Scalable desktop visualization of very large radio astronomy data cubes.

Simon Perkins[a], Jacques Questiaux[a], Stephen Finniss[a], Robin Tyler[a], Sarah Blyth[b], Michelle M. Kuttel[a,*]

[a]*Department of Computer Science, University of Cape Town, Private Bag X3, 7701, Cape Town, South Africa*
[b]*Astrophysics, Cosmology and Gravity Centre (ACGC), Department of Astronomy, University of Cape Town, South Africa*

Abstract

Observation data from radio telescopes is typically stored in three (or higher) dimensional data cubes, the resolution, coverage and size of which continues to grow as ever larger radio telescopes come online. The Square Kilometre Array, tabled to be the largest radio telescope in the world, will generate multi-terabyte data cubes – several orders of magnitude larger than the current norm. Despite this imminent data deluge, scalable approaches to file access in Astronomical visualisation software are rare: most current software packages cannot read astronomical data cubes that do not fit into computer system memory, or else provide access only at a serious performance cost. In addition, there is little support for interactive exploration of 3D data.

We describe a scalable, hierarchical approach to 3D visualisation of very large spectral data cubes to enable rapid visualisation of large data files on standard desktop hardware. Our hierarchical approach, embodied in the *AstroVis* prototype, aims to provide a means of viewing large datasets that do not fit into system memory. The focus is on rapid initial response: our system initially rapidly presents a reduced, coarse-grained 3D view of the data cube selected, which is gradually refined. The user may select sub-regions of the cube to be explored in more detail, or extracted for use in applications that do not support large files. We thus shift the focus from data analysis informed by narrow slices of detailed information, to analysis informed by overview information, with details on demand. Our hierarchical

*Tel:+27 21 6505107, email: mkuttel@cs.uct.ac.za

solution to the rendering of large data cubes reduces the overall time to complete file reading, provides user feedback during file processing and is memory efficient. This solution does not require high performance computing hardware and can be implemented on any platform supporting the OpenGL rendering library.

Keywords: Methods: data analysis, Techniques: miscellaneous, visualization

1. Introduction

Astronomers have long used graphical representations of observational data for research and educational purposes alike. Three-dimensional (3D) visualisation is useful for the interpretation of spectral data cubes from both radio- (Norris, 1994) and optical telescopes, as, compared to 2D, it allows for improved comprehension of global features, enhanced pattern identification and easier detection of instrumental and data processing errors (Hassan et al., 2011). 3D views help users to interpret data qualitatively (Tory, 2003) and are also useful in conveying the meaning of information to non-experts. However, existing data visualisation packages remain focussed on 2D images, such as channel maps and spectrum arrays, and few existing data visualization packages support interaction with very large 3D datasets.

Another problem is the sheer size of the data files. Radio telescopes observe many different radio wavelengths, or frequency channels and images can be formed from observations across a range of frequencies (according to the bandwidth of the radio receiver) for specific locations in the sky. While frequency is not a spatial quality, images comprising consecutive frequency bands are stacked on top of each other to form a 3D voxel volume. These spectral line cubes can be several gigabytes in size and will rise to terabytes for the next generation of radio telescopes. Therefore, even with the increasing memory sizes of modern computers, it not expected to be practicable to store entire cubes in main memory. Indeed, effective handling of large datasets has been identified as one of the six Grand Challenges for the petascale Astronomy era (Hassan and Fluke, 2011).

However, despite this imminent data deluge, current Astronomical visualisation and analysis applications (such as Karma (Gooch, 1996) and others listed in Hassan and Fluke (2011)) typically do not implement scalable approaches to file access. In general, the standard approach to visualising data

cubes used by Astronomy applications is focussed on the display of the entire detailed, full resolution data cube, or else sections thereof. Software packages either do not support access to large astronomical data cubes that do not fit into computer system memory at all (the *larger-than-memory* data size problem identified in Hassan et al. (2011)), or else support access only at a serious performance cost.

Prior solutions to the *larger-than-memory* data size problem used the same high-resolution approach and employed High Performance Computing hardware to permit interactive visualization of very large data cubes (Beeson et al., 2003; Hassan et al., 2011). In the most recent of these, Hassan et al. (2011) partition a data cube into smaller volumes that are distributed over a heterogenous cluster of CPUs and GPUs. A GPU-based ray casting volume rendering generates images for each sub-volume, which are composited to generate the whole volume output, and returned to the user. However, though effective, this is not a very satisfactory solution for the standard Astronomical researcher working on a desktop PC: 3D volume rendering of an entire large cube at interactive frame rates (i.e. better than ~ 5 frames per second (fps)), is well beyond the capability of a single, standalone workstation. A cluster- or web-based approach, as in the VisIVO software (Bandieramonte et al., 2013), allows for the use of a distributed cluster as a remote service to avoid costly data transfers (Fluke et al., 2010). However, while a remote service is clearly a desirable option, there are likely to remain situations where a desktop solution will be useful or necessary. For example, where internet access is unreliable/unavailable or where guaranteed access to the data is required.

A common desktop work-around for the larger-than-memory data size problem is to divide the data. This often involves a large window displaying a static, or non-user controlled, view of a point in the file. Changing the position removes the old information and replaces it with new information. With file sizes growing, this approach of working with detailed sections of data is becoming increasingly ineffective. In addition, a global view of the dat may be critical for global quality control of data cubes (Fluke et al., 2010).

Furthermore, the ratio of seen-to-unseen data is decreasing rapidly. While global views are still necessary and can play a vital role as a quality control tool, and may aid in discovering systematic and non-systematic noise effects (Hassan et al., 2011), such all-or-nothing rendering strategies provide high visual fidelity not strictly necessary for identifying general structures within

particular frequency ranges. As such, an informed approach to visualisation combined with an approximate representation of the cube is desirable.

1.1. Contribution

We explore an alternate approach, aiming to provide an intuitive and effective method for visualisation and exploration of extremely large files on a standard commodity PC with limited memory capacity. We seek to shift the focus from analysis informed by narrow slices of detailed information, to analysis informed by overview information, with details on demand. The focus is on rapid initial response: our prototype astronomical cube visualisation system, termed *AstroVis*, presents a reduced 3D representation of a large data cube, enabling sub-regions of the cube to be focussed on and explored in more detail. This is accomplished by, firstly, approximating voxel cubes in main memory via a *down-sampling* process and, secondly, iteratively streaming in data from the voxel cube.

System responsiveness is a key priority: our approach offers approximate visual fidelity that is iteratively refined over time. This iterative refinement occurs while the user is viewing the data: the user is still able to visualise and interact with the current approximation. They can also isolate their view to specific portions of the cube; which in turn directs the iterative refinement process to only read visible portions of the cube.

While CPU/GPU cluster based approaches to cube rendering offer real-time visualisation at full visual fidelity, this is accomplished through the use of complex parallel algorithms and the expense of a cluster. By contrast, our approach offers the ability to visualise approximations of cubes, which are iteratively refined during visualisation on commodity (desktop) hardware.

2. System Design and Implementation

The *AstroVis* astronomical visualisation prototype architecture is designed to support efficient viewing of large datacubes on a commodity personal computer. Therefore, it is designed to efficiently stream portions of a datacube from disk while continuously updating an in-memory cube approximation. While these processes occur, the user must be able to view and manipulate the cube. Responsibility for meeting these requirements is provided by three core modules: a **File Access Component**, an **Image Processing Component** and a **User Interface Component**. Each component runs in its own thread and are described in detail below.

- At the lowest level, the **File Access Component** (FAC) implements a Flexible Image Transport System (FITS) file reader and writer. FITS is the open standard digital image file format commonly used for representing astronomical data (Pence et al., 2010). To ameliorate the impact of very large file sizes on performance, the FAC aims both to minimise memory requirements, by avoiding loading an entire FITS file into system memory at any one time, and to service memory requests rapidly, through the use of a data caching strategy.

- The **Image Processing Component** (IPC) converts FITS file data returned by the FAC into images. Images can be requested in either the overview or detail display. The overview representation presents a down-sampled view of the entire image dataset. The detail display is typically invoked to view subsections of the data in greater detail.

- At the top level, the **User Interface Component** (UIC) aims to enhance file navigation and exploration through files of especially large sizes and resolutions. Images returned by the IPC are volume rendered (details below) and tools for effective navigation through the rendered image data are provided. Given data sets of ordinarily unmanageable size, our user interface allows the user to rapidly find and focus on the information most relevant to them and ignore irrelevant and noisy information.

  In summary, the IPC requests FITS data from the FAC in order to construct an image. In turn, the UIC requests images from the IPC for the purposes of rendering FITS data for the user.

2.1. Down-sampling in the IPC

Downsampling occurs once and is on-going until completion. The first iteration of the down-sampler provides a rapid rough estimate of the data cube and subsequent iterations serve to improve the accuracy of the down-sampled view. We thus focus on reducing the initial response time (which is important for users) rather than the total time to generate a complete view.

The IPC generates an approximation to the full 3D FITS voxel cube, which is configured to be small enough to fit in the main memory of a commodity computer. To do this, each voxel in the in-memory approximation cube is mapped to an entire voxel sub-cube in the original FITS file. This is accomplished by subdividing each FITS voxel cube dimension by those of

the in-memory cube. Where dimensions are not exactly divisible, sub-cube dimensions are rounded either up or down. For example, Figure 1 illustrates a $9 \times 9 \times 9$ FITS voxel cube with the corresponding $3 \times 3 \times 3$ in-memory cube.

To calculate an average value for each in-memory voxel, the FITS voxel sub-cube is *down-sampled* and then averaged. The FAC implements a streaming approach to the down-sampling process to avoid the delays associated with reading the entire FITS file for averaging. In addition, the IPC relies heavily on threading for improved performance, where each thread does the fairly simple task of averaging pixels.

2.2. Streaming in the FAC

For large FITS files, loading and then down-sampling the entire cube would decrease the responsiveness of the UIC. Therefore, we implement a streaming approach to loading and down-sampling in the FAC, allowing users to rapidly view the general outline of structures within the cube. A user can immediately interact with these initial observations to direct and refine the visualisation to specific portions of the cube.

The FAC streaming process is conceptually divided into successive passes, which iteratively refine the visualisation (Figure 2) until the entire FITS file has been read. During a pass, slices along the z-axis of the FITS voxel cube are read into main memory and used to improve the in-memory voxel approximation. This is accomplished through the use of an accumulation buffer, which holds the sum of each sub-cube, and a count buffer, which records the number of sub-cube voxels read. At the end of each pass, each in-memory voxel is assigned its respective value in the accumulation buffer, divided by the voxel count in the count buffer. Slices are read in an interleaved sequence to evenly distribute updates among in-memory voxels. Figure 3 illustrates the process. Pass One provides the initial estimate of the in-memory voxel cube, as follows. First, a 9x9 FITS voxel slice whose values contribute to nine in-memory voxels is read. Each 3x3 block is averaged and assigned to an in-memory voxel. Then, the second and third slices, each containing partial data for nine different in-memory voxels, are read in to provide a complete initial estimation of the FITS voxel cube.

The second and subsequent passes further update and refine the initial estimates of the in-memory voxels, until all voxels of the FITS sub-cube have been read. After each FITS voxel slice is read, the associated in-memory voxels are updated and displayed to the user (Figure 4). As voxel cubes

are stored in row major order contiguously on a magnetic hard disk, reading a voxel slice is an efficient disk access pattern. There will be significant contiguous reads performed before it is necessary to reposition the hard drive head to read the next slice (Arpaci-Dusseau and Arpaci-Dusseau, 2013). The frame rate is dependent on the output resolution of the cube and remains constant throughout the process.

2.3. User Interface Component

We followed the visualisation approach of Overview and Detail, described by Schneiderman (1996), showing a low resolution view of the whole dataset at all times, while allowing the user to pursue detailed high-resolution analysis of a specific selected sub-region. An annotated screenshot of the user interface is shown in Figure 5. Once the application has been started, the user selects a FITS file that they are interested in viewing, using the File Menu $\rightarrow$ Open option. This process only loads the *ASCII headers* within the FITS file into the application, which are displayed as a list in the status window. Clicking once on a list item displays the dimensions of the voxel cube associated with the header in the status window. Double-clicking a list-item displays a dialog which allows the user to specify the dimensions of the in-memory voxel cube.

Once the dimensions have been specified, the specified FITS voxel cube is streamed and down-sampled into the in-memory voxel cube. This approximation is rendered in the overview display using the Cabral et al. (1994) method. A series of OpenGL quadrilaterals are projected, perpendicular to the viewers direction, into the 3D cube, and are rendered as texture slices taken across the cube.

Furthermore, the portion of the cube that the user wishes to view can be modified via the use of sliders, attached to the x, y and z dimensions, that adjust the six clipping planes. This view refinement automatically instructs the down-sampling component to only load data related to the specified portion of the cube, further increasing the responsiveness of the visualisation.

By double-clicking a cube face, the user is able to select a FITS voxel slice to be displayed as an image in the detail display, which shows a full resolution, axis-aligned, voxel slice from the FITS file. On selecting a slice, a down-sampled version (obtained from the overview display) is initially displayed and then replaced by high-resolution data, as follows. Any other down-sampling file reads are paused and the requested FITS voxel slice is read into memory and rendered on the detail display. Reading slices on the

xy plane is most efficient, since the FITS voxel data is row contiguous on disk. Slice requests from the xz plane also have some contiguity and are relatively efficient. However, slice requests from the yz plane have no contiguity, resulting in scattered disk reads and slower read times. Users have the ability to pan over the detail display and zoom within it to view important or interesting features. Sliders on the colour and opacity maps can be manipulated to assign different colour ranges and opacities to voxel intensities. When moving the mouse over this area, the corresponding voxel location in the FITS file is also provided to the user. The region of the image data currently displayed in the detail display is always indicated in the Overview Display, providing the user with a reference point for further navigation.

Panning and zooming tools are used to navigate through the data. Subregions may be saved in separate FITS files, for later analysis. The end result is a system which reduces search time, allows the detection of overall patterns, and aids the user in refining their search.

2.4. Implementation Detail

The *AstroVis* prototype was developed for the 64 bit Ubuntu Linux platform in C++, with the Qt API used for the Graphical User Interface components and OpenGL graphics API for display of a 3D representation of the data cube. All testing was done on an Intel i7 3 GHZ, 8 GB of RAM, a 1TB 7200 RPM hard disk and the Ubuntu 11.04 64bit operating system. For testing purposes, the UI was bypassed to obtain the true times for the segmenting and down-sampling procedures. The FAC was still used when testing, but all variables in the module were kept constant so as to not skew the results and performance of the IPC. A script was used to perform autonomous execution of each experiment consecutively. Three tests were run per file and the results averaged.

3. Software Evaluation

Four files of increasing size were used to evaluate the scalability of the file reading component. In each case, the user interface component was bypassed to obtain the true times for the down-sampling operation. A 1 GB FITS file was chosen as an exemplar and three further test files of 8, 27 and 120 GB were created from this file by scaling its contents.

The time taken to completely render a file increases linearly with file size (Figure 6): the largest 120 GB file required 18 minutes to fully render on an

i7 desktop system with 8 GB RAM. However, the first results were visible in 4 seconds and the system iteratively improved the image. The volume render's performance proved to be good even when the datacubes are very large (for the images shown here it exceeded 60 fps). Performance is only reduced when the resolution of the volume renderer is increased (which is set before downsampling and is constant once the downsampling process is started). Better graphics cards support higher output resolution.

The memory limit for our system is simply the size of the slices in the x and y dimensions. If a single slice does not fit into memory, then the system will fail. There is no effective limit on the z dimension. For the 120 GB file, a single slice requires approximately 380 MB (roughly 10000×10000 pixels). However, newer data sets, such as the upcoming $6144 \times 6144 \times 16384$, $\sim$ 2.5 TB ASKAP cube, are approaching these limits. Extrapolating from the fitted line in Figure 6, the $(y = 8x + 4)$ linear plot suggests 5.5 hours would be required to fully render ASKAP data on our test system. However, the down-sampling and streaming process would present an initial coarse visualisation of the cube much sooner than this.

In terms of qualitative analysis, the interface was evaluated using a client base of three astronomers who performed expert reviews of the software. All three of the astronomers interviewed prioritised rapid visual feedback in the application over total time to complete and memory usage. Additional comments suggested that the most important aspect was the support for large FITS files, as contemporary astronomical software packages for desktop PCs do not support large data cubes. This validates our choice of a file streaming strategy for reading and displaying FITS files.

It is important to note that analysis and visualisation are separate tasks: AstroVis allows for rapid qualitative visualisation, but quantitative visualisation or other data analysis tasks (e.g. obtaining the statistical properties of different regions, finding the global minimum or maximum) cannot be calculated until the entire data set is read. Desktop GPU acceleration of such compute-intensive tasks could be explored in future work. Further, we note that, while the *Astrovis* prototype currently operates on a single machine, its architecture adheres to a client/server model: the File Access and Image Components respectively read and marshal FITS data before transmission to the User Interface Component for user interaction. In future work, this division between server and client can be linked with a network protocol to provide a remote service. It is also possible to reduce the amount of FITS data transmitted through the use of quantisation and compression. As *As-*

trovis approximates data via downsampling, the use of compression would be complementary for the purpose of qualitative visualisation of data.

4. Conclusions

Our prototype system, *AstroVis*, allows researchers in Astronomy to explore 3D visualisations of large data cubes on standard commodity hardware. While previous CPU/GPU cluster based approaches to cube rendering offer realtime visualisation at full visual fidelity, this is accomplished through the use of complex parallel algorithms and high performance computing hardware. Our alternative approach offers the ability to visualise approximations of cubes, which are iteratively refined during visualisation on commodity (desktop) hardware, with resultant good responsiveness and ease of navigation. In addition, the combination of overview and detail views provided by *AstroVis* can facilitate critical global quality control of data cubes. *AstroVis* is open source [1], and can be extended to provide further data analysis and statistics.

5. Acknowledgements

NGC 2403 data cube was obtained from The HI Nearby Galaxy Survey. We are grateful for the financial support of the National Research Foundation (NRF) of South Africa and the Square Kilometre Array (SKA) project, under the HPC Programme for radio astronomy research, NRF Grant No. 78552.

References

Arpaci-Dusseau, R., Arpaci-Dusseau, A., 2013. Hard Disk Drives. Arpaci-Dusseau, Ch. 3.

Bandieramonte, E. S. M., Becciani, U., Costa, A., Krokos, M., Massimino, P., Petta, C., Pistagna, C., Riggi, S., Vitello, F., 2013. VisIVO workflow-oriented science gateway for astrophyscial visualization. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.

[1] `https://bitbucket.org/Siggi_za/astrovis`

Beeson, B., Barnes, D. G., Bourke, P. D., 2003. A distributed data implementation of the perspective shear-warp volume rendering algorithm for visualisation of large astronomical cubes. PASA 20, 300–313.

Cabral, B., Cam, N., Foran, J., 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: VVS '94 Proceedings of the 1994 symposium on Volume visualization.

Fluke, C. J., Barnes, D. G., Hassan, A. H., 2010. Visualisation and analysis challenges for wallaby. In: Proceedings of the Sixth IEEE International Conference on e-Science Workshops.

Gooch, R. E., 1996. Karma: a visualisation test-bed. In: Barnes, G. J. . J. (Ed.), Astronomical Data Analysis Software and Systems V, ASP Conf. Series. Vol. 101. pp. 80–83.

Hassan, A., Fluke, C. J., 2011. Scientific visualization in astronomy: Towards the petascale astronomy era. Publications of the Astronomical Society of Australia 28, 150–170.

Hassan, A. H., Fluke, C. J., Barnes, D. G., 2011. Interactive visualization of the largest radio astronomy cubes. New Astronomy 16, 100–109.

Norris, R. P., 1994. The challenge of astronomical visualisation. In: Crabtree, D. R., Hanisch, R., Barnes, J. (Eds.), Astronomical Data Analysis Software and Systems III, A.S.P. Conference Series. Vol. 61. pp. 51–54.

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., Stobie, E., 2010. Definition of the flexible image transport system (FITS), version 3.0. Astron. Astrophys. 524, A42.

Schneiderman, B., 1996. The eyes have it: A task by data type taxonomy for information visualizations. In: Proc. 1996 IEEE Conf. on Vis. Lang. IEEE Computer Society Press, pp. 336–343.

Tory, M., 2003. Mental registration of 2D and 3D visualizations (an empirical study). In: Proceedings of the 14th IEEE Visualization 2003 (VIS'03). VIS '03. IEEE Computer Society, Washington, DC, USA, pp. 49–.
URL http://dx.doi.org/10.1109/VISUAL.2003.1250396

6. Figure Captions

Figure 1: A (a) $9 \times 9 \times 9$ voxel cube in a FITS file is approximated by a (b) smaller $3 \times 3 \times 3$ in-memory voxel cube. The FITS voxel cube is divided into 27 $3 \times 3 \times 3$ sub-cubes which logically map to each of the voxels of the in-memory cube.

Figure 2: Interleaved reads result in an initial estimate (a) of the FITS voxel cube that is steadily refined until (j) the entire cube has been read in. The display iteratively improves over time, starting from a coarse, inaccurate representation, but rapidly becoming a more accurate representation of the image data. This approach allows the application to be usable almost immediately.

Figure 3: FITS voxel slices are read in multiple passes using an interleaved access pattern. Here, a 9x9x9 FITS voxel cube is subdivided into 27 3x3x3 voxel sub-cubes, which are mapped to a 3x3x3 in-memory voxel cube. Each slice contains partial data from multiple sub-cubes. In the first pass, three interleaved slices are read to provide an initial estimate for the in-memory voxel cubes. Subsequent passes improve this estimate.

Figure 4: A slice of the in-memory cube is updated by three FITS voxel slices. The $9 \times 9 \times 9$ FITS voxel cube is subdivided into 27 $3 \times 3 \times 3$ sub-cubes, mapped to 27 in-memory voxels. Each slice of the FITS voxel cube contains a portion of 9 sub-cubes. The contents of these portions are summed in the accumulation buffer and the number of FITS voxels read for the particular in-memory voxel are stored in the count buffer.

Figure 5: The user interface is mainly occupied by the Overview Display, which contains the rendered 3D cube, and the Detail Display, which displays a slice across the cube. Colour and Opacity controls controlling the RGBA values associated with particular voxel values are located at the bottom of the screen.

Figure 6: Time to process a file to completion for 1, 8, 27 and 120 GB file sizes on an i7 desktop system with 8 GB RAM. The dashed line indicates extrapolation to larger files sizes.