Depth-buffer algorithms for molecular modelling

Michael L Connolly

Department of Molecular Biology, Research Institute of Scripps Clinic, La Jolla, CA, USA

Methods for drawing sophisticated pictures of molecules on colour-raster graphics terminals are presented. These methods utilize depth buffers, which store, for each pixel, not only a colour and a shade, but also a z coordinate or depth. Depth buffers make it convenient to combine several different molecular representations into a single picture. The algorithms can produce cutaway views of protein interiors, cross-sections of protein–protein interfaces, and translucent solvent-accessible surfaces through which the chemical structure may be seen.

Keywords: raster graphics, depth buffers, algorithms

received 8 May 1984

In molecular modelling on raster graphics systems, the atoms are usually represented as spheres[1–2], although recently secondary structure has also been modelled[3–4]. The work presented here is an attempt to transfer some of the capabilities of vector molecular modelling systems to a raster graphics system:

- the ability to clip the image so that the molecular interior is visible[5]
- the ability to display a translucent molecular surface[5–7]
- the ability to display general graphical objects in conjunction with molecular models[8–9]

Implementing these capabilities has required the development of a hidden-surface algorithm based on a 3D depth buffer. A depth buffer[10] stores the depth or z coordinate of each pixel. A 2D depth buffer stores only one pixel for each xy position: the pixel nearest the observer. A 3D depth buffer[11] stores several pixels (or rather voxels) for each xy position, each at a different depth. Hidden surfaces are not eliminated, but rather sorted by z coordinate, so they can be revealed by clipping and translucency.

Before this 3D algorithm is presented, a simpler, 2D depth buffer algorithm will be described. A wide variety of images computed by these algorithms are given as examples.

2D DEPTH BUFFER

This algorithm converts an analytically defined solvent-accessible molecular surface into a square array of pixels. The surface consists of convex spherical, saddle-shaped toroidal, and concave spherical faces generated by rolling a probe sphere over the molecule[12–14]. This surface will be referred to by either of the two terms, molecular surface or solvent-accessible surface. The latter term was originally used to refer to a surface pushed outward from the van der Waals surface by a distance equal to the radius of the probe[12], but that surface is less useful for graphical applications. The saddle-shaped and concave faces will collectively be referred to as re-entrant surface.

Each pixel has associated with it a hue, shade and z coordinate (height or depth). The pixel hue is determined by an input file specifying the colour of each atom. The convex and re-entrant surface[13] of each atom may be given either the same, or differing, colours. Each face of the re-entrant surface bridges two or three atoms. Each saddle-shaped face is divided into two regions, and each concave triangular face is divided into three regions, so re-entrant surface may be coloured according to the nearest atom.

The pixel shade is determined from the angle between the surface normal vector and the direction of the light source, and by the height or depth of the pixel along the z axis. The relative influence of these two factors is controlled by an input parameter.

The height or z coordinate of the pixel is determined by different methods for spherical and toroidal faces, as discussed below. Each face of the surface is considered in turn and the pixels it generates are put into the depth buffer. If there is already an old pixel with the same xy coordinates as the newly generated pixel, the depths of the two pixels are compared, and the one further from the observer is discarded, because it is hidden from view by the nearer pixel.

The method for generating pixels from a saddle face is shown in Figure 1(a). The face is subdivided by a

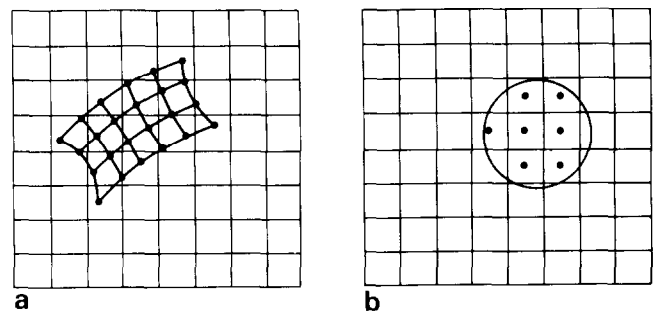

Figure 1. (a) Projection of saddle-face grid onto xy plane. The mesh of the grid is chosen to be fine enough so that each pixel underneath the saddle face contains the projection of at least one node of the grid. (b) Intersection of a sphere with rays parallel to the z axis passing through centres of pixels

 0263-7855/85/010012-06 $03.00 © 1985 Butterworth & Co (Publishers) Ltd

rectangular grid, and each node of the grid is projected onto the *xy* plane. If more than one node is projected onto a particular pixel, the node nearest the observer is used.

The spherical faces are handled by considering an array of rays parallel to the *z* axis, passing through the centres of each pixel. The rays intersecting the sphere are selected (Figure 1(b)). An intersection point between each ray and the sphere is calculated. Of course, a ray intersects a sphere in two points, but only one of these intersections is used. For convex faces, the higher point is used (Figure 2(a)); but for concave faces, the lower point is used (Figure 2(b)). For concave faces that intersect each other, intersection points on one sphere lying inside the volume of the other sphere must be removed (Figure 2(c)). A similar removal occurs for saddle faces.

Since a spherical face covers only part of the area of a sphere, a decision must be made about whether the intersection point is inside the face. Concave faces are geodesic triangles lying on a probe sphere cut out by three planes passing through the probe sphere centre. If the intersection point lies on the correct side of each of these planes, then it belongs to the concave face (Figure 3(a)). Convex faces are borderd by arcs,

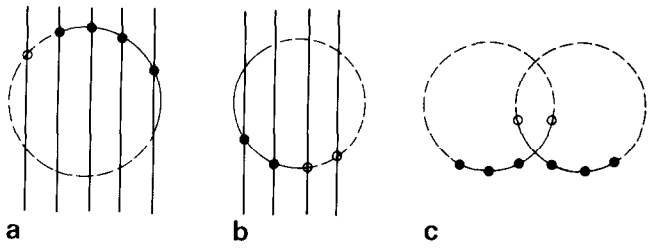

Figure 2. (a) Rays intersecting atomic sphere. The observer is at the top of the diagram (positive z). The solid arc represents the convex face, the dashed arc represents the remainder of the sphere. Intersection points of the rays with the convex face are represented as solid circles, other intersection points as open circles. (b) Concave face. The relevant intersection points between the pixel rays and the probe sphere occur on the lower hemisphere. (c) Overlapping re-entrant surface removal. Open circles: deleted pixels

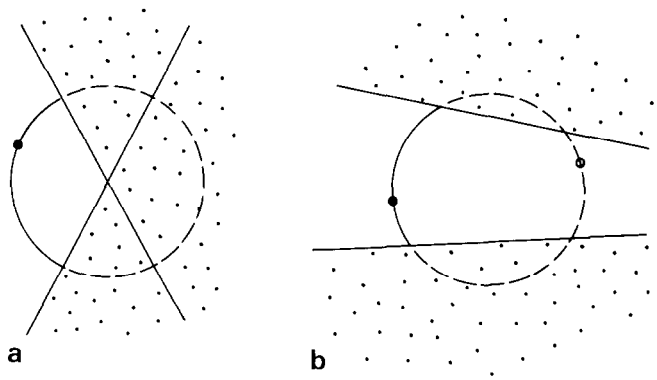

Figure 3. Two-dimensional representation: determining whether an intersection point lies within a spherical face (represented as a solid arc). The lines represent the planes of the arcs that border the face. The regions on the forbidden sides of the arc planes are stippled. (a) Concave face. (b) Convex face. The small open circle represents a point that passes the plane test, despite the fact that it does not lie on the face

where the arcs are parts of circles of contact between the atom and the probe sphere as it rolls along an interatomic crevice[14]. The contact circles lie on planes which do not necessarily pass through the atom centre. A necessary (but not sufficient) condition for the intersection point to lie inside the convex face is that the point lies on the correct side of each of these planes (Figure 3(b)). Stereographic projection is used as a further test for intersection points passing the contact circle plane test, in a manner analogous to its use in the analytical molecular surface algorithm[14].

The computer program implementing the 2D depth buffer algorithm is called RAMS (raster analytical molecular surface). RAMS is written in Fortran 77 and runs on a VAX-11/750 operating under VMS. RAMS accepts as input an analytical molecular surface, as calculated by the AMS program[14], an atomic colouring file, and a parameter file. The parameter file specifies a molecular rotation, a window, the direction of the light source and the degree of depth cueing. The output of the program is a binary disc file of pixels, one byte per pixel, which is displayed on an AED 767 colour raster terminal. The program uses 20–60 minutes of CPU time to generate an image, depending mainly on the area of the screen covered by the image, and secondarily on the number of atoms.

3D DEPTH BUFFER

The 3D depth buffer program can display the analytical molecular surface and also more general objects. The faces of the objects may be not only pieces of spheres and tori, but also polygons and pieces of cylinders and cones. The boundary of a face may be formed not only by circular arcs, but also by straight line segments.

A ray parallel to the *z* axis passing through a given *xy* position intersects the surfaces of the objects in several points. In a 2D depth buffer, only the nearest intersection point is stored, but in a 3D depth buffer, all intersection points are stored. The 3D analogue of a pixel is a voxel. For each *xy* position in the image, there is a linked list of voxels, sorted in order of depth. The first voxel of the list is nearest to the observer. Each voxel has a depth or *z* value, a shade, a face number, and a flag stating whether the voxel corresponds to the inner or outer side of the surface. A face number, rather than a hue, is stored so that several related images with different colour schemes may be produced, with only one pass through the hidden-surface algorithm.

Even with a 32-bit virtual address computer, such as a VAX-11, it is difficult to store an entire 3D depth buffer in program memory. For this reason, the 576 × 768 image area is divided up into 12 × 16 = 192 square-shaped cells, each 48 pixels on a side. There is one scratch file on disc for each cell. Only one cell at a time is in the program memory.

For each face of each object, a list is formed of the cells it may intersect. The first step is to transform the coordinates of the face by the current translation and rotation of the object. The next step is to determine the minimum and maximum extents in *x* and *y* coordinates of the face. Such extrema may occur on a face, on an edge, or at a point. These extrema define a rectangle circumscribing the projection of the face onto the *xy* plane (Figure 4). Any cells that overlap the cir-

cumscribing rectangle are put into a list of cells that the face may intersect.

From the cell lists for each face, a converse set of lists is formed, namely a list for each cell of the faces it intersects. This makes it possible to fill the depth buffer with voxels, with only one cell in core at a time. The creation of voxels is done by algorithms that are similar to those described for the 2D depth buffer hidden-surface elimination algorithm, except that now hidden surfaces are sorted by depth, rather than eliminated. There are additional complexities because both the outer and inner sides of the surface must be considered, and because more general objects are handled.

The outward-pointing normal vector at a point on the surface determines whether the inner or outer side of the surface faces an observer located at positive z (Figure 5). If the z coordinate of the normal vector is positive, the point represents the outer side of the surface; if negative, it represents the inner side of the surface.

Cylindrical faces are handled by a grid, in a manner similar to the handling of saddle-shaped faces. Conical and polygonal faces are handled by calculating the intersection of a ray along the z axis with the cone or plane, and then checking whether the intersection point is within the boundary contour of the face.

The voxels belonging to a given xy position are stored as a linked list, rather than in an array, for two reasons. First, the number of voxels belonging to an xy position varies from zero to several dozen, and an array would need to be dimensioned to the maximum number, wasting storage space. Second, a new voxel being entered into the depth buffer can be inserted into its proper place in the linked list, based upon its z coordinate, merely by changing a couple of pointers. In contrast, inserting a voxel into an array would require shifting all deeper voxels by one position.

Once the 3D depth buffer has been filled with voxels from the objects, 2D arrays of pixels, that is pictures or frames, may be written to disc. The picture created depends not only on the contents of the depth buffer, but also on how the objects are coloured and clipped, and whether they are solid or hollow.

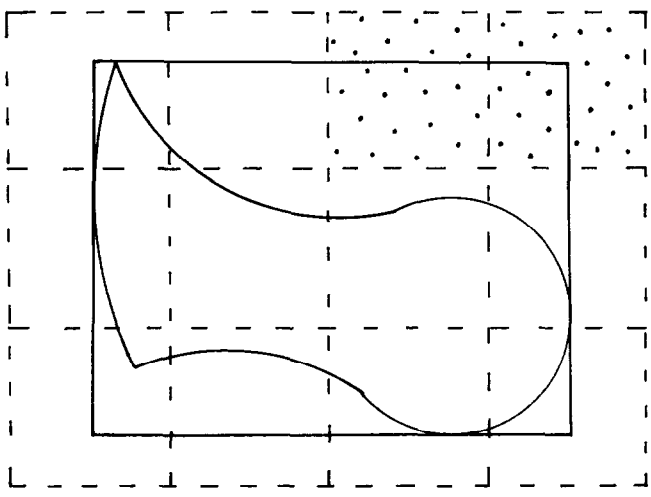

Figure 4. Rectangle circumscribing the projection of a face onto the xy plane. Cells (stippled) that intersect the circumscribing rectangle, but not the projected face itself, are included in the cell list for the face, causing some inefficiency, but no error

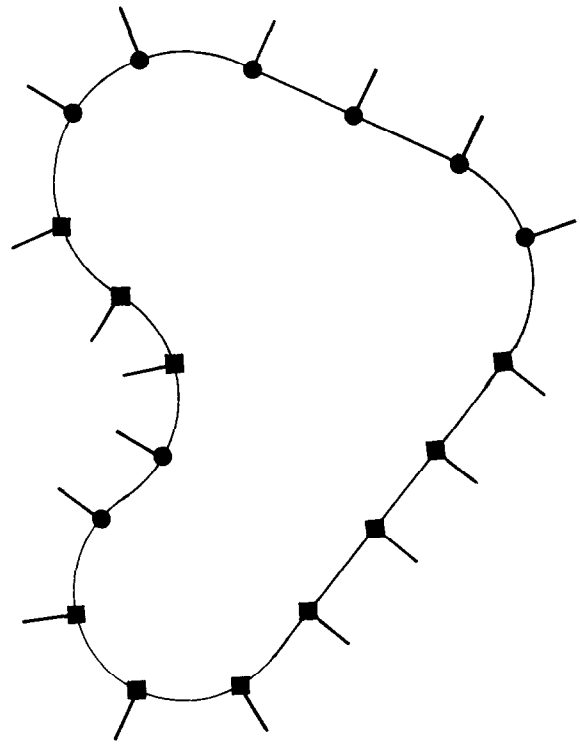

Figure 5. Points on the surface representing the outer side of the surface are circles. Points representing the inner side of the surface are squares. This classification is based on whether the normal vector points towards or away from the observer located at positive z (top of diagram)

In the simplest case, the highest voxel of each linked list is chosen as the pixel for that xy position of the picture. If there is a clipping plane (Figure 6(a)), only the part of each linked list below the plane is considered. The nearest voxel of each sublist is chosen. If it is desired to make the object appear solid, all the pixels with an inner surface colour are given a constant shade.

The situation for two clipped, interpenetrating objects is more complicated (Figure 6(b)). If the objects are hollow, then the nearest voxel beneath the clipping plane is chosen, as before. If the objects are solid, a two-step procedure is followed. First, a voxel is chosen for each object: the voxel nearest the observer of those underneath the clipping plane. Second, a pixel is created, based on this pair of individual object voxels. If both voxels are outer surface, the one nearer the observer is selected. If one voxel is outer surface, and the other inner, the inner surface voxel is selected and given a constant shade. If both voxels are inner surface, then a special overlap colour with a constant shade is used. Three or more interpenetrating objects are handled similarly.

Translucent surfaces are generated by treating half of the voxels of the surface as opaque and half as transparent. This is done in a chess-board pattern too fine to be seen without zooming by pixel replication.

The computer program implementing the 3D depth buffer algorithm is named MCS (molecular cross-section). It is written in Fortran 77 and is run on a VAX-11/750 computer operating under VMS. It uses 10 000–50 000 512-byte blocks of disc storage for scratch files and 1–30 hours of CPU time per run. The

computation time and disc storage space are functions of the area of the surfaces of the objects.

The execution of the MCS program is controlled by a command file. The commands are listed in Table 1. The main command is the FILL command, which creates the voxels in the depth buffer. The WINDOW, DEPTH, LIGHT, CENTRE, MATRIX and TRANSLATE commands may occur only before the FILL command. The COLOUR, PLANE, CUT, HOLLOW and SOLID commands may occur either before or after the FILL command. The WRITE command may occur only after the FILL command. Multiple pictures may be written from a depth buffer, by changing the clipping planes, the solid/hollow attribute of the object, and the colours of the object faces, between each picture write operation. Thus the amount of computation time, per image, is typically much less than the maximum of thirty hours.

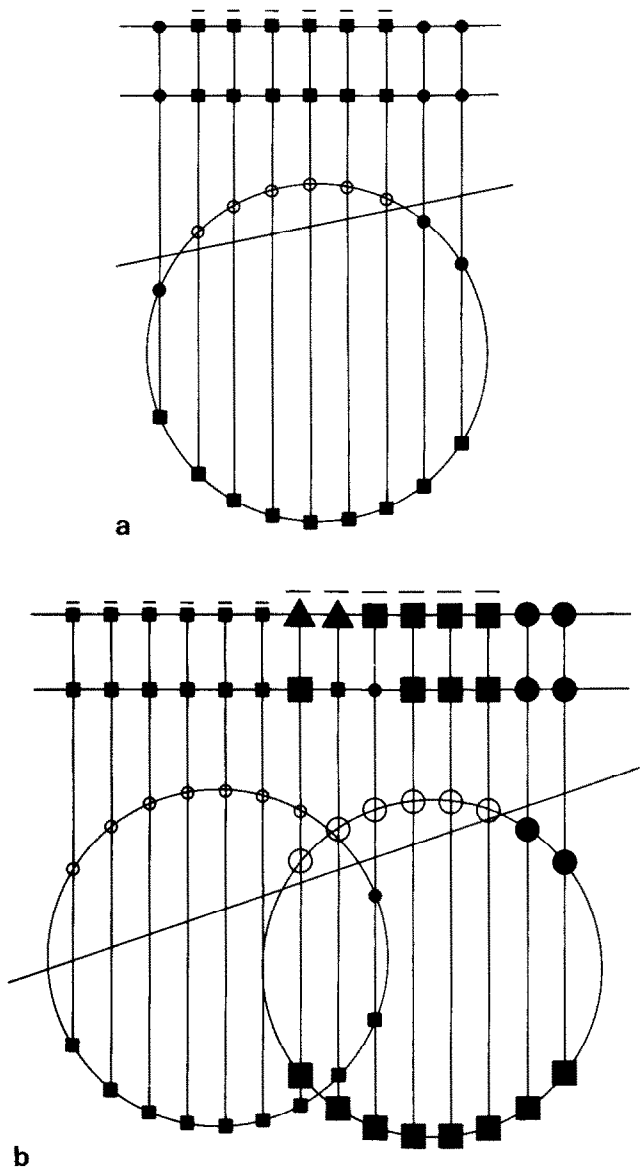

Figure 6. Read-out of depth buffer into pictures. The voxels on a ray are stored as a linked list. Circles: voxels seen as outer side of the surface; squares: voxels seen as inner side of the surface; slanted line: clipping plane; open circles: clipped voxels. Two pictures may be created, one with a solid interior (top line), the other with a hollow interior (line below it). Interior pixels in a solid-interior image have a constant shade (over-bar), independent of the angle of the surface normal with the light source. (a) One surface. (b) Two interpenetrating surfaces. Larger circles and squares: second surface. Triangles: solid overlap volume pixels.

Table 1. Commands controlling the MCS program

CENTRE	specify rotation centre of object
COLOUR	colour faces of object
CUT	clip object by cutting plane
DEPTH	degree of depth cueing
FILL	fill depth buffer with voxels
GET	read an object input file
HOLLOW	make object interior appear hollow
LIGHT	specify light source direction
MATRIX	specify rotation matrix of object
PLANE	define clipping plane
READ	read commands from file
SOLID	make object interior appear solid
STAT	write statistical information
STOP	stop the program
TRANSLATE	translate the object
WINDOW	define a window
WRITE	write a picture file to disc

A simple example of a command file is as follows:

```
WINDOW -20.0 20.0 -15.0 15.0 -100.0 100.0
! read solvent-accessible surface of lysozyme
GET SURF LYZ.GEO BINARY
CENTRE SURF 10.0 -15.0 8.0
! rotate the surface 90 degrees
MATRIX SURF 0.0 1.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 1.0
HOLLOW SURF
! the outer side of the surface will be colour 3,
! the inner side colour 5
COLOUR 3 5 SURF
! the light source is over the left shoulder
LIGHT -1.0 1.0 1.0
! no depth cueing
DEPTH 0.0
FILL SURF
WRITE FIRST.FRM
! clip the front by a plane at plus ten in z
PLANE 1 10.0
CUT 1 SURF
WRITE CUT.FRM
SOLID SURF
WRITE SOLID.FRM
! add ball-and-stick model
GET MODEL BAS.GEO FORMATTED
CENTRE MODEL 10.0 -15.0 8.0
MATRIX MODEL 0.0 1.0 0.0 -1.0 0.0 0.0 0.0 0.0 0.0 1.0
HOLLOW BAS
HOLLOW SURF
! read a file of face-colouring commands for the model
READ BAS.TXT
FILL MODEL
WRITE MODEL1.FRM
CUT 1 MODEL
WRITE MODEL2.FRM
STAT
STOP
```

RESULTS

A number of images produced by the 2D depth buffer algorithm have been published[6,14]. Since then, depth cueing has been added to the algorithm. When the depth cueing has been turned up to a maximum (100%), the surface topography of Cu, Zn superoxide dismutase[15] is strongly defined (Colour plate 1). Typically, only 20–30% depth cueing is used, because full depth cueing has the disadvantage that the atoms no longer appear rounded.

Colour plates 2 and 3 were produced from a single run of MCS. The interdigitation of side chains at this rigid haemoglobin[16] interface is apparent. The fit at the allosteric interface (Colour plate 4) is not as good. This is what one would expect, since these two subunits must move relative to one another when oxygen is bound and released.

The colour raster display of the trypsin–trypsin inhibitor[17] interface (Colour plate 5) has the advantage over the colour vector display of the interface[5] that the whole of both proteins may be shown. The amount of surface that can be dynamically displayed on a vector graphics system, such as the Evans and Sutherland Multi Picture System, is limited to about 15 000 dots, but a raster device can display several hundred thousand pixels.

Overlapping volumes are prominent in Colour plate 6, because some of the water molecules are occupying holes in trypsin[18] too small for a 1.5 Angstrom radius probe sphere. This is possible because hydrogen bonding brings the protein and water atoms closer together than the sum of their van der Waals radii.

Both the solvent-accessible surface and the chemical structure of the active site of lysozyme[19] can be seen in Colour plate 7, because the surface has been made translucent. Here the chemical structure is represented as a ball-and-stick model constructed from spheres and cylinders. Another, more abstract representation of chemical structure is shown (Colour plate 8) for the concanavalin A dimer[20]. A tube following the alpha carbon chain is constructed from cylinders and pieces of tori and resembles a wire-bender model[21].

The MCS program may be used to illustrate a vector field. This is shown in Colour plate 9 for the surface normal vectors of a small molecule. The arrows are constructed from cones and cylinders.

The van der Waals surface can be divided into two parts, the area that is accessible to a probe sphere, called the contact surface[22], and the inaccessible area buried in crevices between atoms. These two parts are coloured differently in the picture of the crambin protein[23] shown in Colour plate 10. In solvent-accessible surface images, the inaccessible van der Waals surface is replaced by the re-entrant surface. It is difficult to create an object representing just the inaccessible van der Waals surface, so an alternative procedure was used to create Colour plate 10. Two objects were created, one representing the entire van der Waals surface, and the other representing just the contact surface. The contact object was made with atoms one-tenth of an Angstrom larger in radius, so that it would cover the contact surface part of the van der Waals object and leave visible only the inaccessible surface part of the van der Waals object.

A variety of ancillary programs were used to make the input files to MCS for these pictures. One program constructed a sphere and cylinder input file for the ball-and-stick model, from atomic coordinate and connectivity files. Another program constructed a cylinder and torus input file for the wire-bender model, from alpha carbon coordinates and virtual connectivity. The output of the analytical molecular surface (AMS) program was processed by two different programs to produce MCS input files for the solvent-accessible surface and the van der Waals surface. The input file for the arrows was constructed from a list of surface points and normal vectors.

FUTURE DIRECTIONS

The addition of a feature for displaying text would add to the ease of interpretation of the images. Atoms and amino acid residues could then be labelled. This could be done either by an ancillary program which would convert characters into input objects for MCS, or by adding new subroutines to MCS for creating voxels from characters directly.

There is much room for improvement in the efficiency of the algorithms. For example, solvent-accessible surfaces could be handled by intersecting them with a stack of closely-spaced planes parallel to the *xz* plane. This would generate a set of contours that could be converted into pixels or voxels.

The production of raster images would be much more convenient if the user, instead of preparing the input files by hand, could simply type a command into a vector graphics system molecular modelling program that would create the files required to produce the raster equivalent of the current image on the vector display. Ideally, a single molecular modelling program would create both vector and raster images.

REFERENCES

1 **Porter, T** *Comput. Graphics* Vol 13 (1979) pp 234–236
2 **Max, N L** *Comput. Graphics* Vol 13 (1979) pp 165–173
3 **Lesk, A M and Hardman, K D** *Science* Vol 216 (1982) pp 539–540
4 **Lesk, A M and Hardman, K D** *Methods in enzymology* in press
5 **Langridge, R, Ferrin, T E, Kuntz, I D and Connolly, M L** *Science* Vol 211 (1981) pp 661–666
6 **Connolly, M L** *Science* Vol 221 (1983) pp 709–713
7 **Bash, P A, Pattabiraman, N, Huang, C, Ferrin, T E and Langridge, R** *Science* Vol 222 (1983) pp 1325–1327
8 **O'Donnell, T J and Olson, A J** *Comput. Graphics* Vol 15 (1981) pp 133–142
9 **Connolly, M L and Olson, A J** submitted to *Comput. Chem.*
10 **Foley, J D and Van Dam, A** *Fundamentals of interactive computer graphics* Addison-Wesley, USA (1982) pp 560–561
11 **Atherton, P R** *Comput. Graphics* Vol 15 (1981) pp 279–287
12 **Lee, B and Richards, F M** *J. Mol. Biol.* Vol 55 (1971) pp 379–400

13 **Richards, F M** *Ann. Rev. Biophys. Bioeng.* Vol 6 (1977) pp 151–176
14 **Connolly, M L** *J. Appl. Cryst.* Vol 16 (1983) pp 548–558
15 **Tainer, J A, Getzoff, E D, Beem, K M, Richardson, J S and Richardson, D C** *J. Mol. Biol.* Vol 160 (1982) pp 181–217
16 **Ladner, R C, Heidner, E J and Perutz, M F** *J. Mol. Biol.* Vol 114 (1977) pp 385–414
17 **Huber, R, Kukla, D, Bode, W, Schwager, P, Bartels, K, Deisenhofer, J and Steigemann, W** *J. Mol. Biol.* Vol 89 (1974) pp 73–101
18 **Chambers, J L and Stroud, R M** *Acta Cryst.* Vol B35 (1979) pp 1861–1874
19 **Diamond, R** *J. Mol. Biol.* Vol 82 (1974) pp 371–391
20 **Reeke, G N, Jr, Becker, J W and Edelman, G M** *J. Biol. Chem.* Vol 250 (1975) pp 1525–1547
21 **Rubin, B and Richardson, J S** *Biopolymers* Vol 11 (1972) pp 2381–2385
22 **Richmond, T J and Richards, F M** *J. Mol. Biol.* Vol 119 (1978) pp 537–555
23 **Hendrickson, W A and Teeter, M M** *Nature* Vol 290 (1981) pp 107–113