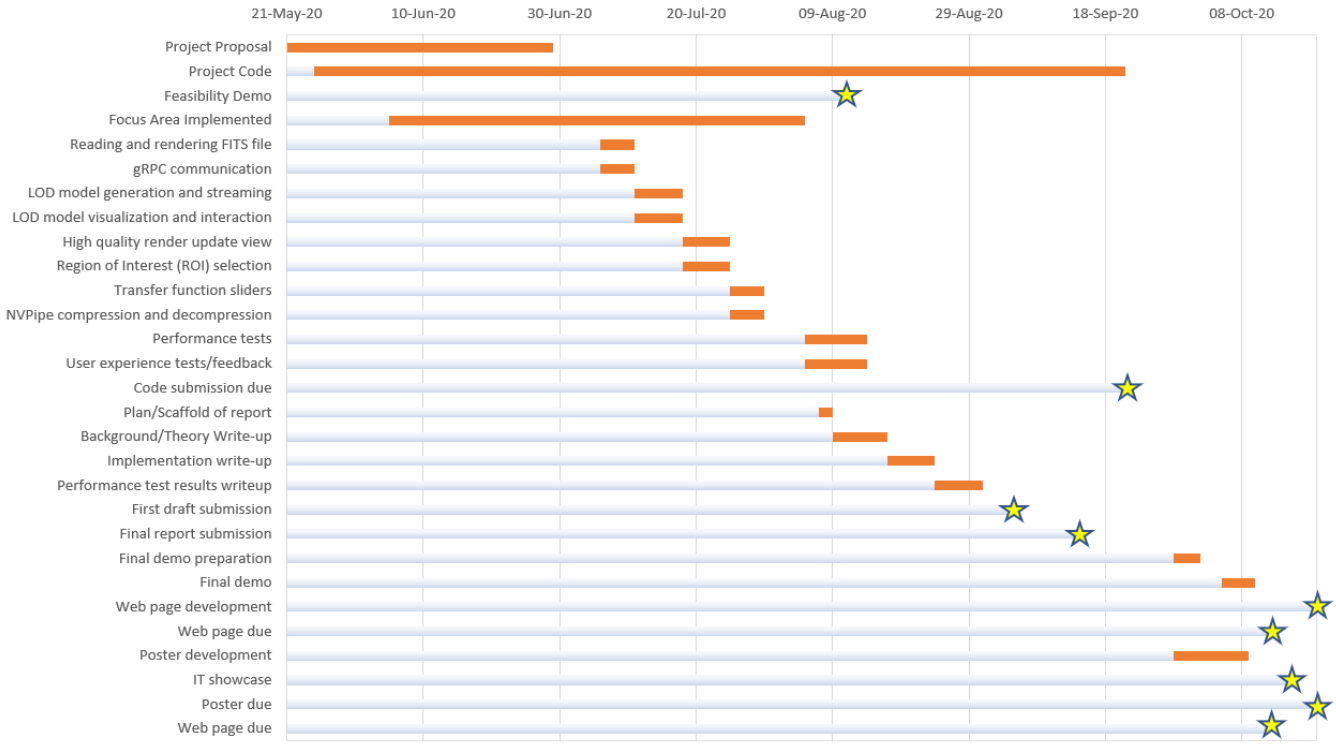

Figure 3: Gantt chart