Extending Capabilities of GIMIC Visualisation Pipeline

Abstract

The distribution of neutral hydrogen is a useful tracer of the underlying dark matter across cosmological distances and the key to understanding star formation and galactic growth. In this paper we present extensions to the existing visualisation pipeline that is in place. This paper describes the methods of optimising, porting to multiple platforms, exploring differing visualisation methods and extending the file formats that the program can export.

Introduction

The Australian Square Kilometre Array Pathfinder radio telescope is a new radio telescope due to be completed and fully operational in 2013. The ASKAP telescope will be incredibly sensitive over large areas of sky and has the capability to produce more data in a single week than what is currently stored on the entire World Wide Web. A solution is needed to operate on large amounts of data quickly because this data cannot be stored for an extended period of time.

The GIMIC Visualisation Pipeline is a method of operating on simulation data that the ASKAP telescope might produce. This pipeline is functional, but lacks crucial features if it is to be used in production. In this paper, I present methods of extending the existing visualisation pipeline to be able to export natively to several different file formats, create stereoscopic images and run on many different platforms including High Performance Computing and Cloud Computing solutions.

Drishti Export

One of the tools used to produce images from the volumetric data is a cross platform, open source volumetric rendering package named Drishti. In the previous version of the pipeline, volumetric data was exported in a RAW format. The drawback of this approach is that these files need to be converted to the drishti native file format (.pvl.nc) and this takes considerable time.

The native file format of Drishti consists of two parts, the XML file and the volume file. The XML file has an extension of (.pvl.nc) and contains information about the volume such as minimum and maximum values of the voxel data and the data format. Figure 1 shows an example XML file.

```
<!DOCTYPE Drishti_Header>
<PvlDotNcFileHeader>
  <rawfile>test.raw</rawfile>
  <voxeltype>unsigned char</voxeltype>
  <gridsize>161 108 115</gridsize>
  <voxelunit>micron</voxelunit>
  <voxelsize>1 1 1</voxelsize>
  <description>Information about volume</description>
  <slabsize>115</slabsize>
  <rawmap>703 5000 12242 </rawmap>
  <pvlmap>0 128 255 </pvlmap>
</PvlDotNcFileHeader>
```

Figure 1: A Typical Drishti XML File [1]

The file that contains the actual volumetric data is a similar file format to the RAW files used in the earlier version of the pipeline. The file consists of a single byte that contains voxel type, three integers that contain the dimensions of the volume in the x, y, and z axes. The rest of the data is the actual voxel data. How the voxels are stored depends on the first byte. Figure 2 shows the possible values of the first byte. In the current version of Drishti, the first byte must be 0 and voxels must be stored as single bytes. This means that voxels can have values ranging from 0-255. Figure 3 shows an image rendered with Drishti.

```
0: Unsigned Byte - 1 byte per voxel
1: Unsigned Short - 2 bytes per voxel
2: Unsigned Integer - 4 bytes per voxel
3: Float - 4 bytes per voxel
```

Figure 2: pvl.nc.001 Header Structure [2]

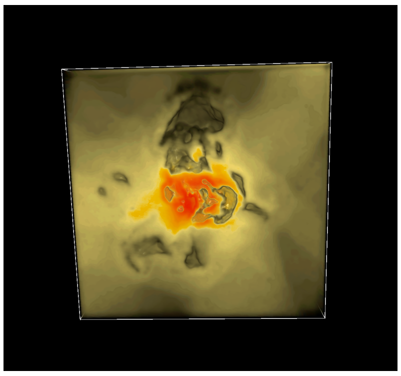

Figure 3: Drishti Rendered Image

The method I used to export to this format was to create the XML file first with the correct fields, then to create the pvl.nc.001 file. When writing data to this file I store the volumetric data with the z axis varying the fastest and the x axis varying the slowest. Figure 4 shows the structure of a typical Drishti volume file.

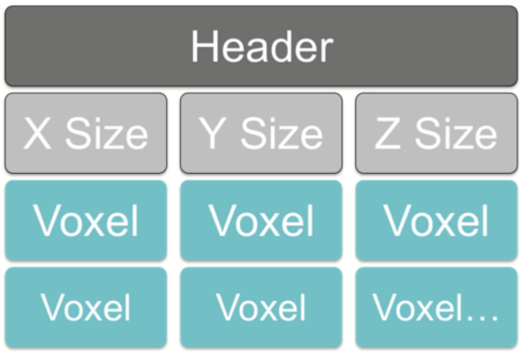

Figure 4: Internal structure of a pvl.nc.001 file

POVray Export

POVray is an a free cross platform raytracing program that supports volumetric textures. The format of a DF3 file is similar to a RAW file. There is a header that is 6 bytes long that contains the dimensions of the volume. E.g. X, Y, Z as short integers. I am writing the voxel data as unsigned short integers and this gives me a range of possible voxel values of 0-65535. Although writing voxel data as unsigned short integers gives me a high dynamic range, if this wasn't sufficient I can output values as integers, floating point or double data types.

When writing to a povray DF3 file, the bit order must be in the form of most significant bits first (big-endian). Because I am running most of my calculations on Itanium2 processors, they will produce integers that are little endian. I used the following macro that swaps the order of the bits and converts little-endian integers to big-endian integers and vice versa.

```
#define SWAP_2(x) ( (((x) & 0xff) << 8) | ((unsigned short)(x) >> 8) )
#define SWAP_4(x) ( ((x) << 24) | \
         (((x) << 8) & 0x00ff0000) | \
         (((x) >> 8) & 0x0000ff00) | \
         ((x) >> 24) )
#define FIX_SHORT(x) (*(unsigned short *)&(x) = SWAP_2(*(unsigned short *)&(x)))
#define FIX_INT(x)   (*(unsigned int *)&(x)   = SWAP_4(*(unsigned int *)&(x)))
```

Figure 5: Swapping Significant Bit Order [3]

A clear advantage of exporting volumetric data natively to povray is that povray can run on supercomputers and has a powerful command line interface whereas Drishti is a graphical program only. Povray supports stereo, fisheye and spherical output as well as multiple cameras and powerful rendering methods. Exporting natively to a povray df3 file gives allows videos and images to be rendered in parallel reducing overall compute time. Figure 6 shows an image that has been rendered in POVRay.

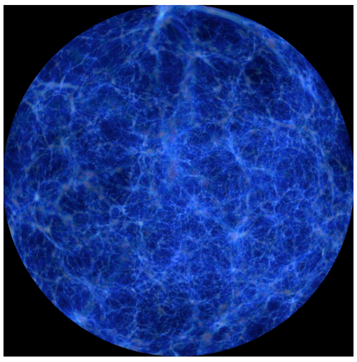

Figure 6: OWLS volume rendered with POVRay.

Tracking

When visualising a time series, it can be difficult to keep track of moving objects within the dataset. It is important that the volume itself does not move. To keep track of the volume, we introduced a method called tracking to keep the centre of mass in the middle of the volume. The old method of scaling the points to fit the volume is shown in figure 7.

```
scale &= MAX(max.x-min.x, max.y-min.y, max.z-min.z)
scale &= (VolumeSize - 1) / scale
```

Figure 7: Flexible method of scaling

The above method works well for single volumes or volumes where the dimensions are unknown. When rendering a time series this method of scaling can introduce distortion effects. The method that I am using scales by bounding box values instead of minimum and maximum point values. An hdf5 file was already created for the GIMIC data set by Rob Crain [4]. This file contained the changing bounding box sizes and offset values for all 949 volumes.

When the volume is loaded, I load two sets of variables from the hdf5 tracking file. These are the bounding box length and the offset values. I then subtract the offset values from each point in the set, recalculating the range afterwards. This puts the centre of mass in the centre of the volume. The next step is to clamp the values shown in figure 8.

```
for (ip=0; ip<npoints; ip++) {
    points[ip].x = Clamp(0 - (BBlength/2), points[ip].x, BBlength/2);
    points[ip].y = Clamp(0 - (BBlength/2), points[ip].y, BBlength/2);
    points[ip].z = Clamp(0 - (BBlength/2), points[ip].z, BBlength/2);
}
```

Figure 8: Clamping points to bounding box length

This has the effect of allowing particles to exit the volume area which is expected behaviour in this simulation. The points are then translated by half the bounding box length. This gives only positive values that can be easily converted to the volume size without problems. The points are then multiplied by a scaling factor that is determined by (nvol-1) divided by the bounding box length. Figure 9 shows a centred volume.

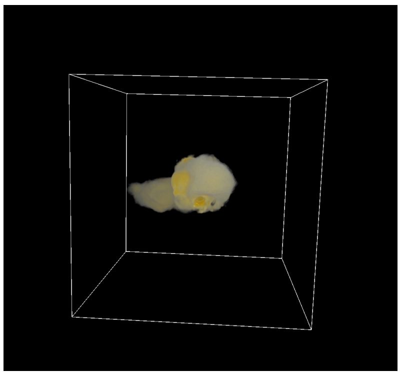

Figure 9: Volume showing centre of mass in centre of volume

Alternate Visualisation Methods

Stereoscopic Images

One of the major problems that the scientific community faces is how to communicate their data to the general public. Stereoscopic imaging is a method of creating 3D images and video that allow the viewer to perceive depth and increase immersion. Stereo images can be used in conjunction with other visualisation methods such as fisheye or spherical projections.

Stereographic images are created from two cameras in a scene that export two images from different perspectives, these images are then combined into a stereo pair which are mixed together on the fly when projected. The stereo pair then becomes what is seen by the left and right eyes.

The method I used to create stereo pairs was to export images from Drishti using the Stereo command line option. I then combined the left and right images using image magick as shown in figure 10. This method is optimised for use with linearly polarized 3D glasses and the two DLP projector approach. This method is more immersive than the typical anaglyph red / blue method. Figure 11 shows a stereo pair.

```
convert left_eye.png right_eye.png  +append -quality 100 'stereo_pair.png'
```

Figure 10: Command line conversion tool

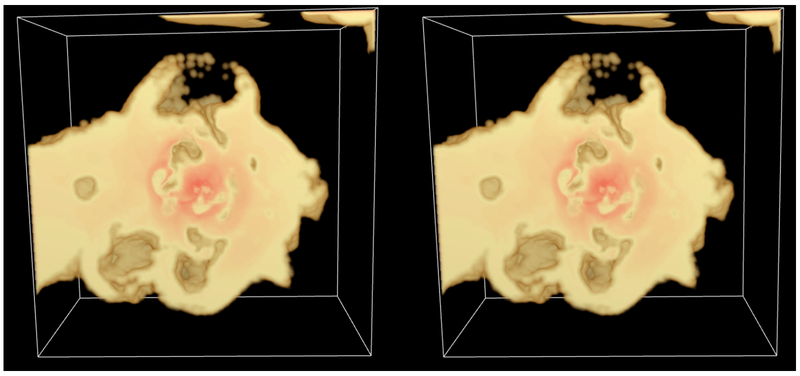

Figure 11: Stereo pair produced with Drishti and Image Magick

R-G-B HI, HII, Htwo Image

An interesting method of visualising more than one dataset at a time is known as RGB volumes. Instead of producing three animations side by side, a well used method is to render the three datasets with different transfer functions. This allows the observation of how the datasets interact and relate to each other.

Drishti supports the loading of more than one volume at a time. I was able to load three volumes, the HI, HII and Htwo datasets. Using different transfer functions (Red for HI, Green for HII and Blue for Htwo) it is possible to observe all three datasets in one animation. This method can be extended to work with animations.

Future work on this subject includes automatically generating a Drishti project file along with the volumes that specifies which volumes to include and their respective transfer functions. This could also be extended to import the volumes, set the correct transfer functions and settings then batch render without interaction from the user. Figure 12 is an RGB rendered image.

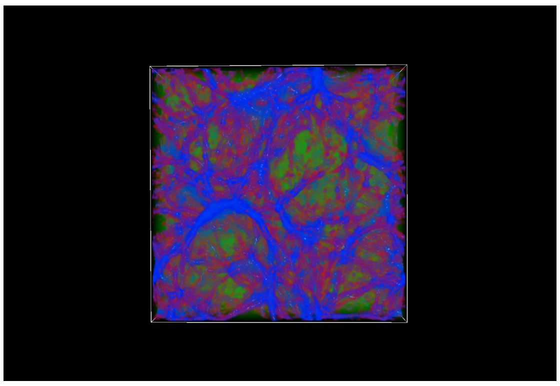

Figure 12: RGB - HI, HII, Htwo Image

Cross Platform Support

The most time consuming part of working on a project that handles large amounts of data is the computation time. Part of this project has been spent optimising code and porting to other platforms to decrease the total computation time and independence from a certain architecture.

The first version of code ran in serial, computing one volume at a time with each volume taking 2 hours to complete. Calculating all volumes for the GIMIC data set this way would have taken about 80 days. The second version of code ran in parallel, computing a single volume per processor. This method takes 20 minutes per volumes and takes 10 hours to complete. The largest bottleneck for the second version of the code was not enough free processing time on supercomputing platforms.

A solution to the problem was to use a distributed computing solution. The third version of code was ported from C to Java. I am using the Java Nereus platform which is an open source Client-Server cloud computing platform [5]. Total computation time using the Nereus platform decreased to 20 minutes. The Nereus client is installed on many machines located at educational facilities. An http server has been set up that contains all the data files and jobs. Each client checks the server periodically and downloads the required files, operates on the data and then uploads the final volumes.

Version	**Single Calculation Time**	**Total Calculation Time**
Serial	2 hours	80 days
Parallel	20 minutes	10 hours
Distributed	9 minutes	20 minutes

Figure 13: Computation Times

Further Work

Further work on this project includes extending the Drishti export capabilities to project files and automatic batch rendering of volumes. The main program code will be ported to run on other platforms such as GPU clusters or other supercomputing platforms like iVec's Xe machine to further reduce platform dependence. Drishti now supports plugins, so a direct hdf5 import plugin could be written for importing single volumes.

There is a production under way at the moment to create content for a presentation at the Horizon Planetarium. This work is ongoing but will be completed soon.

Conclusions

Experimentation has shown that investigating different platforms and technologies can result in a significant speed up in processing time. Through optimisation and parallelisation, total calculation time has dropped from 80 days to less than 20 minutes. Extending the different file formats that the visualisation pipeline can export to has decreased total calculation time and enhanced the functionality as well as making effective visualisation much easier. Using different methods to display data sets can help to visualise and detect problems in code far easier than just running calculations on numbered data sets.

References

[1] A. Limaye, ".pvl.nc file format", Drishti, 22/11/2009, [Online]. Available: http://code.google.com/p/drishti-2/wiki/PvlDotNcFormat [Accessed: 23/02/2010]

[2] P. Bourke, "PVL (PVL (Processed VoLume) File Format, As Used By Drishti", Western Australian Supercomputing Program, November 2004, [Online]. Available: http://local.wasp.uwa.edu.au/~pbourke/dataformats/pvl/ [Accessed: 04/05/2010]

[3] P. Bourke, "POVRay density (DF3) files", Western Australian Supercomputing Program, November 2004, [Online]. Available: http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/df3/ [Accessed: 10/05/2010]

[4] Rob Crain, dh04_950tracking.hdf5 Tracking File. [Software]. Swinburne University: 2010.

[5] Rhys Newman, Nereus V. [Software]. Oxford University: 2010 http://www-nereus.physics.ox.ac.uk/about_overview.html