A general approach to surface modelling applied to molecular graphics

Peter Quarendon

IBM UK Science Centre, Athelstan House, St Clement Street, Winchester, Hampshire SO23 9DR, UK

A program is described that produces realistic representations of surfaces on a raster graphics terminal. The program uses a spatial subdivision algorithm similar to that of Woodwark and Quinlan. Simple geometric primitives, such as spheres, cones, cylinders and cuboids, are combined to build more complex pictures. This paper describes the program and how it is applied to molecular graphics. Examples of several applications are given.

Keywords: surface modelling, raster graphics, subdivision techniques

received 1 December 1983

Programs that produce realistic representations of molecular models have been in use for some time. They have generally been used for producing space-filling models of molecules[1-3], but schematic diagrams of proteins are also generated[4]. Generally, space-filling pictures have been composed from spheres and other shapes have been approximated by using many planar facets. Blinn[5] has recently departed from this and rastered directly from the algebraic definition of the surface.

Researchers outside the field of molecular graphics who are interested in solid modelling have developed other approaches to surface generation. For two reviews of these systems see Baer *et al*[6] and Requicha[7]. One method of defining the object of interest is to specify its volume by carrying out Boolean operations on geometric primitives. Using this system primitives may be logically UNIONed, combining their volumes, or logically DIFFERENCEd to remove one volume from another. This Boolean strategy is that used in the program currently being used at IBM UK Scientific Centre.

A major problem in displaying solids is the elimination of hidden surfaces. There are three main ways to achieve this. One is to draw the picture from the back so that objects nearer the viewer are drawn last and will overlay those farther away. A common method, used for example by Lesk and Hardman[4], is to store the depth with each pixel in a buffer called a *z*-buffer. Then, as each pixel is written, its depth can be compared with that of any pixel already at that screen position.

The present program constructs the picture from front to back. This has the advantage that obscured objects need not be processed at all, so that the execution time depends more on the visual complexity of the picture than on the total number of objects that it contains. Woodwark and Quinlan show that this algorithm exhibits, in practice, an approximately linear dependence on the number of objects. This compares with other algorithms which can depend on the fourth power of the number of objects[8]. Solid modelling systems for mechanical design expect models which consist of at most some hundreds of primitive parts. Models of proteins tend to be more complex and may contain many thousands of primitives, so that this aspect of the performance is comparatively important.

However, with a front-to-back processing order, anti-aliasing is somewhat more difficult to perform and has not been included in the program.

PROGRAM INPUT

The input syntax for the program is shown in Table 1. Literals in the syntax are shown in upper case (although they can be in upper or lower case in program input). Lower-case words are meta-syntactic variables and are either defined in the syntax or should be replaced by numbers. Alternatives have been placed one under the other and surrounded by braces. Two samples conforming to the syntax are given in Figures 1 and 2.

The simplest primitive is a half-space with a planar boundary. Every point on one side of the plane is considered to be inside the object and every point on the other side is considered to be outside. Points on the boundary have to be treated very carefully to obtain correct results in all cases (see Requicha[6]), but the program simply considers these surface points as being inside the object. Using conventional vector notation, if the boundary of planar half-space is given by the equation $\mathbf{a.x} = 0$, the half-space is defined by the set of points:

$$\text{PLANE}(a) = \{\mathbf{x} | \mathbf{a.x} \langle= 0\}$$

Six planar half-spaces can be used to construct a cube. For convenience, this is provided as an additional primitive.

Table 1. Input syntax

```
object =  ⎧ primitive-object                          ⎫
          ⎪        ⎧UNION⎫                            ⎪
          ⎨ object ⎨INT  ⎬ primitive-object           ⎬
          ⎪        ⎩DIFF ⎭                            ⎪
          ⎪ transformed-object                        ⎪
          ⎩ coloured-object                           ⎭

primitive object: = ⎧ PLANE (normal-vector)                       ⎫
                    ⎪ CUBOID (diagonal-vector)                    ⎪
                    ⎪ SPHERE (radius)                             ⎪
                    ⎨ CYLINDER (radius, axial-vector)             ⎬
                    ⎪ CONE (radius1, radius2, axial-vector)       ⎪
                    ⎪ TORUS (radius1, radius2, normal-vector)     ⎪
                    ⎩ (object)                                    ⎭

transformed-object: = object ⎧ AT (displacement)      ⎫
                             ⎪ SCALE (scale-factor)   ⎪
                             ⎨ XROT (rotation)        ⎬
                             ⎪ YROT (rotation)        ⎪
                             ⎪ ZROT (rotation)        ⎪
                             ⎩ ROT (rotation-matrix)  ⎭

coloured object: = object ⎧COLOUR (colour)⎫
                          ⎩GLOSS (gloss)  ⎭

normal-vector
diagonal-vector: = x-distance, y-distance, z-distance
axial-vector
displacement
```

```
(TORUS(0.12,0.05,1,0,-1) AT (0.88,0.29,0.27) COLOUR(4)
    GLOSS(0.8))
UNION (SPHERE(0.2) AT (0.25,0.3,0.75)  COLOUR(7)
    GLOSS(0.4))
UNION  (CYLINDER(0.17,0,0.50,0)  AT  (0.60,0.1,0.50)
    COLOUR(5) GLOSS(0.2))
UNION (CUBOID(0.18,0.15,0.15) YROT(20)AT)(0.05,0.1,0.25)
    COLOUR(6))
UNION (CONE(0.1,0.0,0.17,-0.08,-0.03) YROT(20) AT
    (0.34,0.2,0.15) COLOUR(3) GLOSS(0.6))
UNION (PLANE(0,1,0) AT (0,0.1,0) COLOUR (1))
```

Figure 1. Program input for Colour plate 1

```
(cpk part)
DRAW ( SPHERE(1.300) AT( 2.879 , 5.481 , 13.579 )
    COLOUR (2))
UNION ( SPHERE(1.500) AT( 2.508 , 4.313 , 13.479 )
    COLOUR (7))
UNION ( SPHERE(1.300) AT( 0.013 , 1.638 , 6.149 )
    COLOUR (2))
(re-entrant toroids)
UNION((CONE(0.275,0.320,0.900,-1.393,-5.340)
    AT(2.870,5.481,13.579))
DIFF (TORUS(0.550945717,1.3,0.900,-1.393,-5.340)
    AT(3.279,4.848,11.152)))
UNION ((CONE(1.483,1.608,1.068,0.293,-1.434)
    AT(2.702,3.795,9.673))
DIFF (TORUS(2.76870194,1.3,1.068,0.293,-1.434)
    AT(2.948,3.863,9.343)))
UNION ((CONE(0.889,0.829,0.168,1.686,3.906)
    AT(2.702,3.795,9.763))
DIFF (TORUS(1.65891533,1.3,0.168,1.686,3.906)
    AT(2.791,4.688,11.742)))
(re-entrant spheres)
UNION (PLANE( 1.955, 0.037, 1.856) AT(11.939,0,0)
INT  PLANE(-0.516,-1.064, 2.421) AT(0,0, 9.147)
INT  PLANE(-0.494,-0.133,-2.451) AT(0,0,11.105)
DIFF( SPHERE( 1.300) AT( 2.114, 2.191, 10.560))
INT  PLANE(0.988, 2.014, 1.496) AT(0,13.789,0)
INT  PLANE(-1.480, 0.905, 2.062) AT(0,0,-12.649)
INT  PLANE(-0.066,-1.007,-2.292) AT(0,0,12.636)
DIFF( SPHERE( 1.300) AT( -0.016, 6.543, 9.767)))) COLOUR(5)
```

Figure 2. Section of program input for Colour plate 6

In a similar way, a spherical volume, radius r, centred at the origin is defined by the set of points:

$$\mathrm{SPHERE}(r) = \{\mathbf{x}|\mathbf{x}.\mathbf{x} <= r^2\}$$

An infinite cylinder whose axis passes through the origin is defined by its radius r and a vector $\mathbf{u}$ which gives the direction of the axis:

$$\text{INFINITE-CYLINDER}(r, \mathbf{u}) = \{\mathbf{x}|\mathbf{x}.\mathbf{x} - (\mathbf{x}.\mathbf{u})^2 <= r^2\}$$

The actual program input provides a finite cylinder where the vector $\mathbf{u}$ specifies not only the direction but also the length of the cylinder. Two planar half-spaces are used to truncate the infinite cylinder to the correct length.

An infinite cone with half-angle α has a similar definition:

$$\text{INFINITE-CONE}(\alpha, \mathbf{u}) = \{\mathbf{x}|\mathbf{x}.\mathbf{x} - (\mathbf{x}.\,\mathbf{u})^2 <= (\mathbf{x}.\mathbf{u})^2 \tan^2\alpha\}$$

Again, the program input provides a truncated cone:

$$\mathrm{CONE}(r1,r2,\mathbf{u})$$

where $r1$ and $r2$ give the radii at the two ends, and $\mathbf{u}$ gives not only the direction of the axis, but also the length of the cone. This is again converted internally into an infinite cone and two planar half-spaces.

Lastly, a torus is defined by:

$$\mathrm{TORUS}(r1,r2,\mathbf{u}) = \{\mathbf{x}|(r1 - \mathrm{sqrt}(\mathbf{x}.\mathbf{x} - (\mathbf{u}.\mathbf{x})^2))^2 <= r2^2 - (\mathbf{u}.\mathbf{x})^2\}$$

Colour plate 1 (see p C1) shows examples of the primitives. The plate was generated by the program input shown in Figure 1.

Objects can be combined by set union, intersection and difference operations. These have their usual meanings. Colour plate 2 shows some simple compound objects. The object on the lower right is the union of two blocks and a cylinder, that on the lower left is a rectangular block with a cylindrical hole. Objects can also be transformed by translation, uniform scaling, rotation about any of the axes or a general rotation matrix.

PICTURE CONSTRUCTION

The picture-drawing program is in two parts. The first part reads the input file and checks for errors. It then performs any transformations specified and builds an internal data structure to represent the description of the picture as an expression in constructive solid geometry.

This structure is passed to the second phase of the program which constructs the picture as a large (up to 1024 × 1024) array of pixel values. This image is generated by the method of 'spatial subdivision' similar to that of Woodwark and Quinlan[9].

A cubic volume is considered which contains all the objects to be drawn. These objects will be described by a single constructive solid geometry expression, such as that in Figure 1. The expression is examined to see whether the volume is empty. On the assumption that it is not empty, it is then subdivided into eight smaller cubes of equal size.

Each subcube is then examined in turn. Any subcubes which are empty are discarded as are any which are totally within an object. The remaining cubes contain one or more surfaces and each of these is again divided into eight smaller cubes and the process is repeated, discarding empty and full cubes. The subdivision continues until the cubes are sufficiently small to correspond to pixels in the resulting image. As each cube contains a part of a surface, the result is an approximation of the surface of the required objects.

At each stage in the subdivision, the expression describing the objects in a particular volume of space is simplified as far as possible. Each primitive object is tested to see whether it intersects the volume, is completely outside or completely encompasses the volume. Objects which intersect the volume are left unchanged, but those which do not are replaced by 'empty', if they are separate from the volume, or 'full', if they enclose the volume. The constructive solid geometry expression can then be simplified using the equivalent of the rules of Boolean logic, eg:

a UNION empty $\Rightarrow$a
a UNION full $\Rightarrow$full
b INT full $\Rightarrow$b
b INT empty $\Rightarrow$empty

If the result of applying this simplification to the expression describing the contents of a volume is 'empty', then the volume is entirely empty and can be discarded. Otherwise, the simplified expression must be carried on to the next stage of the subdivision.

For many types of object it is possible to test directly whether it intersects a cubic volume[10]. However, the procedure is somewhat lengthy and so instead the test is made against a circumscribing spherical volume. This is simpler to program but allows some primitives to be kept which should be eliminated. The error involved diminishes as the subdivision proceeds and cannot be detected in the resulting pictures.

The process can be optimized when a large volume is found to contain only a single object which can be drawn directly. Because of their common occurrence, this optimization is carried out only for planes and spheres. More extensive optimizations of the same basic procedure which are applicable when the objects are simple planar half-spaces are discussed by Woodwark and Quinlan[8].

The above algorithm would generate all surface volumes whether or not they are visible from the position of the viewer. By convention in the program, the viewer is assumed to be looking down the z-axis from an infinite distance. Thus cubic volumes with smaller z values will obscure those with the same values of x and y but higher values of z. To achieve hidden surface elimination, at each stage in the subdivision, volumes are processed from front to back, ie from low to high z. A map is kept (in the form of a quad tree[11]) showing which areas of the screen have been shaded. Once an area has been completely shaded, volumes behind are not processed and so do not contribute to the resulting image.

As a visible surface pixel volume is generated, the colour and intensity are calculated at its centre. For a pixel which is on the surface of only one object, the surface normal at the point is determined. The intensity can then be calculated from the illumination model described below, the colour being the colour of the object. Pixel volumes which lie on the intersection of two or more objects will contain more than one surface. In these cases it is necessary to determine which surface to display. This is done by casting a ray through the pixel centre, parallel to the z-axis. The intersection of each object in the volume with this ray is then found and hence the section of the ray which is inside the object. The sections of ray from the different objects are then compared so that the surface nearest the viewer is determined. The procedure is described in more detail by Roth[12].

ILLUMINATION

The reflected intensity at a point on an object surface is considered to be made up of three parts comprising ambient, diffuse and specular components.

Ambient illumination is assumed to come equally from all directions, so the reflected intensity is constant, independent of the surface orientation. Increasing the amount of ambient illumination decreases the contrast in the picture.

The diffuse component derives from light from a point source which hits the surface from a certain angle and is subsequently scattered equally in all directions. The intensity depends on the angle the surface makes with the direction of the light source but not on the position of the viewer. This component simulates very matt surfaces.

The specular component derives from light which starts from the point light source and is reflected directly from the surface to the viewer's eye. This is greatest when the angle of incidence at the surface equals the angle of reflection and falls off sharply when this condition is not met.

Phong[13] approximated the total reflected intensity by the expression:

$$I_n = p_a + p_d(\mathbf{N}.\mathbf{L}) + p_s(\mathbf{N}.\mathbf{H})^n$$

where **L** is the direction of the light source, **N** is the normal to the surface and **H** is a vector half-way between the direction of the light and that of the eye. This is shown in Figure 3. In the expression, n is a shaping constant of the order of 60.

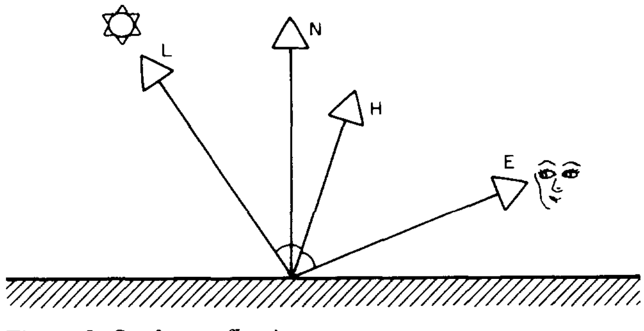

Figure 3. Surface reflection

The constants p_a, p_d and p_s vary the proportions of the three components and can be used to vary the appearance of the surface. This formulation is used in the present program, except that the specular term $(\mathbf{N}.\mathbf{H})^n$ is replaced by an expression derived from Blinn[14] which is slightly easier to compute:

$$(\mathbf{N}.\mathbf{E})\left\{\frac{K+1}{K+(\mathbf{N}.\mathbf{H})^2}\right\}^2$$

where K is a variable shaping constant.

APPLICATIONS

The overall structure is shown in Figure 4. At present, four basic molecular representations can be produced. They are:

- space-filling
- ball and stick
- protein secondary structure
- solvent-accessible surface

The source data for the molecular structure are held on a relational database[15]. The main relation for a protein molecule contains an entry for each atom, giving the atomic coordinates, the protein residue of which the atom is part and the atom type within the residue.

To produce a space-filling model, only spherical primitives are required. An auxiliary relation is kept in the database which gives the colour and atomic radius to be assumed for each atom type. The application program extracts the atomic coordinates together with this additional data and generates an input file consisting of sphere definitions combined by union operations. Colour plate 3 shows a space-filling representation of the insulin hexamer, looking down the z-axis. (The insulin coordinates used to generate these pictures were taken from the model of bovine insulin, built by A J Morffew from the porcine insulin of Isaacs and Agarwal[16].) The user interface allows the researcher to select a range of residues which can then be coloured

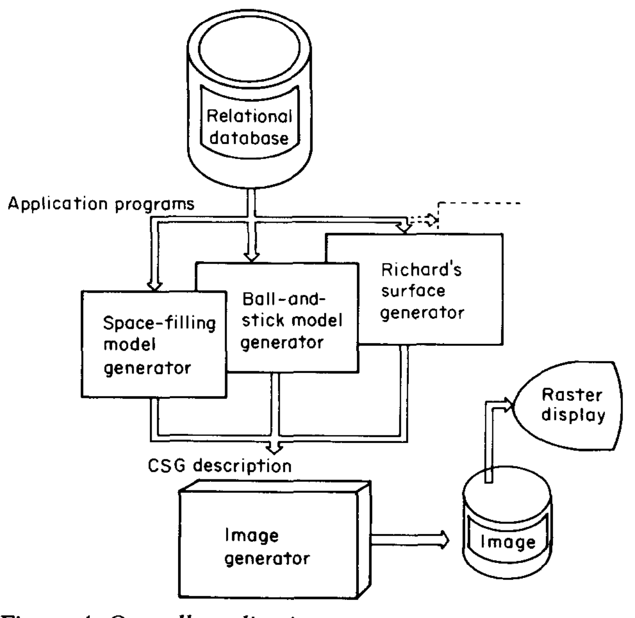

Figure 4. Overall application structure

either in a functional system or in contrasting colours. Here the standard functional system has been used with nitrogen atoms blue, oxygen atoms red, carbon atoms white and sulphur yellow. The insulin hexamer contains two zinc atoms which have been made to look metallic by increasing the specular reflection.

Colour plate 4 shows a secondary structure picture of lysozyme. The α-helices are shown as cylinders as in the Lesk and Hardman program[4]. As yet, we have not implemented a representation of the β-sheet. In principle this could be done by approximating the curves by planar sections, but work is in progress to add spline-like shapes as primitive objects. The example also includes some atoms in the space-filling representation showing how these different entities can be combined to show features of interest.

Colour plate 5 also makes use of cylinders, in this case to construct a ball-and-stick representation of lysozyme. Each ball represents an amino acid residue and the balls are colour coded according to the properties of the amino acid. The hydrophilic residues are red and the hydrophobic residues are blue. Ball-and-stick plots can also be produced so that individual atoms, rather than entire residues, are represented by balls.

The solvent-accessible surface of a molecule was defined by Richards[17] and consists of two parts, the contact surface and the re-entrant surface. The contact surface is that part of the van der Waals surface which can be touched by a probe sphere representing a solvent molecule. The re-entrant surface is the surface swept out by the probe sphere when it is in contact with two or more atoms of the molecule. Connolly[18] has implemented an analytical algorithm generating the surface. The algorithm generates a series of spherical and toroidal surfaces bounded by circular arcs.

To generate the surface as the boundary of a solid model the application output must consist of primitive objects combined by set operations rather than a series of surface patches.

The result consists of three parts:

1. The CPK (Corey, Pauling, Koltun) model, in which a sphere represents each atom.
2. For each pair of atoms which are sufficiently close together the probe sphere generates a toroidal surface. This is constructed as shown in Figure 5(a) by subtracting a torus from a cone. The conical primitive is formed between the two circles of contact between the probe sphere and the two atoms. The torus to be subtracted is the volume swept out by the probe sphere when it is in contact with the two atoms.
3. For each triplet of atoms, again assuming they are sufficiently close together, a spherical surface is created where the probe sphere touches all three atoms. This is shown in Figure 5(b). The volume with this surface is created from an irregular tetrahedron with its vertices at the centres of the base atoms; the volume of the probe sphere is subtracted from this. Two such objects are needed for each triplet of atoms, one above as shown in the figure and one below.

The mathematics required to generate these objects is identical with that given by Connolly. Colour plate 6

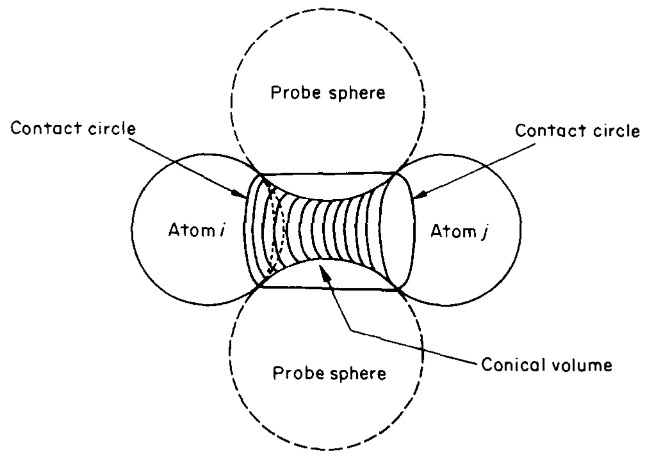

Figure 5(a). Re-entrant toroidal volume

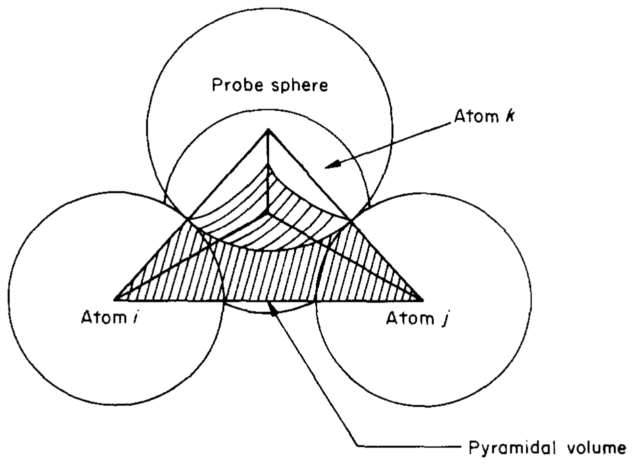

Figure 5(b). Re-entrant spherical volume

shows the Richards surface of a small tripeptide F-met-leu-phe[19]. Approximately 300 re-entrant toroidal volumes and 200 re-entrant spherical volumes in addition to the 30 atomic spheres were required. A short sequence from the application output file is shown in Figure 2. The atoms themselves are shown in standard functional colouring and the re-entrant surface is shown in light blue.

CONCLUSIONS

A general solid surface program is being used at IBM UK Science Centre for molecular graphics. The program uses the spatial subdivision algorithm and its input is a list of geometric primitives which are combined by Boolean operations. This approach allows the UK Science Centre researchers to combine representations to produce informative pictures. At this time we are developing the program so that more primitives can be specified, the code is optimized and a friendly menu-driven user interface is developed.

ACKNOWLEDGMENTS

I would like to thank Dr A J Morffew for his help in preparing this paper and for his assistance in writing the molecular model interfaces. Also my other colleagues at IBM UK Science Centre who have been involved in developing this program and its user interface. In particular, Stephen Todd has helped in many ways.

REFERENCES

1 **Max, N L** *Comput. Graph.* Vol 13 No 2 (1979) pp 165–173
2 **Porter, T K** *Comput. Graph.* Vol 13 No 2 (1979) pp 234–236
3 **Feldmann, R J, Bing, D H, Furie, B C and Furie, B** *Proc. Natl. Acad. Sci. USA* Vol 75 (1978) pp 5409–5412
4 **Lesk, A M and Hardman, K D** *Science* Vol 216 (1982) pp 539–540
5 **Blinn, J F** *ACM Trans. Graph.* Vol 1 No 3 (1982) pp 235–256
6 **Baer, A, Eastman, C and Henrion, M** *Comput.-Aided Des.* Vol 11 No 5 (1979) pp 253–272
7 **Requicha, A A G** *ACM Comput. Surv.* Vol 12 No 4 (1980) pp 437–464
8 **Woodwark, J R and Quinlan, K M** 'Reducing the effect of complexity on volume model evaluation' *Comput. Aided Des.* Vol 14 No 2 (1982) pp 89–95
9 **Woodwark, J R and Quinlan, K M** 'The derivation of graphics from volume models by recursive subdivision of the object space' *Proc. Computer Graphics 80 Conf.* (1980)
10 **Field, D A and Morgan, A** 'A quick method for testing if a second degree polynomial has a solution in a given box' *Proc. Computer Graphics 80 Conf.* (1980) in *General Motors Research Publication GMR 3656* (April 1981)
11 **Pavlidis, T** *Algorithms for graphics and image processing* Springer (1982) p 105ff
12 **Roth, S D** 'Ray casting for modeling solids' *Comput. Graph. Image Process.* Vol 18 (1982) pp 109–144
13 **Phong, B,-T** 'Illumination for computer generated images' *Commun. ACM* Vol 18 No 6 (June 1975) p 311
14 **Blinn, J F** 'Models of light reflection for computer synthesised images' *Comput. Graph.* Vol 11 No 2 (1977) pp 192–198
15 **Morffew, A J, Todd, S P J and Snelgrove, M J** *Comput. Chem.* Vol 7 No 1 (1983) pp 9–16
16 **Isaacs and Agarwal** *Acta Crystallogr.* Vol A34 (1978) pp 782–791
17 **Richards, F M** *Annu. Rev. Biophys. Bioeng.* Vol 6 (1977) pp 151–176
18 **Connolly, M L** 'Analytical molecular surface calculations' *J. Appl. Crystallogr.* Vol 16 (1983) pp 548–558
19 **Morffew, A J and Tickle I** *Cryst. Struct. Commun.* Vol 10 (1981) pp 781–788