Visualising Astronomy Data using VRML

Brett Beeson[a], Michael Lancaster[a], David G. Barnes[a], Paul D. Bourke[b] and Guy Rixon[c]

[a]School of Physics, The University of Melbourne, Parkville, VIC 3010, Australia;
[b]Centre for Astrophysics and Supercomputing, Swinburne University of Technology,
Hawthorn, VIC 3122, Australia;
[c]Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge CB3 0HA,
United Kingdom.

ABSTRACT

Visualisation is a powerful tool for understanding the large data sets typical of astronomical surveys and can reveal unsuspected relationships and anomalous regions of parameter space which may be difficult to find programatically. Visualisation is a classic information technology for optimising scientific return. We are developing a number of generic on-line visualisation tools as a component of the Australian Virtual Observatory project. The tools will be deployed within the framework of the International Virtual Observatory Alliance (IVOA), and follow agreed-upon standards to make them accessible by other programs and people. We and our IVOA partners plan to utilise new information technologies (such as grid computing and web services) to advance the scientific return of existing and future instrumentation.

Here we present a new tool − *VOlume* − which visualises point data. Visualisation of astronomical data normally requires the local installation of complex software, the downloading of potentially large datasets, and very often time-consuming and tedious data format conversions. *VOlume* enables the astronomer to visualise data using just a web browser and plug-in. This is achieved using IVOA standards which allow us to pass data between Web Services, Java Servlet Technology and Common Gateway Interface programs. Data from a catalogue server can be streamed in eXtensible Mark-up Language format to a servlet which produces Virtual Reality Modeling Language output. The user selects elements of the catalogue to map to geometry and then visualises the result in a browser plug-in such as *Cortona* or *FreeWRL*.

Other than requiring an input `VOTable` format file, *VOlume* is very general. While its major use will likely be to display and explore astronomical source catalogues, it can easily render other important parameter fields such as the sky and redshift coverage of proposed surveys or the sampling of the visibility plane by a rotation-synthesis interferometer.

Keywords: virtual observatory, visualisation, astronomy, data mining

1. INTRODUCTION

Visualisation is a primary knowledge discovery tool in astronomy. Examples range from ubiquitous two-dimensional scatter plots, such as those that reveal the Tully-Fisher relation[1] and constitute the Hertzsprung-Russell diagram, to volumetric renderings of the Virgo data[2] which reveal filaments, sheets and voids in cosmological simulations. As well as guiding standard data analysis, visualisation can reveal unsuspected relationships and anomalous regions of parameter space which are difficult to determine programmatically and describe analytically. Visualisation can also be helpful in identifying faults in data, caused by such problems as acquisition errors or simulation artefacts.

Astronomy data can be broadly categorised as gridded data, sparse point-like data, or a combination of both. Examples in the first category include the raster data which make up scanned or direct images of the sky, and the volumetric data which constitute the output of several popular magnetohydrodynamic codes (*e.g.*

ZEUS-2D[3]). Sparse data sets include source catalogues of all kinds, broadband spectral energy distributions and the output of N-body simulations (*e.g.* Hydra[4]) to name a few. Many other data sets have the property that some of their axes are gridded, while others are sparsely sampled. For example, complex sky visibilities measured with a radio interferometer having a multi-channel spectrometer (correlator) lie in a space whose photon frequency axis is smoothly gridded, whose time and visibility *axes* are sparsely sampled, and whose polarisation *axis* is discretely sampled.

A new paradigm for visualising astronomy data is emerging in the form of the Virtual Observatory (VO). A common aim of VO projects worldwide is to unify astronomical data description and on-line publication protocols so that users can find and access all manner of data using a very small set of software tools. *Interoperability standards* are being developed by the International Virtual Observatory Alliance (IVOA) partners, and of the earliest standards are the `VOTable`— an eXtensible Mark-up Language (`XML`) format for encapsulating tabular astronomy data — and `conesearch`— a protocol for obtaining astronomical source catalogue extracts primarily based on sky position.

We have previously described our efforts to provide a new perspective on larger-than-memory, multi-dimensional, lattice-based (ie. gridded) datasets.[5] Originally developed with no awareness of the VO paradigm, our distributed-data volume rendering implementation found an interesting role as a demonstrator of remote visualisation capabilities in the context of the VO and the Grid.[6] Motivated by this success, we decided to turn our attention to sparse, point-like data – such as that returned in a `VOTable` by a `conesearch` request – and how it might be displayed and explored in the VO context.

In this paper, we present a new visualisation tool for virtual observatories – *VOlume*. In Sect. 2, we describe the context of the tool and define the requirements it is based on. In Sect. 3 we outline the design of *VOlume*, the implied technology choices and brief implementation details. Some examples are given in Sect. 4. Rendering performance characteristics are examined in Sect. 5, and we conclude with some remarks on possible extensions to *VOlume* in Sect. 6.

2. VISUALISATION OF VIRTUAL OBSERVATORY DATA

2.1. The VO visualisation paradigm

The main plotting tools available for visualisation of data in `VOTable` format are *VOPlot*[*], *TopCat*[†] and *Mirage*.[7] The first two are products of the astronomical community, and provide simple two-dimensional (2D) plotting capabilities and limited statistical operations. *Mirage* is a more general data exploration tool for scientific data, providing linked 2D plots of tabular data and some reasonably sophisticated statistical analysis procedures. None of these tools produce publication-quality output and they all struggle with even moderate-sized tables. Instead, they satisfy *quick look* and interactive visualisation requirements.

The *quick look* approach enables the user to immediately examine data without explicitly downloading and converting it to a format suitable for a legacy plotting package. For example, prior to requesting a large extract of a catalogue, the user may request a smaller extract (*e.g.* a bright sub-sample of the large extract) and make a plot to verify the sky coverage of the sample. Similarly, in the exploratory phase of a new science project, an astronomer may want to visually verify the overlap in sky and redshift coverage of two or more catalogues made at different wavelengths.

Quick look tools must be convenient. In the VO context this basically means they must support the standard data interchange formats (ie. `VOTable`), they must be able to transparently retrieve remote data files via the standard transport protocols (`HTTP` now; `GSIFTP` and others in the future), and they must be integrated with the VO environment in which the user finds and requests data (ie. operable from a web browser).

Motivated by the data-mining aspects of the VO, the new visualisation tools are also expressing more interactive features. *TopCat* enables the user to propagate a selection from one 2D plot (or from a spreadsheet-style view of the source data) to another 2D plot, allowing rudimentary by-eye cluster analysis for example.

Mirage takes this further and supports highlighting of a data selection in many 2D plots. It is interesting that capabilities like this have rarely been exposed to users even in the largest legacy astronomy data reduction packages.

2.2. From two-dimensional to three-dimensional visualisation

The ubiquity of 2D plots (whether scatter diagrams or images) as visualisation tools in science is probably due to three reasons:

- the publishing medium (paper) is a (static) 2D format,

- the display device (monitor) is a 2D format, and

- the computational demands — in terms of both programmer skill, and processing power to provide interactivity — are small compared with three-dimensional (3D) plots.

It is widely acknowledged that the computational capabilities of desktop workstations have essentially followed Moore's Law (Electronics, April 1965), doubling every 18 months. Less obviously, and certainly less exploited in the scientific domain, the three-dimensional graphics capabilities of workstations have grown even faster, driven primarily by the entertainment industry. The combined growth in computational and graphical power means that today's typical workstation can easily compute and render three-dimensional environments at a rate of order 30 million triangles per second. At these rates, virtual movement through a 3D space can provide a genuine spatial comprehension of a dataset, even on a 2D display device.

2.3. VOlume

The archetypal *use case* which motivates our development of *VOlume* is that a user has located and/or generated a `VOTable` which they wish to explore visually. The `VOTable` is available via a generic data stream, allowing the input to come from an application, a file or a network socket. The streamed `VOTable` is parsed by *VOlume* and the user is presented with a simple interface to choose which columns of the table to visualise, and how to map values in the table to geometry and colour. Sensible default values should be provided, such as mapping right ascension to θ and declination to ϕ if a spherical coordinate system is chosen.

Once the user is satisfied with these mappings a visual representation is produced by *VOlume*. Each row in the input `VOTable` corresponds to a discrete point in the visualisation which has been scaled, coloured, textured or oriented to convey more information such as integrated flux or apparent diameter. The user should be able to interactively rotate the visualisation to see obscured features and to get a sense of the structure. The user should also be able to zoom to inspect more distant or finer-scale structure, and to move into and through the data volume. To aid navigation, visual decorations might be added such as coordinate axes and planes.

3. DESIGN AND IMPLEMENTATION

3.1. Browser and plug-in paradigm

We envisage our system being used primarily as a quick sanity check or for rapid inspection of data. It is unreasonable to require the user to download, compile and install software locally. Therefore our primary design requirement is that the user should be able to use any reasonably modern web browser, equipped with an appropiate third-party plug-in (which is installed once), to visualise their data. The speed of the server-side processing of `VOTables` and the graphics performance of their workstation must be adequate to transform and view medium-sized `VOTables` of say 10,000 sources. High performance is not required since in practice network bandwidth will limit the practical transferrable `VOTable` size. A suitable web browser plug-in ought to be available for Microsoft Windows, Apple Macintosh OS X, Linux and perhaps Sun Solaris.

We chose to use Virtual Reality Modelling Language (`VRML`) to describe the visual model of a `VOTable`. `VRML` is widely used and supported, and nearly all common hardware and software platforms have freely available web-browser plugins of varying standards. As well as defining a syntax for describing 3D environments, `VRML`

defines a standard method of moving within and controlling the view of the virtual environment. By choosing an existing standard with wide support we can focus on the production of content rather the rendering and manipulation of the content. The main alternative would be to write and supply custom visualisation code, which the user would have to download, possibly compile, and install. While this would undoubtedly produce superior results in most respects (flexibility, transformation and rendering speed, *etc.*), it would be a substantial additional effort over and above the `VRML` approach in terms of developing and supporting a multi-platform 3D content viewer and scene description language. Although we have not pursued this route, we give some indication of the possible performance gains in Sect. 5.

3.2. Web-application and Java Servlet paradigm

We provide the conversion from `VOTable` to `VRML` using a web-application. This satisfies the requirement that the user need never explicitly download a remote `VOTable` to their local system. The `VOTable` is specified via a URL, which allows any resource accessible via URL, **including other web-applications**, to act as `VOTable` sources. This model makes it possible to allow a `VOTable` to be supplied via an `HTTP POST` request (see Sect. 6). The URL may be specified via a parameter in the web-application address, as well as within the user interface, *e.g.*`http://services.aus-vo.org/volume?SourceXMLURL=http://any.address.org/votable.xml` This follows the model of VO protocols such as `conesearch` and `Simple Image Access Protocol`. and allows *VOlume* to *interoperate* with other VO-compliant web-applications, such as *SkyCat*[‡].

Our choice of Java Servlet Technology (hereafter "servlets") restricts our user-interface to HyperText Markup Language (`HTML`) forms. We choose *not* to use applets since users often experience problems enabling Java in their browsers. A wizard-style interface guides the user through the selections of columns, geometry mappings, *etc.*. A wizard is a very good solution to the common problem of complex user interfaces in `HTML`. Often the user interface depends on previous user input. For example, if a user selects spherical coordinates then r, θ, ϕ mappings must be displayed, but for cartesian co-ordinates, x, y, z mappings are needed. Standard `HTML` cannot change dynamically to directly accommodate this. Wizards present manageable portions of the interface to the user, one step at a time. After each step the input can be validated and verified, and the next set of options presented to the user. We can create flexible, dynamic interfaces in `HTML` using this method.

3.3. Transformation paradigm

To transform `VOTable` to `VRML` we initially looked at Extensible Stylesheet Language Transformations (XSLT). Using XLST seemed to promise less coding and faster development. The stylesheet we produced could easily have been used by other VO services and applications in different projects. However we need to transform not just the *style* of the document, but also its *content*. This entails amongst other things handling geometric coordinate conversion, and XSLT's very basic arithmetic functions quickly make this solution unworkable. Instead we coded our own transformer. Even here, though, a number of tools are available to help speed development. There are several `VOTable` parsers, as well as numerous generic `XML` parsers.

3.4. Streaming paradigm

The parsing and transformation of the `VOTable` is done on a data-stream, in preference to loading the table into memory. Despite this approach introducing some technical difficulties, it allows the server to process extremely large datasets simultaneously, providing a much more scalable solution than the in-memory paradigm. `VOTables` having sizes of order 100 MB can be feasibly processed in this manner. The performance of a web-server would degrade seriously trying to parse just one, let alone several, such tables in memory.

Although parsing streaming content provides a much more scalable solution, it creates problems because the parsing code cannot anticipate potential issues downstream. If, for instance, there is a syntax error in the `VOTable`, (*e.g.* the `XML` is not well formed), a streaming parser can only interrupt and write an error to the output stream. It is very difficult to generate sane `VRML` if this occurs. Another issue lies in the design of the `VRML` generating code. It is much harder to generate sensible `VRML` if the input must be parsed in sequential order. We built a custom streaming `VRML` java class library to do this, as we could not find a suitable pre-existing

---

[‡]`http://services.aus-vo.org/skycat`

product. This library enables the programmer to easily generate sane `VRML` without having to worry too much about the `VRML` syntax.

The two other issues that we faced due to the need to stream the content were finding a suitable streaming `VOTable` parser, and the need to read the data stream multiple times. The streaming `VOTable` parser that we originally looked at was *SAVOT*. However, at the time, *SAVOT* operated by extracting an entire `TABLE` element from the stream at a time and since a single element may be very large, *SAVOT* was not a satisfactory solution. Consequently, we built our own rudimentary `VOTable` streaming parser for *VOlume*, based on the *kXml*[§] parser. We note that the latest version of *SAVOT* (2.1) now supports row-by-row parsing, and indeed and so does the *STIL*[¶] parser. A future version of *VOlume* could be modified and simplified to use one of these third-party, event-based parsers.

There are a number of reasons why we do multiple passes of the data stream. We do a first pass to get the table meta-data and a row count, used in presenting options to the user. Actually this is done in two passes due to a subtle design issue, but could be changed to a single pass. The data stream is then parsed again to apply the selected transformation. If the user chooses to display the data as points, we need to parse the data a third time in order to extract the colour information. This is due to the way sets of points are described in `VRML`. Again, the colour information could be extracted in a single pass, but the resulting `VRML` would be more complex.

3.5. Language paradigm and implementation issues

Java is the language of choice for the *VOlume* application. It is ideally suited to network applications where performance is not critical (the reader is reminded that *VOlume* is only charged with *transforming* `VOTable` streams to `VRML` streams, and *rendering* performance is relegated to the browser and plug-in combination chosen by the end-user). Java provides excellent `XML` support and supports rapid development. Our web application needs to be *stateful* – that is, it needs to remember the user's previous inputs – and therefore session (or cookie) support is needed. Java servlets are a good choice since they provide all these features, as well as `SOAP` support if we want to create a web service (see Sect. 6). Finally, Apache Tomcat[‖] provides a free, robust and well-tested web application (servlet) container.

The implementation is relatively straightforward and consists of a Java package to read a `VOTable` stream and write a `VRML` stream. Around this is a Java 2 Platform, Enterprise Edition (J2EE) `HTTP` servlet to provide the user interface and `HTTP` response streaming. Construction of the user interface was just as time-consuming as the conversion of `VOTable` to `VRML`. In subsequent development, we would use one of the web-application frameworks (such as Struts, or plain Java Servlet Page [JSP] tag libraries) to streamline the user interface development.

The main issue during implementation was the use of streams rather than in-memory manipulation of data. This will arise more and more in VO applications as they move from prototypes to production versions which need to handle very large datasets. Error handling is tricky in this paradigm: if we encounter an error halfway through outputting `VRML` we must ignore it and continue. We cannot return to an `HTML` page since the response type has already been set to `VRML` and part of the document has been sent. It may be possible to describe the error via text in the `VRML` virtual world. This would increase the code complexity and is not reliable since at least one popular `VRML` plugin doesn't display text! An inelegant and inefficient alternative would be to to transform and produce output twice: once to validate the transformation, and once to stream the validation to the consumer.

4. EXAMPLES

In this Section, we provide some simple examples of how *VOlume* can be used to explore data. In the first example, we also include a workflow diagram and figures depicting the user interface used to control the

---

[§]`http://www.kxml.org`

[¶]`http://www.star.bristol.ac.uk/~mbt/stil`

[‖]`http://jakarta.apache.org/tomcat`

transformation from `VOTable` to `VRML` format. By far the best way to become familiar with *VOlume* and what it can do is to use it! Readers are encouraged to install a `VRML` plug-in viewer for the browser, visit `http://services.aus-vo.org/volume`, and transform and view one of the provided `VOTables` or one of their own.

4.1. The HIPASS High Velocity Clouds

First, we show a worked example of setting up the *VOlume* transformation parameters and the resultant visualisation. For this example, we use a `VOTable` containing the entire High Velocity Cloud catalogue (Putman et al., 2002) generated by the Aus-VO *SkyCat* service. Figure 1a shows the web form presented to the user, once they have provided their `VOTable`; in this case, the `VOTable` was provided to *VOlume* by *SkyCat*. A brief summary of the columns of the `VOTable` is given and the user can select the geometry of the transformation (rectangular [default], spherical or cylindrical), the type of object to display (points [default], boxes, spheres or tetrahedra), and whether to show geometric decorations.

Figure 1b shows an excerpt of the next form presented to the user. Here, their existing choices are summarised, and they can now select which columns (*fields*) in the `VOTable` should be mapped to the three coordinate system axes (in this case r, θ, and ϕ), and which column is to be used to colour the objects. In this example, the user has chosen to map the fields `V_LSR` to r, `RA` to θ, `DE` to ϕ, and `FLUX` to colour.

The final mapping parameters the user can modify are shown in Fig. 1c. Again, their existing choices are summarised, and their task is now to select input and output ranges for the (linear) transformation mappings, and choose an appropriate colourmap. The ranges are initially populated to show the full range of the data, but the user can modify these to select a subset of the data, or pin the end-points of the transformation mappings. In this case, the user has set the output radius range to $[1.0, 1.0]$, which will result in all points being mapped to the surface of the unit sphere, yielding a *sky view* type visualisation. `VRML` views of the resulting environment are shown in Fig. 2.

4.2. The 2dF Galaxy Redshift Survey

One of the main uses of *VOlume* will likely be to explore structure in redshift catalogues. Figures 3 and 4 show *VOlume* representations of a brightness-limited sub-sample of the 2dF Galaxy Redshift Survey 100k data release.[8] In these figures, the wedge-shaped nature of the survey is clearly visible, but so too is structure within the wedges of the survey. This example was simply constructed by obtaining the sub-sample in a `VOTable` from the CDS *Vizier* service, storing it temporarily on a web server, and giving the URL of the `VOTable` to the top-level *VOlume* servlet page.

5. TRANSFORMATION AND RENDERING PERFORMANCE

5.1. Conversion and bandwidth considerations

The time taken for *VOlume* to convert an arbitrary `VOTable` to `VRML` format is generally very small, and is ordinarily dominated by the network bandwidth between the *VOlume* service and the source `VOTable` and between the *VOlume* service and the client workstation. In the absence of a bandwidth bottleneck (ie. for local conversion), the parsing speed is an entirely respectable $\sim 10^4$ rows per second. Input network bandwidth problems can be alleviated to a large extent by preparing the source `VOTable` with only the fields (columns) that are going to be mapped to elements of the geometry. Output bandwidth needs are generally lower because `VRML` files are typically smaller than the input `VOTable` and can be efficiently compressed at the service end and automatically uncompressed by most well-configured browsers. `VRML` viewers themselves readily accept compressed `VRML` even if the host browser fails to decompress the incoming stream.

Figure 1. Transformation servlet forms: **a.** — selecting geometry, object type and decorations; **b.** — setting column mappings; **c.** — selecting linear transformation ranges, clipping and colourmap.
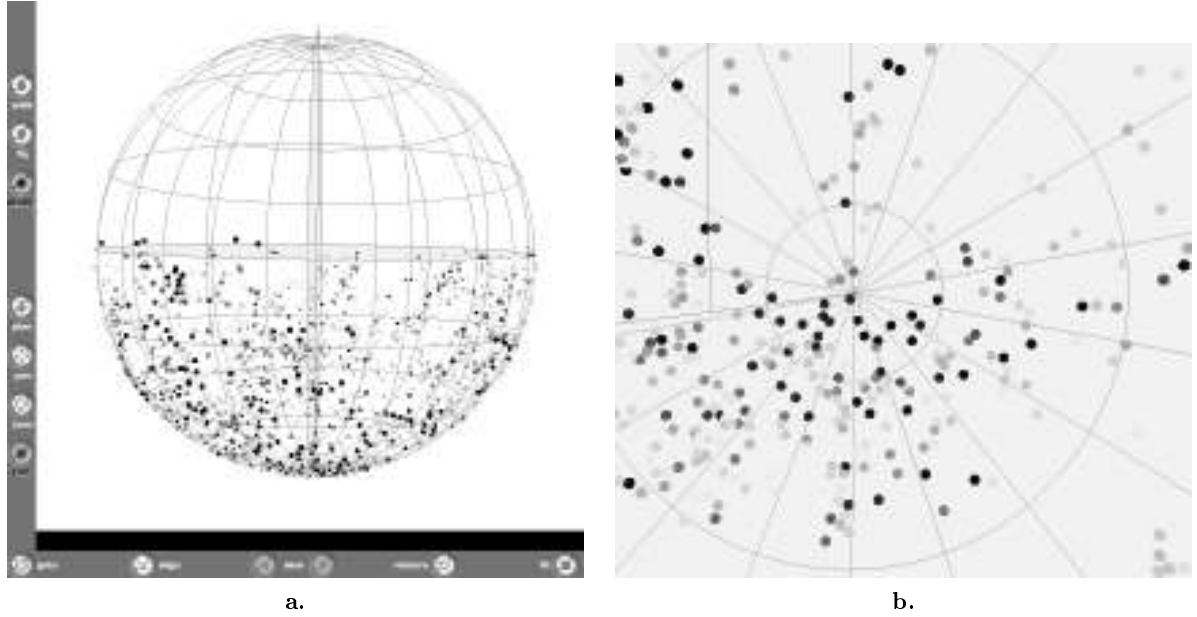
Figure 2. VRML view of the HIPASS High Velocity Clouds: **a.** — external view showing entire southern HVC population, and *Cortona* user interface; **b.** — internal view towards the south celestial pole.

5.2. Rendering speed

The conversion from `VOTable` format to `VRML` is fast in all but the most pathological cases, and in any case, is done at most a few times per session. and so the usefulness of *VOlume* as an exploratory tool depends almost entirely on the rendering of the constructed environment being accomplished at interactive speeds. This in turn relies wholly on the third-party `VRML` viewers that are available to render *VOlume*-created environments. It is therefore pertinent to briefly assess the performance characteristics of `VRML` browsers compared to a native, C-based OpenGL[**] renderer.

For our tests, we selected the brightest catalogue entries in the 2dF Galaxy Redshift Survey data.[8] Using *VOlume* we generated `VRML` files with right ascension mapped to spherical coordinate θ, declination mapped to ϕ and $z_{\odot}$ mapped to r. We created two files, one – **points** – with 9600 sources each represented by a point object coloured by to B_{J} magnitude, and another – **spheres** – with 1000 sources each represented by a sphere, also coloured by magnitude. We displayed the generated `VRML` files using the Open Source `VRML` renderer, *FreeWRL*[††], and measured the rendering frame rate. We would have preferred to use the *Cortona*[‡‡] renderer for its refined interface but we could not measure frame rates in any sensible way with *Cortona*. For comparison with a fast, native renderer, we also transformed the sample into suitable input for the *Stereo2* program, using the same coordinate and colour mappings. *Stereo2* is a highly-optimised, C-based virtual environment renderer which can render points and spheres using native OpenGL calls. Frame rates were measured on an Apple PowerBook with hardware OpenGL acceleration, at a screen size of 1024×768 pixels, and are shown in Table 1. We went to some length to ensure that spheres were rendered with the same number of polygons in *FreeWRL* and *Stereo2*.

Both *FreeWRL* and *Stereo2* offer excellent frame rates for environments containing points and lower frame rates for (smaller) populations of spheres. For interactive work, frame rates of ~ 10 fps or more are desirable. The advantage of the native OpenGL implementation is clear, giving frame rates better by a factor of at
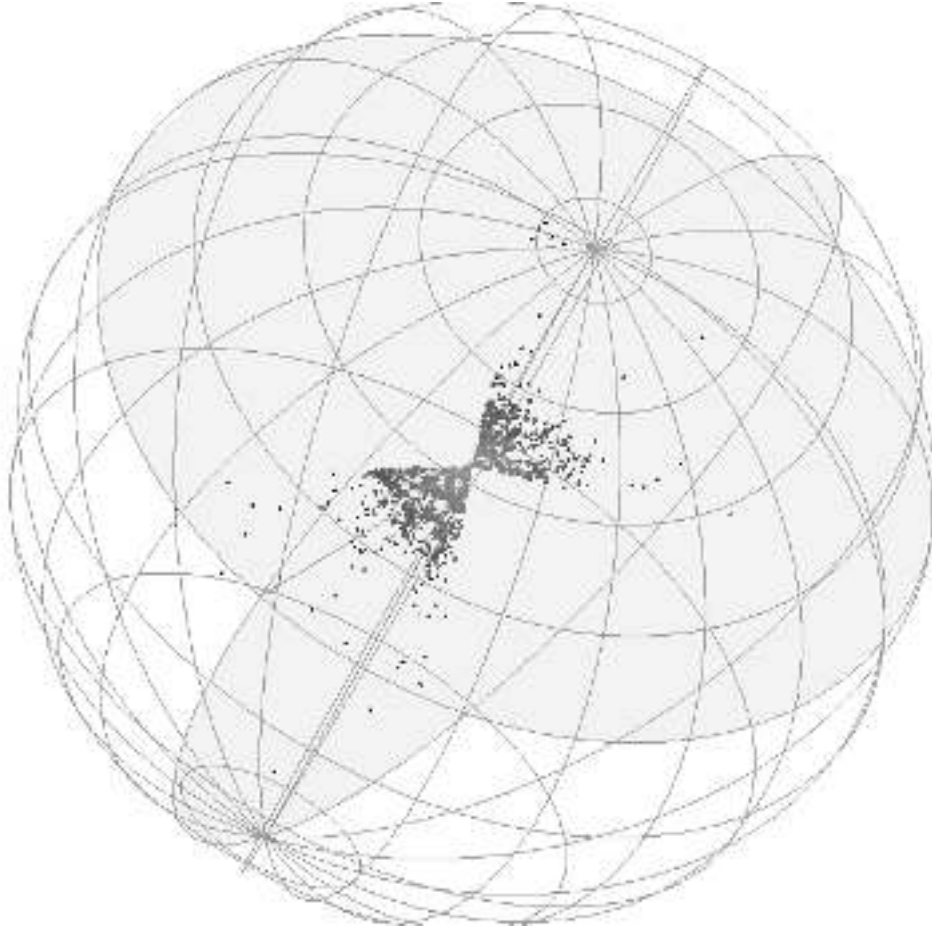

[**]`http://www.opengl.org`
[††]`http://freewrl.sourceforge.net`
[‡‡]`http://www.parallelgraphics.com/products/cortona`

Figure 3. `VRML` view of the 2dF data release: external view of spherical projection, with tetrahedra coloured by redshift.

Table 1. Rendering speeds (in frames per second [fps]) of *FreeWRL* and *Stereo2* for **points** and **spheres** geometries.

(fps)	**9600 points**	**1000 spheres**
FreeWRL	27 ± 2	1.7 ± 0.3
Stereo2	> 100	15 ± 1

least ~ 4 for point geometry, and nearly 10 for spheres. This is somewhat surprising, because in our tests, *FreeWRL* was used in "stand-alone" mode (*i.e.*,, not as a browser plug-in), and there ought not to be such a large difference: both *FreeWRL* and *Stereo2* are using direct hardware OpenGL acceleration, and once the scene is parsed and stored in an OpenGL display list, performance should be very similar. We are tempted to conclude that *FreeWRL* is either not using OpenGL display lists at all, or using them inefficiently. Subjective tests comparing *Cortona* to *FreeWRL* gave the impression that *Cortona* had a clear edge over *FreeWRL* for spheres, but still performed well below the level of *Stereo2*.

A point should be made regarding spheres and how they are drawn. All OpenGL geometry must eventually be drawn as a set of quadrilaterals or triangles. Spheres can be tesselated or tiled in many ways, and at different resolutions (scales). The `VRML` viewers we have used seem to provide no way to control (or even know!) the tesselation level, whereas *Stereo2* provides this option and it can be used to very good effect. However, even
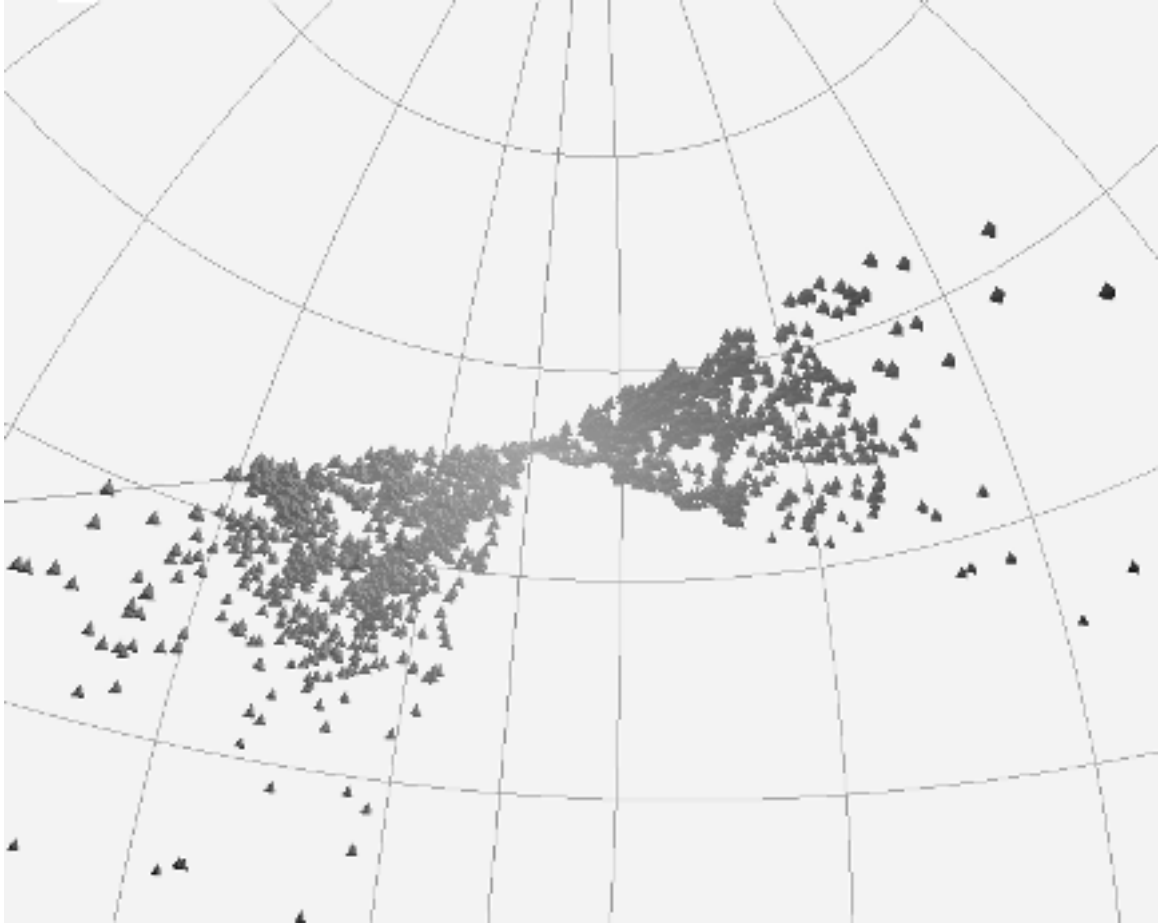
Figure 4. `VRML` view of the 2dF data release: view from inside a spherical projection, with tetrahedra coloured by redshift. Tetrahedra are rendered faster than spheres, yet still provide distance cues via obscuration and scaled size.

Stereo2 lacks some relatively simple but probably highly effective optimisations, such as creating one sphere at the desired resolution, storing it in a display list, and then translating and scaling it into place for each instance.

The catch is that spheres, or at least solid objects, are far more useful than simple points in interactive 3D work. Solid objects that have volume are drawn smaller when further away, and nearby objects obscure distant objects. In lieu of control over the level of tesselation of spheres by the various `VRML` viewers, we elected to provide the coarsest-possible control in *VOlume*: the user can choose to use tetrahedra instead of spheres. This yields solid objects with only four facets, and produces effective and more rapidly rendered `VRML` environments (see Fig. 4).

Aside from sphere tesselation, and for that matter, any differences in geometry sent to the OpenGL hardware, the `VRML` viewers have additional features above and beyond *Stereo2* which may also be slowing their performance. For example, whether or not a `VRML` viewer is running stand-alone or as a plug-in, its event loop may be quite different to that in *Stereo2*. It may have to calculate and correct for collisions between the virtual camera and objects in the 3D environment, or it may need to relinquish control to the user interface and/or the host browser for some appreciable fraction of time. It may also be the case that `VRML` players must use some non-optimal OpenGL in order to support older hardware, something that *Stereo2* does not do.

6. EXTENSIONS

There are several possible extensions to *VOlume* which would increase its utility, performance and flexibility. We highlight a few of them here.

Simple extensions. *VOlume* does not yet ascribe any attribute to the elements of the VRML world other than colour. This could easily be extended so that additional columns in the input data stream could be mapped to size, orientation or even dilation along an axis, thereby preserving more of the information in the input VOTable. Text could also be used to label key objects or regions in the VRML, or to provide simple axis labels and scale information. Indeed, a colour cylinder could be embedded in the scene to convey the colour mapping information. VRML also provides for pre-defined viewpoints, animation of elements of the scene, and the incorporation of Java-based components. Support for these latter extensions is often incomplete or unreliable in VRML viewers though, so we have not explored them at this stage.

X3D. We currently use VRML97 which is a text file format. A new VRML format, called X3D, is currently a draft standard and already has prototype browsers. X3D will provide XML and binary formats in addition to the current text format. The XML format opens the possibility of using existing XML tools to write the scene, instead of using custom code, and the ability to validate the output against the X3D schema as it is written.

Optimisations. Our performance tests show that sphere elements are slow to render compared to points. We plan on exploring the 3D gaming technique of *billboarding* to reduce the polygon count while still providing a volumetric effect to each point. A texture is applied to a billboard (a single polygon) which is rotated to always face the viewer, giving the impression of a 3D object. Billboards have the advantage of (optionally) obscuring more distant billboards (textures can be made transparent so this does not happen), as well as being scaled according to distance from the camera. Textures representing flat and specularly shaded spheres are simple to generate, and can be given arbitrary colours at render time. Another approach to improving sphere rendering speed would be to tesselate the spheres within the *VOlume* code and write the resultant polygons to the VRML stream; a tesselation (scale) control could be provided in the user interface so the user could choose how finely the spheres are rendered.

Web services. Instead of a servlet, a web service could use *VOlume* code to provide other programs with the ability to produce VRML visualisations. There are a number of transformation parameters which are presently set interactively via the user interface. The web service would receive an XML document describing the geometry mappings and return an X3D XML document. Using a web service we could, for example, send any fixed-width text catalogue to a service which converts it to a VOTable then streams it directly to *VOlume* for conversion to VRML. The user would never see the VOTable nor VRML, but rather just their text file displayed visually.

Stereoscopic display. The perception of 3D environments on flat 2D screens can be improved dramatically by using stereoscopic display techniques. On single monitors, stereoscopy is frequently implemented in a *frame-sequential* mode — subsequent vertical refreshes of the monitor render the scene for the left eye, then the right eye, then the left again, and so on. Synchronised LCD shutter glasses alternately block the view of each eye, leading to a stereoscopic perception of the scene. For projection, passive stereoscopy is feasible using dual projectors and oppositely polarised filters over the projector lenses and the viewers' eyes. There are essentially no useable and free stereoscopic VRML viewers available now. However, it shouldn't be a particularly difficult project to create a stereoscopic version of *FreeWRL*. We leave this as an exercise for the reader.

ACKNOWLEDGMENTS

REFERENCES

1. R. B. Tully and J. R. Fisher, "A new method of determining distances to galaxies," *Astron. & Astroph.* **54**, pp. 661–673, 1977.

2. A. Jenkins, C. S. Frenk, F. R. Pearce, *et al.*, "Evolution of structure in cold dark matter universes," *Astroph. J.* **499**, pp. 20–40, 1998.

3. J. M. Stone and M. L. Norman, "Zeus-2d: A radiation magnetohydrodynamics code for astrophysical flows in two space dimensions. i - the hydrodynamic algorithms and tests," *Astroph. J. Supp.* **80**, pp. 753–790, 1992.

4. H. M. P. Couchman, P. A. Thomas, and F. R. Pearce, "Hydra: an adaptive-mesh implementation of p 3m-sph," *Astroph. J.* **452**, pp. 797–813, 1995.

5. B. Beeson, D. G. Barnes, and P. D. Bourke, "A distributed-data implementation of the perspective shear-warp volume rendering algorithm for visualisation of large astronomical cubes," *Publ. Astron. Soc. Aust.* **20**, pp. 300–313, 2003.

6. G. Rixon, D. G. Barnes, B. Beeson, J. Yu, and P. Ortiz, "Visualizing data cubes on the grid," in *Astronomical Data Analysis Software and Systems XIII*, F. Ochsenbein, M. G. Allen, and D. Egret, eds., *Astron. Soc. Pacific Conf. Series.*

7. T. K. Ho, "Mirage: A tool for interactive pattern recognition from multimedia data," in *Astronomical Data Analysis Software and Systems XII*, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds., *Astron. Soc. Pacific Conf. Series* **295**, pp. 339–342, 2003.

8. M. Colless, G. Dalton, S. Maddox, *et al.*, "The 2df galaxy redshift survey: spectra and redshifts," *Mon. Not. Royal Astron. Soc.* **328**, pp. 1039–1063, 2001.