Publ. Astron. Soc. Aust., 1997, 14, 106–10

Grasping the Wispy Tendrils

Richard E. Gooch

Australia Telescope National Facility, CSIRO,
PO Box 76, Epping, NSW 2121, Australia.
Macquarie University, Nth Ryde, NSW 2109, Australia
rgooch@atnf.csiro.au

Received 1996 August 26, accepted 1996 December 16

Abstract: The vast quantities of data produced by modern radio telescopes require advanced visualisation tools to explore them. The quantitative nature of astronomy requires more than just representing data visually. This paper discusses research in displaying, and using, a three-dimensional 'cursor' in amorphous, noisy data which often resemble wispy tendrils.

Keywords: techniques: spectroscopic — methods: miscellaneous — methods: numerical

1 Introduction

Previous visualisation research work at the Australia Telescope National Facility has concentrated on visual representation of large three-dimensional data sets to allow the astronomer to gain insights into the global structure of the data [see Norris (1994) and Gooch (1995*a*,*b*,*c*) for more information]. The next step in the process of extracting science from data involves obtaining quantitative information.

Something the astronomer would often like to do is point at a feature of interest in the three-dimensional data cube, usually some small 'blob' or 'lump', and find out what the three-dimensional coordinate of the blob is. The astronomer would also like to draw a three-dimensional polygon around a blob and do some further analysis on that sub-volume, such as computing the total flux.

Because of the nature of astronomical data, no sharply defined edges and surfaces appear solid. As a consequence, the impressive illusion of depth we are familiar with in other fields of visualisation (which often use surface-rendering techniques) tends not to be so dramatic when visualising astronomical data. We are left then with the problem of how to grasp the wispy tendrils of space without stabbing into the void.

To do this we must be able to determine the three-dimensional position of features of interest and display three-dimensional objects inside a volume of data. This paper discusses new research towards this goal.

1·1 Cursors

To determine the three-dimensional position of a feature the user sees, there must be some mechanism to point to the feature. This is analogous to the two-dimensional cursor (usually controlled by a 'mouse') available in many image-display tools. Simply moving the cursor to a feature of interest and reading off the horizontal and vertical coordinates is not much help, because there is no information about depth. Is the feature close to the front or the back of the volume?

There are two main problems with implementing a three-dimensional cursor: how to input the position and how to display the cursor. This paper will focus on the latter problem.

1·2 Practicality

Astronomers generally consider themselves to be chronically under-funded, and generally do not have access to the latest computational resources. Many of the obvious 'brute-force' techniques are not an option and hence algorithms to reduce computational effort are required. Some techniques in this area are also discussed.

2 Two-dimensional Active Cursor

A simple but quite useful cursor for three-dimensional position determination is a modification of the familiar two-dimensional cursor.[1] Here the user moves the conventional cursor over the projected volume-rendered data to a feature of interest. The position of the two-dimensional cursor is then projected back into the volume and an estimate is made of the depth down the ray of the feature the user sees. The algorithm used to compute the estimate changes, depending on the shading algorithm used to render the volume.

For a simple 'maximum voxel' algorithm, where the maximum voxel value along each projected ray is displayed, the depth of the feature is approximated

[1] This technique is based on an idea by Tom Oosterloo of the Australia Telescope National Facility.

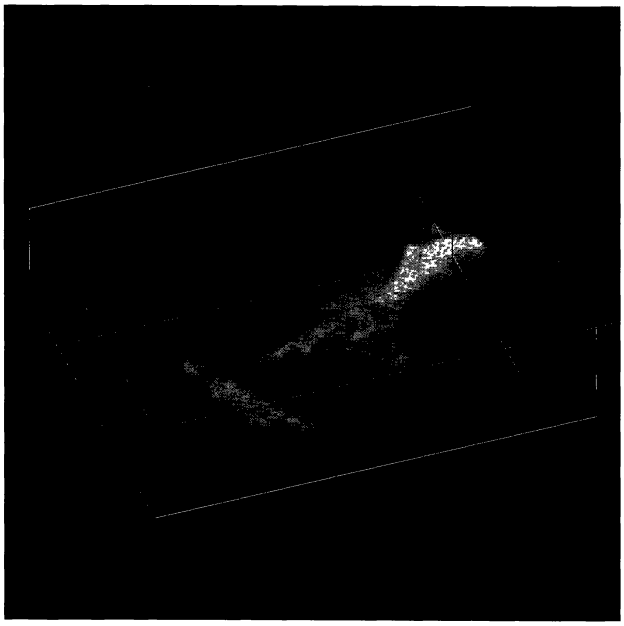

Figure 1—NGC 4631 with three-dimensional overlay cursor.

to the depth of the voxel at which the intensity is a maximum.

For more complex 'hot gas' shaders (described in Gooch 1995*c*), which use a radiative transfer algorithm, the depth is computed using the following:

$$depth = \sum \sigma_i d_i \Big/ \sum \sigma_i \,, \qquad (1)$$

where *depth* is the estimated depth, σ_i is the opacity of voxel i, and d_i is the depth of the voxel. This gives meaningful results for compact, nearly opaque features and for extended, less opaque features. It does not perform so well where there are two compact, nearly opaque features, one in front of the other. I have added a threshold function which is effective if the nearer feature has sufficient cumulative opacity. The threshold function ignores data that lie behind quite opaque features.

Once the approximate depth of the feature is computed, a simple geometric transformation may be applied to convert this and the two-dimensional screen coordinate to a three-dimensional coordinate. This 'active cursor' allows the user simply to point to a feature and obtain the full position information.

As indicated above, the estimate of the depth is not ideal, so to remove ambiguity a further extension has been made which interactively displays the voxel data and opacity data (where appropriate) along the projected ray. The estimated depth is also shown on this display, which allows the user to determine whether the estimate is reasonable.

Another limitation of this cursor is that it cannot work with a stereoscopic display. This cursor requires that every point on the screen be projected from a unique ray through the volume. Clearly this is not the case with stereo, where each point on the screen is projected from two unique rays through the volume. Hence, the user cannot use stereo to assist in perceiving depth if using the active cursor.

3 Three-dimensional Cursor

To overcome the limitations of the two-dimensional active cursor we turn to a true three-dimensional cursor. Here a three-dimensional coordinate is defined by the user (usually with the aid of a three-dimensional pointing device such as a Spaceball) and is projected onto the screen. Where a stereoscopic display is available the cursor can be seen to move closer and further away.

The two main difficulties in implementing an effective three-dimensional cursor are: (i) how to draw it onto the screen, and (ii) how to enable the user to judge depth accurately. A number of techniques have been experimented with, each having its particular advantages and disadvantages. In the following sections it may be assumed that for a stereoscopic display the operations described may be done separately for each eye-view.

3·1 Two-Dimensional Overlay Cursor

Here the three-dimensional cursor position is projected onto the screen. A small crosshair is drawn through this projected point. This cursor is always visible, never disappearing even if it is pushed behind a feature. This cursor is very simple to implement and requires the least computation. Unfortunately,

https://doi.org/10.1071/AS97106 Published online by Cambridge University Press

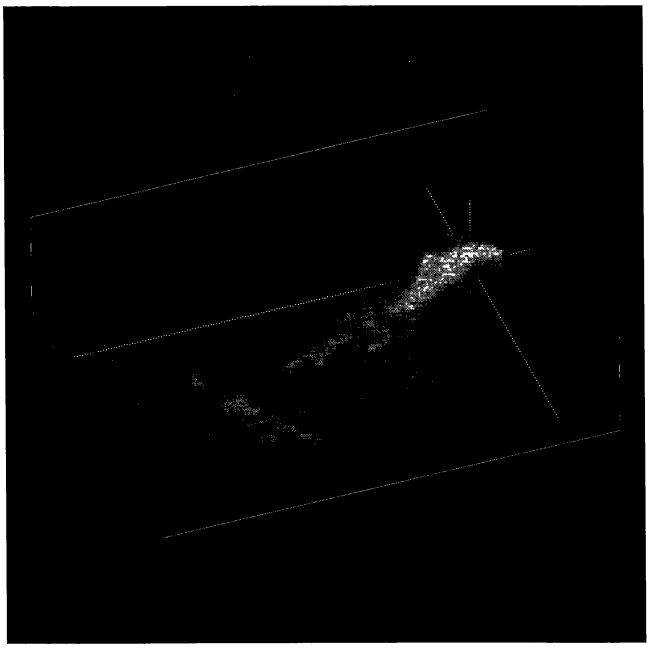

Figure 2—NGC 4631 with three-dimensional shaded cursor.

depth perception is often limited, even when displaying in stereo. Feedback from users indicates that it was too difficult to measure the depth of the cursor. Therefore, while stereoscopic display is an important tool, it should not be solely relied upon to provide depth cues.

3·2 Three-Dimensional Overlay Cursor

Here the cursor position defines three intersecting lines, each line parallel to one of the principle axes of the volume (see Figure 1). These three lines extend from one face of the volume to the opposite face. This may be thought of as a large, three-dimensional crosshair. Depth perception is improved with this cursor. In addition, if a wireframe is drawn around the volume, the user gets an improved sense of where the cursor lies in the volume. As with the previous cursor, this cursor does not disappear when moved behind a feature.

3·3 Two-Dimensional Shaded Cursor

This cursor is similar to the two-dimensional overlay cursor except that an estimate is made of whether the cursor lies in front of or behind an obscuring feature. This estimate is based on the same technique used in the two-dimensional active cursor. If the cursor is in front of the feature, it is drawn, otherwise it is not drawn. While an improvement, this cursor still does not leave the user with much of a sense of where it lies in the volume.

3·4 Three-Dimensional Shaded Cursor

This cursor defines the same three lines as in the three-dimensional overlay cursor, except that, instead of drawing the projected lines over the rendered volume, they are effectively merged into the data. This merging process differs according to the shading algorithm used to render the volume.

For a maximum voxel algorithm each point along the three lines is drawn or not drawn according to whether the point is considered to be in front of or behind an obscuring feature, using the same depth estimation algorithm used by the two-dimensional active cursor.

For an opacity-based shader each point is properly blended with the ray of voxel values passing through that point. If the point is in front of a feature, it is visible; if it is moved behind an opaque feature it will gradually fade to invisibility (see Figure 2).

Of the three-dimensional cursors discussed so far, this one is clearly the best in terms of visual effect, depth perception and placement inside the volume, and received modest praise from users. It is of course the most computationally expensive, but even this cost is relatively moderate.

4 Depth Perception

The nature of astromical data gives it a soft, 'cloudy' effect when volume rendered. This makes the task of implementing effective three-dimensional cursors more difficult. Even with the three-dimensional shaded cursor, the user still finds it difficult to place a cursor accurately relative to a feature of interest.

The use of stereoscopic displays can help depth perception, but often the effect is not enough to allow the user to make accurate depth estimates. The stereo effect can be enhanced by changing the geometry of the eye positions. However, too much enhancement can cause the user to lose the stereo effect. Only users with high visual acuity can enhance the stereo effect sufficiently to place a three-dimensional cursor accurately. To cater for most users, i.e. those with average visual acuities requires an alternative, or an additional, technique.

5 Interactive Slice Plane

If users have difficulty in determining whether a three-dimensional cursor is at a greater depth than a feature, then they may benefit from being able to move an opaque or translucent plane through the volume. This 'slice plane' is merged into the data in a way similar to that used for the cursor. The slice plane may be moved closer to, and further from, the viewing plane (an abstraction in three-dimensional space which corresponds to the screen). As the slice plane is moved from the back to the front of the volume, progressively less of the volume will be visible. If an ambiguity in the structure of a feature exists, this slice plane may be brought forward to obscure all but the closest tip of the feature. Early results indicate that the slice plane can indeed reduce these ambiguities.

6 Performance

Because of the requirement that these features be usable on mid-range workstations, a number of optimisation techniques have to be considered. Some of these techniques are discussed below.

6·1 Damage Repair

The time taken to refresh the display for the two-dimensional overlay and shaded cursors is dominated by the time taken to redraw the image buffer containing the rendered volume. However, since the geometric primitives that have to be moved (namely, two short lines) occupy only a tiny fraction of the image area, huge gains may be made by redrawing only the sub-image over which the cursor was drawn, effectively repairing the damage caused by drawing the cursor.

This technique of damage repair becomes more complicated when the cursor to be 'undrawn' is composed of three, possibly large, diagonal lines. The direct approach would be to define a rectangle which encloses all three lines, or to define three rectangles each enclosing a line. Unfortunately, because these lines are large, the result is that most of the image will be redrawn anyway (and possibly redrawn three times in the latter case). The other extreme would be to rasterise the lines into their component points and redraw sub-images containing each point. This technique requires a large number of small images to be drawn, perhaps several thousand. If the overhead of drawing an image is significant, this technique can take longer than simply drawing the entire image.

A compromise is to draw a smaller number (perhaps fewer than one hundred) of larger images. Unfortunately, this compromise is determined by the overheads in the graphics library/window system that is used, and in the performance of the display and host computer, and will probably vary between combinations thereof. Work is being done to determine optimum compromises for some combinations.

6·2 Depth Planes

To draw a plane that slices through a rendered volume requires not only that the entire image be redrawn, but also that the entire volume be re-rendered, at far greater computational cost. If the position of this plane is to be moved interactively by the user, rendering times must be brought down to a small fraction of a second.

Instead of doing a complete render of the volume, a few shortcuts may be taken. First we assume that as the user moves the position of the slice plane the view position relative to the volume does not change. In the case of a volume rendered with the maximum voxel algorithm, the volume may be rendered to produce not only an image plane (that which the user sees), but also a 'depth plane' which contains an estimate of the depth of the feature the user sees down each ray (corresponding to each of the image pixels).

This depth plane may then be used to merge the slice plane into the volume in a similar way that the three-dimensional shaded cursor is drawn into the volume. The merging process is done on a pixel-by-pixel and basis, hence requires only about N^2 operations, whereas rendering the volume requires about N^3 operations. This is an enormous saving.

The depth-plane technique may be extended to more complex shading algorithms, such as the 'hot gas' shader. Here we compute not only the estimated depth, but also the cumulative opacity to that depth and a measure of opacity rolloff beyond that depth. This algorithm assumes that the volume is transparent until the estimated depth is reached, at which point there is a rapid rise in opacity followed by a gradual decrease. This performs reasonably well for compact, nearly opaque features or extended features. It does not perform well in cases where there is a compact, translucent feature. By adding another estimate of the depth where the opacity suddenly drops, this case is also covered. There are still other cases where this would prove inadequate. However, in the context

https://doi.org/10.1071/AS97106 Published online by Cambridge University Press

of astronomical data these are considered to be of secondary importance. The purpose is to obtain a reasonable first approximation with minimum effort.

The techniques described above deal with avoiding costly rendering operations. When merging a slice plane into a volume, each point on the image plane has to be projected into the volume. The intersection of each projected ray and the slice plane has to be computed and this has to be converted into a depth down the ray. Only then can this depth value be compared with the estimate of the feature depth and a new image pixel computed. The computations involved in finding the depth of the slice plane down a ray tend to dominate the process. It is desirable to have a simple relationship between the position of a slice plane and the depth down a ray of its intersection point with the ray. This relationship exists and is

$$t = \mathbf{N} \cdot \mathbf{SQ} / \mathbf{N} \cdot \mathbf{R} + d / \mathbf{N} \cdot \mathbf{R}, \qquad (2)$$

where t is the distance down the ray of the intersection point in units of $\mathbf{R}$, $\mathbf{N}$ is a unit vector normal to the slice plane, $\mathbf{R}$ is the ray direction vector, d is the distance from a reference plane parallel to the slice plane (in units of $\mathbf{N}$), and $\mathbf{SQ}$ is the vector from the starting position of the ray ($\mathbf{S}$) to a point on the reference plane ($\mathbf{Q}$). This is of the form

$$t = \alpha + d\beta . \qquad (3)$$

Once α and β have been computed for each image pixel, the distance of the slice plane from the reference plane may be altered (by changing d) and a new image can be computed very quickly. It is possible to get update rates of several frames a second on a mid-range workstation.

7 Future Work

Astronomers are interested in drawing polyhedra around an interesting feature, extracting the data in that sub-volume and analysing them. As more complex shapes need to be drawn inside a volume of data, a generalisation of the depth plane may be necessary. Since this depth plane is somewhat analogous to the familiar 'Z-buffer' used in surface-rendering graphics libraries, we may see these developments as a hybrid of volumetric rendering and surface rendering.

8 Summary

Taking visualisation of astronomical data into the realm of interacting with the data has posed new problems that require new solutions. As astronomers demand tools to give them greater insight into their data, placing greater demands on visualisation software, we hope to continue to develop interesting solutions to these problems.

Acknowledgments

I thank Ray Norris and Tom Oosterloo for ideas and contributions to the Visualisation Project.

Gooch, R. E., 1995*a*, in Astronomical Data Analysis Software and Systems IV, ASP Conf. Ser. 77, ed. R. A. Shaw, H. E. Payne & J. J. E. Hayes (San Francisco: ASP), p. 144

Gooch, R. E. 1995*b*, in Workshop on Applications of Radio Science 1995, Australian Academy of Science through the National Committee for Radio Science

Gooch, R. E. 1995*c*, in IEEE Visualisation '95, ed. G. M. Nielson & D. Silver, (IEEE), p. 374

Norris, R. P. 1994, in Astronomical Data Analysis Software and Systems III, ASP Conf. Ser. 61, ed. D. R. Crabtree, R. J. Hanisch & J. Barnes (San Francisco: ASP)

https://doi.org/10.1071/AS97106 Published online by Cambridge University Press