Displaying functions of three variables

J. Butland
B.U.S.S. Ltd., 29 Campus Road, Bradford, West Yorkshire BD7 1HR, UK

A. R. Leach
Chemical Crystallography Laboratory, 9 Parks Road, Oxford OX1 3PD, UK

In this paper we discuss the problem of graphically representing functions of three variables and explain the implementation of a method in which each picture shows the 3D solid shape described by one data level. The method enables us to plot individual data levels from any REAL array ARR(NX, NY, NZ) of equally spaced values over an NX by NY by NZ regular grid. A perspective representation of each 3D shape can be drawn from any chosen viewing position with hidden-line removal; this is illustrated by some pictures drawn using a Fortran 77 program.

Keywords: hidden line removal, 4D data, electrostatic potential, des-iodo-thyroxine

Received 26 August 1987
Accepted 15 December 1987

INTRODUCTION

Given a REAL array ARR(NX, NY, NZ) containing data measurements in three-dimensional (3D) space, the program we have written will draw a graphic representation of the solid object (assuming continuity of the data) whose surface traces a particular data value DVAL. In accordance with the general definitions given by Smith, Price and Absar,[1] such representations are known as 4-dimensional (4D) graphs, the data consisting of a scalar function of three variables that we term "4-dimensional data." Such sets of measurements of REAL quantities that vary in 3D space are collected in many fields; electrostatic potentials and electron density maps are of particular interest to those using molecular graphics. Meaningful representation of such data can be awkward; one common method of illustrating its behavior is to draw contour maps of plane sections through the data, sometimes taking a sequence of parallel planes. To get an all-around view, sections can be taken in different perpendicular planes.

Another way to represent 4D data is to form the image of a solid shape by taking a series of parallel planes through the data and drawing contours on each at the level DVAL with hidden-line removal. The resulting solid shape can be viewed from any angle. The data can be represented by a set of pictures of the solid shapes of a few data levels (Color Plates 1–4). The data used to draw these is of the electrostatic potential of des-iodo-thyroxine (Figure 1) at various levels, generated using the QCPE program VSS.[2] This data has also been used to draw the subsequent figures, the values being in kcal mol^{-1}.

The technique of forming the image by drawing contours on a series of parallel planes is particularly suited to the production of easily interpretable pictures of 4D data on hard-copy vector graphic devices. This was one of our major objectives. As a consequence, the algorithm has to perform its own hidden-line removal and produce a series of line vectors that, when connected in the correct order, produce the desired representation.

The wire-frame pictures drawn, for example, by Jorgensen's PSI77 molecular orbital plotting program[3,4] are another way to produce solid shapes to represent 4D data on plotting devices. This program has been widely used; among its features is the ability to draw more than one isopotential on the same picture, differentiating between them by the use of color. It has been noted,[5] however, that problems can arise if the surface is not closed.

We took as our starting point an algorithm described by Wright[6] for representing an object by constructing contours in parallel planes. The object is described by a 3D array in which the value 1 indicates the presence of the object and the value 0 the absence. For the 4D case, this idea needs extending to use a REAL value at each point; the presence or absence of the object is determined by whether the data value is greater than or smaller than the chosen level.

This method has several drawbacks, including:

(1) A bit map in an array is used as a model of the image plane, filled initially with 0s and with

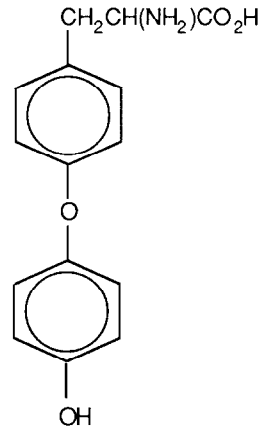


Figure 1. Des-iodo-thyroxine

individual bits switched to 1s to represent the shape of the picture as it is drawn. For each line segment to be drawn, the two end points are perspectively transformed onto the image plane. The line is drawn only if both of the end points are visible, as indicated by zero values in the appropriate elements of the visibility array. However, this bit map is not a true image map. Its ideal size relates tenuously to the size of the data array; hidden lines are not removed properly if it is too big, while some visible lines are omitted if it is too small. Wright suggests some reasonable sizes for a few data dimensions and advises experimental adjustments where necessary.

(2) In the Fortran implementation listed, nonstandard methods are needed to manipulate bits.

(3) All the curves are necessarily very angular in form, consisting of combinations of vectors in only four possible directions.

Pattnaik and coworkers[7] improve on Wright's technique in two ways. They use linear interpolation to calculate the point where the boundary of the cross section intersects each rectangle, thus producing sharper plots, and a sort-merge procedure to decrease the execution time of the plot on the pen-plotter. They still use Wright's method of testing for the visibility of a line segment, transforming the two end points onto the image plane and drawing only if both end points are visible, as indicated by reference to the visibility array. Grassy *et al.*[8] have used a similar representation to investigate structure-activity relationships, but they give no details of the algorithms used to construct their pictures.

OBJECTIVES FOR THE NEW ALGORITHM

We decided that with REAL data, we could interpolate values on cuts in parallel planes other than data planes and that we could achieve the best representation of any data from any viewing position by drawing contour curves in a sequence of planes perpendicular to the viewing direction. This strategy ensures intelligible pictures without sudden changes in rotating sequences (Figure 2). When the direction of cut is not tied to the data planes, the number of slices no longer has a fixed value; we decided to make the number of slices between the front and back of the object user-selectable, with a default of 20.

Any data configuration can be represented, providing that DVAL lies within the range of the data array values. For some data, and some values of DVAL, the 3D curve of DVAL forms closed objects like Wright's data, but many other shapes can occur and should be drawn. A 3D object has two distinct surfaces. For a closed object, one surface faces outwards and is visible. The other, inward-facing surface is hidden (Color Plate 1). When the surface is not closed, however, parts of both surfaces might be visible simultaneously (Color Plates 2–4). In order to give a full representation of any data, we decided to draw any visible parts of both surfaces, distinguishing them by different colors when possible.

DRAWING THE PICTURES

For any viewing position, it is possible to build up a picture working from the front to the back of the object, without any of the sorting that dominates the more general hidden-line methods.[9] The remaining issue is the method of recording occupied areas of the image and using the record to mask subsequent additions to the picture. This algorithm differs significantly from the one published by Wright, both in the method it uses for calculating the lines to be drawn and in the visibility checking.

Two real 2D arrays of identical dimensions are used for the line drawing and hidden-line implementation. One of the arrays, the image array, is repeatedly used to hold images of the plane cuts through the data array. The other, visibility array, is used to accumulate a running visibility record of the total picture drawn at any stage. Corresponding elements in both arrays relate to the same points on a regular grid. The sizes of the working arrays are independent of the resolution of the graphics device and are also independent of the data dimensions; they can be chosen to be relatively small (e.g., 10 by 10) to give rapidly executed approximations, or large (e.g., 40 by 40) to draw more accurate representations (Figure 3). During the assembly of the picture, elements of the visibility array are assigned positive values for visible parts of the picture and negative values for hidden parts of the picture, such that the contour curve at zero (0.0) from the visibility array would trace an outline giving the extent of the object drawn so far.

Before the picture is started, the data array is scanned to calculate the area of the drawing plane that is occupied by the image of the object. At the same time, a REAL value outside the data range is selected; this value is used to represent the absence of data at a point.

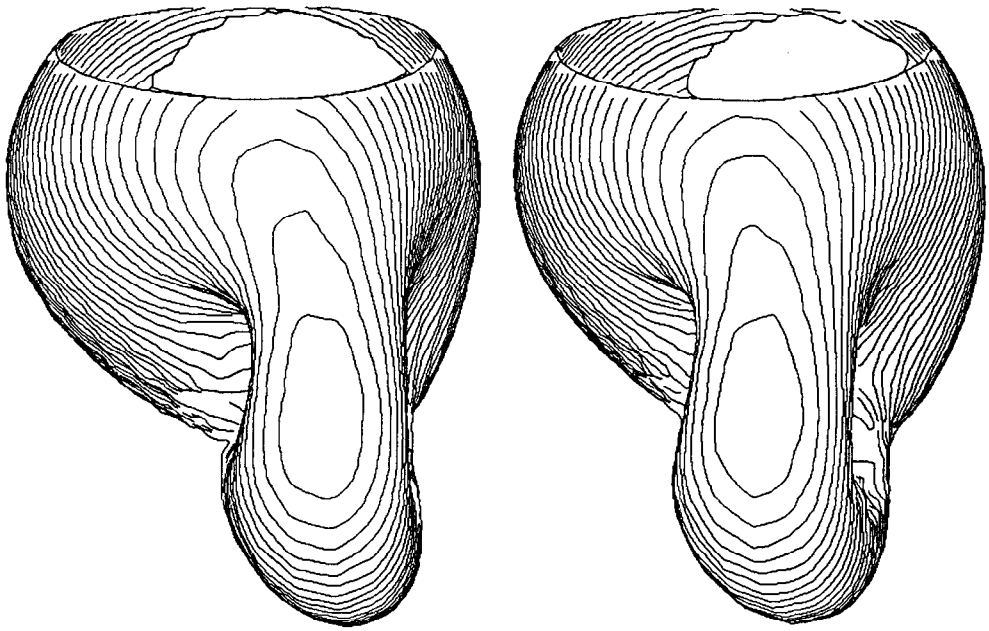
COORDINATE SYSTEMS USED

The relationship between the user and the unrotated data is described using a 3D coordinate system. The data origin is positioned at the center of the data array; x increases toward the observer, y increases from left to right and z increases vertically upward. Alternative views are specified using three parameters: DIST, the distance between the observer and the data origin; H, the angle by which the data is rotated horizontally about its z-axis; and V, the angle by which the data is rotated vertically about its y-axis. In this data coordinate system, one scale unit corresponds to the interval between adjacent array elements, relative to the origin at the center of the data array.

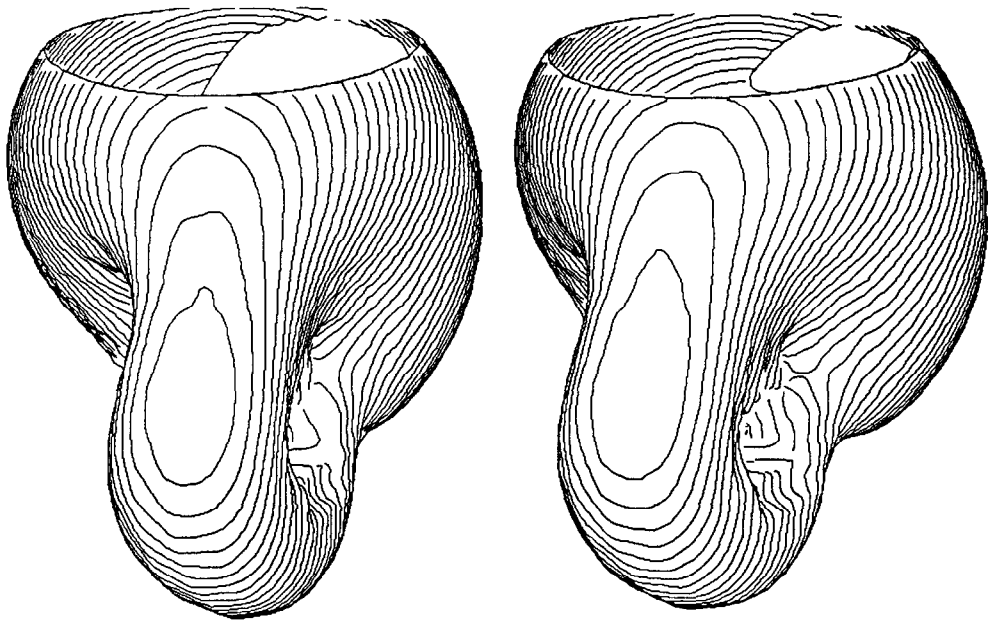
The perspective effect is achieved by calculating the projection of the object onto an image plane that is perpendicular to the line joining the observer to the origin of the data coordinate system. This image plane thus passes through the center of the data; the 2D image coordinate system has the same units and origin as the data coordinates. Each element in the image array thus has specific (XI, YI) image coordinates that are evaluated by offsetting each subscript by a constant related to the dimension of the image array and multiplying by a scale factor to convert to data units.

FILLING THE IMAGE ARRAY FOR EACH SLICE

Within a given plane through the data, a unique point projects onto a point (XI, YI) in the image plane. The



2a Horizontal angle= 10 2b Horizontal angle= 20



2c Horizontal angle= 30 2d Horizontal angle= 40

Figure 2. Rotating sequences: (a) horizontal angle = 10°; (b) horizontal angle = 20°; (c) horizontal angle = 30°; (d) horizontal angle = 40°

interpretation of the adjustable viewing position, the distance of the slice from the user, and an inverse perspective transformation are combined in the calculation of the (XD, YD, ZD) data coordinates of this point in the plane. Consequently, for every element in the image array there is an associated (XD, YD, ZD) point in each of the slices through the data used to construct the picture. If this calculated (XD, YD, ZD) point lies outside the data array, the "no data" value is placed into that element of the image array. Otherwise, the data value at the point (XD, YD, ZD) is linearly interpolated between the eight surrounding data points and the resulting value placed in the image array. This is the important difference to Wright's algorithm; he transforms the current data slice onto the image plane and then tests each line segment for visibility, whereas in

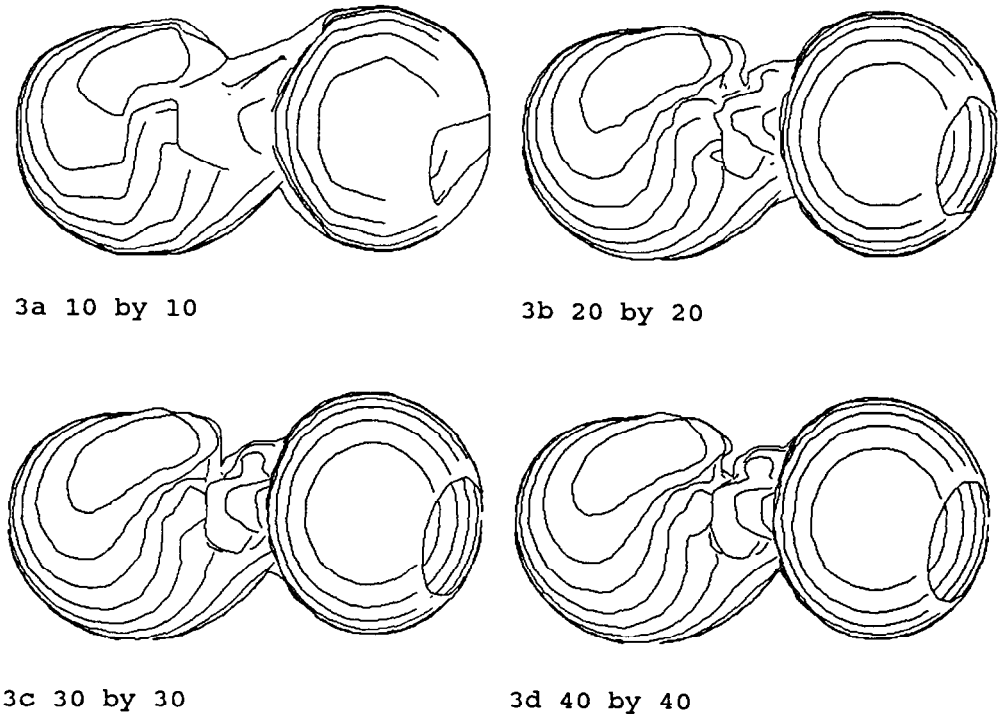


Figure 3. Different-sized working arrays: (a) 10 by 10; (b) 20 by 20; (c) 30 by 30; (d) 40 by 40

the technique described here the image array is transformed to give a plane through the data that is used for all the segment drawing and hidden-line removal. The contour drawn at DVAL from the interpolated values in the image array thus gives the projection of the current slice through the data onto the drawing plane. This is illustrated in Figures 4c and 4e for the first two slices, in which the black-shaded area encloses the contour drawn at DVAL (-0.1 in this example) for the slice. Grid rectangles that have "no data" values in any corner are left unshaded. Gray shading covers those elements of the image array that contain values greater than DVAL.

We decided to use the simplest form of interpolation possible while developing the algorithm and to consider a more rigorous approach later if improvements seemed desirable. In fact, the pictures constructed proved sufficiently good for linear interpolation to be retained.

When interpreting Figures 4b to 4f, note that the slices are taken from the front to the back of the object, rather than through the entire data array. Calculating the positions of the first and last slices is performed during the initial scan through the data. In addition, the arrays cover the area of the picture occupied by the image of the object, which is also calculated during the initial scan through the data.

DRAWING THE LINES

The drawing for each slice is determined solely from the values in the image array, subject to visibility as recognized by reference to the visibility array. The computation involved in the hidden-line testing is relatively simple because of the one-to-one correspondence between points in the visibility and image arrays. For each rectangular region of the picture contained by four adjacent elements of the image array, a simple comparison of values establishes whether or not the DVAL contour crosses the region. When the contour does cross the rectangle, linear interpolation is used to find the coordinates of the two ends of the line. Linear interpolation is also used to calculate whether these two end points are visible, from the values in the visibility array (given that the contour drawn at 0.0 from the visibility array gives the extent of the picture drawn so far). When both are visible, the line is drawn; when both are hidden, the line is omitted; and when one is visible and the other hidden, the zero point on the boundary of the visible area is interpolated between them, and the visible part of the line is drawn.

The drawing for each slice is generated as a set of separate line fragments in an order that is inconvenient for plotting by pen and unsuitable for converting into smooth curves. This algorithm was developed within the context of a data plotting package, SIMPLEPLOT, which already drew conventional contour curves using subroutines that sorted such fragments;[10] the SIMPLEPLOT subroutines were invoked to organize these line fragments similarly.

A buffer with a capacity for about 50 coordinate pairs collects the coordinates of the ends of line fragments as they are generated; an associated index to the buffer keeps a record of which of the buffered coordinates begin connected sequences of points. Both ends of all the buffered sequences are compared with the coordinates of each end of a newly generated vector to find if it is sufficiently close to any existing ends to be assumed coincident. In this way, duplicate points are eliminated, chains are extended, and sometimes two chains are linked together. Whenever the buffer is full, or when the curve is fully assembled, the buffer's contents are drawn as connected sequences; an option has been pro-

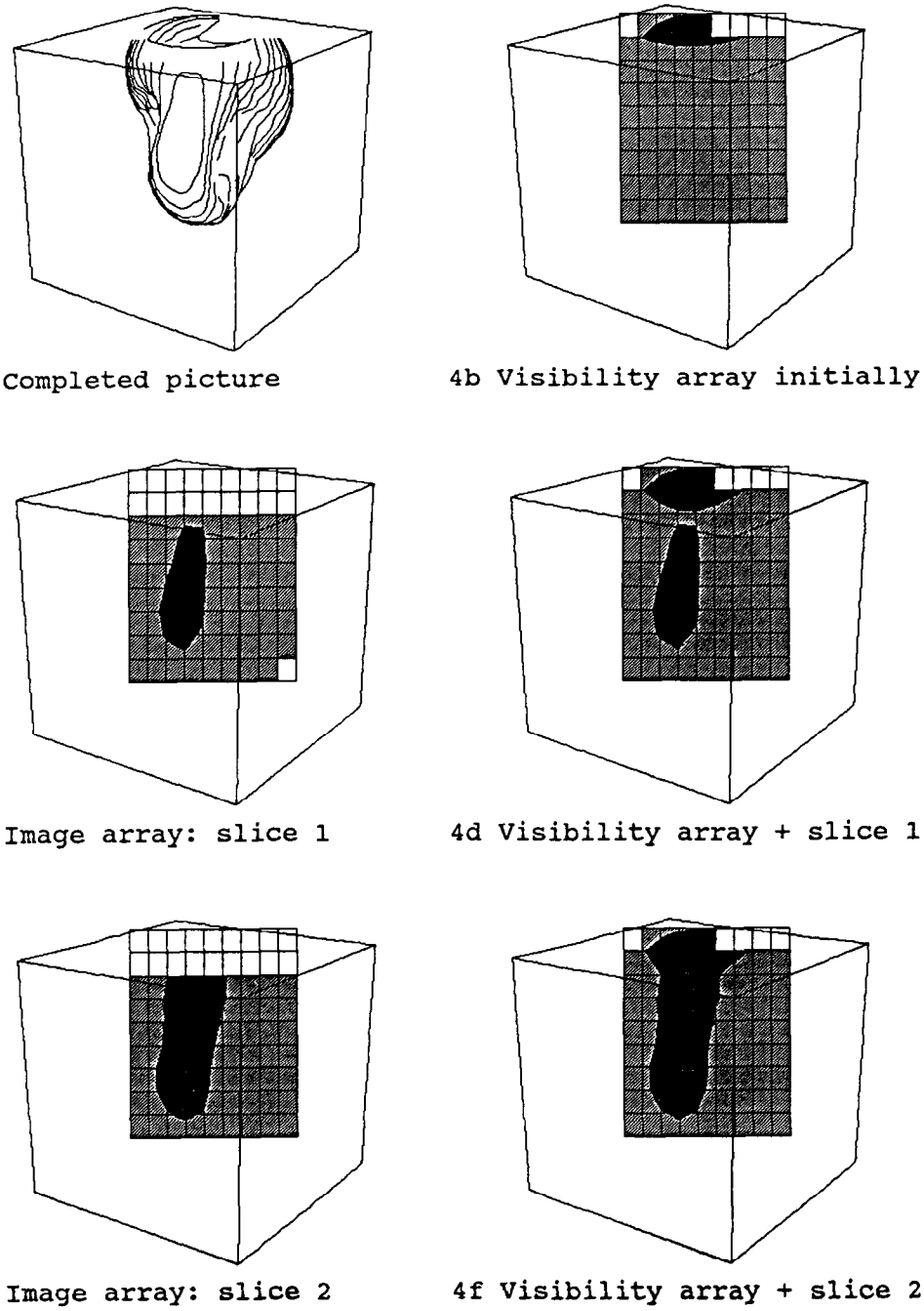


Figure 4. Constructing the picture: (a) completed picture; (b) visibility array initially; (c) image array, slice 1; (d) visibility array plus slice 1; (e) image array, slice 2; (f) visibility array plus slice 2

vided for the sequences to be joined by either straight lines or smooth curves.

UPDATING THE VISIBILITY ARRAY

After each slice has been drawn, the visibility array is updated; each element VA in the visibility array is either left unchanged or replaced, depending on *S*, the value of the corresponding element of the image array. For each position in the slice where *S* indicates "no data," the visibility array is left unchanged; otherwise, VIS, the image visibility, is calculated. The visibility value, VIS, is related to the interpolated value, *S*, and the data level being drawn, DVAL, by a simple subtraction:

$$VIS = S - DVAL$$

VA is replaced by VIS when VIS is smaller than VA; otherwise, it is left unchanged.

The effect of updating the visibility array can be seen in Figures 4b, 4d and 4f. Figure 4b shows the status of the visibility array initially, before any "slicing" is performed. In this initialization each element is tested to see if it lies on one of the faces of the data array visible to the observer. If it does, the data value at that

point is interpolated from the surrounding four data values on the face. In Figure 4 there are three such faces of the data array visible to the observer. A visibility value is then calculated for each element that lies on such a face. Elements that lie outside the data array are assigned a "no data" value. Otherwise, the visibility value is calculated as above (VIS = S − DVAL). The contour line drawn at 0.0 from the visibility array after initialization thus describes those areas of the object that are clipped by a face of the data array (i.e., where the object protudes through a visible face of the data array). The black-shaded area in Figure 4b represents the image of the object where it is clipped by the top face of the data array (VIS negative, DVAL > S). Grid rectangles that have "no data" values in any corner are left unshaded. The remaining areas are shaded gray; the elements in the visibility array that are covered by this shading have positive visibility values. If all the initial visibility values are negative, the surface is completely hidden, and no further processing is necessary. Figures 4d and 4f show the effect of updating the visibility array with the first two slices; black shading indicates the extent of the picture drawn at each stage.

Figure 5 illustrates the logic behind the simple procedure used to update the visibility array. In Figure 5a the image visibility values are negative within the shaded area; previous parts of the picture hide the shaded area in Figure 5b where the visibility array values are negative. The visibility array must be updated to define a new hidden region, which is the union of the two shaded areas (i.e., it must remain negative where it is already negative and become negative wherever the image visibility is negative). The dotted line through the intersections of the superimposed shapes in Figure 5c traces where the image visibilities equal the values in the visibility array. To the left of the dotted line, image visibilities are smaller than the values in the visibility array, so replacement takes place. To the right of the dotted line, visibility array values are smaller, and therefore left intact. The updated visibility array consists of the left-hand side of Figure 5a, combined with the right-hand side of Figure 5b.

When the visibility and image arrays are small (e.g., 10 by 10), some of the lines might appear incomplete. For example, in Figure 4a some of the lines do not completely connect with those drawn for previous slices.

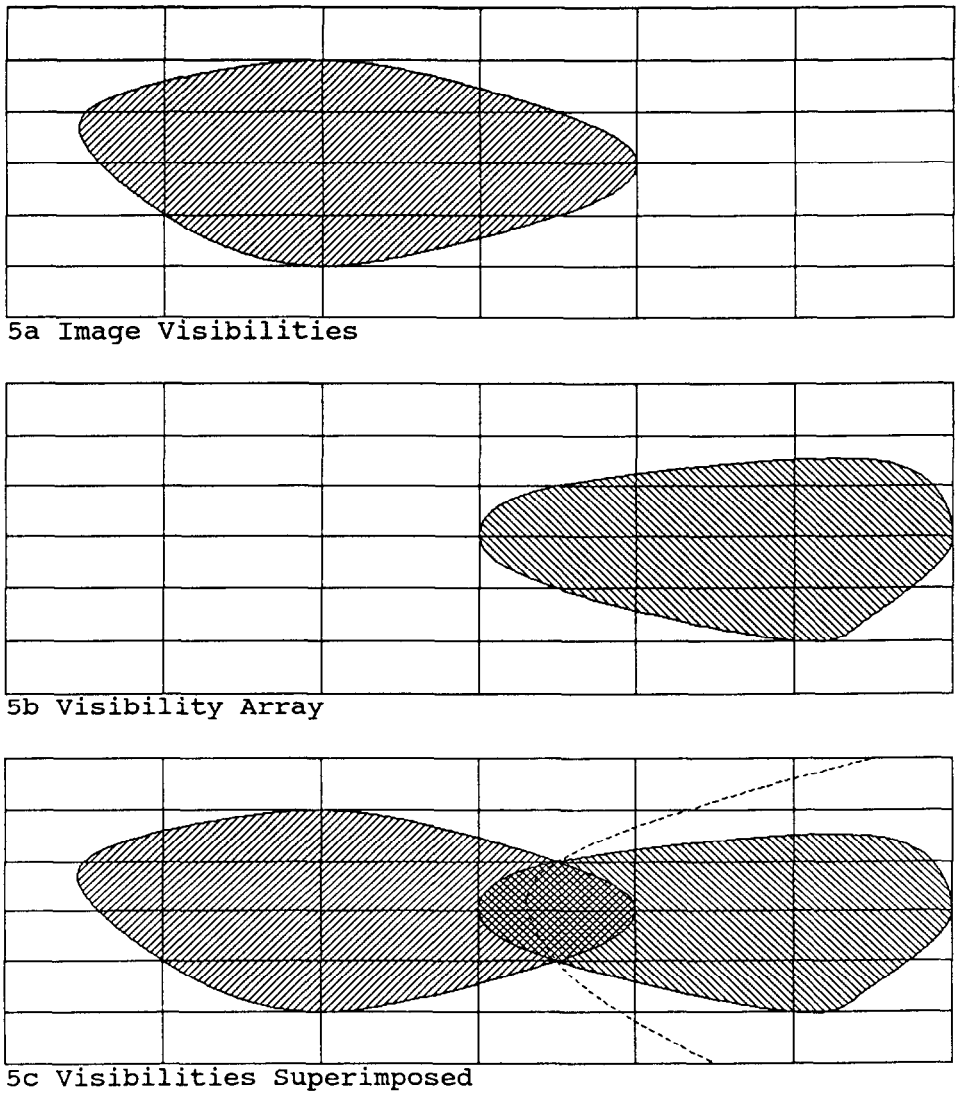


Figure 5. Updating the visibility array: (a) image visibilities; (b) visibility array; (c) visibilities superimposed

This is due in part to the use of linear interpolation to determine the intersection of a line segment with the previously drawn parts of the picture. It is also a consequence of the way in which the visibility array is updated. For example, the black-shaded area in Figure 4b is larger in Figure 4d after the visibility array has been updated with the first slice. This is due to the replacement of values in the visibility array with smaller VIS values from the image array; the result is that the contour drawn at 0.0 from the visibility array traces a larger area. In addition, some of the detail of the picture might be missing. The effect of this approximation is most evident when small arrays are used (contrast Figure 4a with Color Plate 2), but the use of larger arrays requires a corresponding increase in the time required to draw the picture.

The whole procedure described above is executed once for both surfaces of the object. For one surface, visibility values are given by VIS = S − DVAL and the visibility array is updated treating values less than DVAL as opaque. For the other surface, the visibility values are given by VIS = DVAL − S and the visibility array is updated treating values greater than DVAL as opaque.

CONCLUSIONS

Before the advent of computer graphics, curves were usually drawn to show the behavior of 2-dimensional data. Contour maps, which have long been used in some disciplines to display 3-dimensional data, have acquired wider use because of the availability of computer software to draw them. Our algorithm for graphically representing any configuration of 4-dimensional data extends the scope of data plotting to the next dimension. The computing requirements for the method can be controlled by user-selected parameters; the execution time depends mainly on the number of slices through the data and on the (identical) sizes selected for the two working arrays that control the accuracy of the drawing. The amount of computation is independent of the size of the data array but increases with the complexity of the shape of the drawn object.

Some of the advantages of the method were not obvious until we had actually produced the pictures. The use of planes parallel to the user provides a more acceptable picture from all observation points (which is particularly important in the generation of rotating sequences) and produces a pseudo-highlighting effect, with the lines being more closely spaced as a side of the object becomes invisible and more widely spaced close to the observer.

Color Plate 5, which is also of the electrostatic potential of des-iodo-thyroxine, gives a preview of further developments. Two data levels (at −0.1 and 0.1 kcal mol^{-1}), each of which partly hides the other, are drawn in one picture. Only one surface is drawn for each level, and shaded contours of the data on the boundary planes are inserted in the holes where the other surface would be visible.

The software was developed on a VAX 11/730 with output to a variety of graphics devices, including the SIGMA 6000 series, a Calcomp 81 plotter and an HP Laserjet Plus printer. It is available as part of the SIMPLEPLOT graph-drawing package; further details can be obtained from B.U.S.S. Ltd.

ACKNOWLEDGEMENTS

We thank Smith Kline and French Research Ltd. for funding the initial development work. We are grateful to Dr. J. G. Vinter for helpful discussions and for providing the data used to draw the figures. We would also like to thank Dr. C. K. Prout and Dr. D. J. Watkin for their advice in preparing the manuscript and Dr. N. C. Cohen for his correspondence. ARL would like to thank B.U.S.S. Ltd. for vacation employment while he was engaged on the project.

REFERENCES

1 Smith, V. H. Jr., Price, P. F., and Absov, I. Representation of the electron density and its topographical features. *Israel J. Chem.* 1977, **16,** 187–197
2 Giessner-Prettre, C., and Pullman, A. *Theoretica Chim Acta,* 1972, **25,** 83–88
3 Jorgensen, W. L., and Salem, L. *The Organic Chemist's Book of Orbitals.* Academic Press, New York, 1973
4 Jorgensen, W. L. Quantum Chemistry Program Exchange, Indiana University, Program 340
5 Purvis, G. D., and Culberson, C. On the graphical display of molecular electrostatic force-fields and gradients of the electron density. *J. Mol. Graph.,* 1986, **4(2)** 88–92
6 Wright, T. Visible Surface Plotting Program. *Comm. ACM,* 1974, **17(3),** 152–155
7 Pattnaik, P. C., Dickinson, P. H., and Fry, J. L. Fermi-surface: a package to display perspective drawings of Fermi surfaces in cubic systems. *Comp. Phys. Comm.,* 1982, **25,** 63–71
8 Grassy, G., *et al.* Visualisation tridimensionnelle des potentials electrostatiques moleculaires. *European J. Medic. Chem.,* 1985, **20(6),** 501–508
9 Sutherland, I. E., Sproull, R. F., and Schumacker, R. A. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys,* 1974, **6(1),** 1–55
10 Butland, J. Simpleplot Mark 2, Section 2: Plotting 3-Dimensional Data. B.U.S.S. Ltd., 1984.