previous frame needs to be deleted with each new frame unless it is known that each new frame will specify every pixel. However, the technology is advancing rapidly and these restrictions are already disappearing.

Vector machines, on the other hand, are specialized to drawing straight lines between specified points by driving the electron beam along such lines. No time is wasted on blank areas of the screen. Dots may be drawn with arbitrary coordinates, in any order, but areas, if they are to be filled, must be done with a ruling technique which is very seldom done. Images produced by vector machines are naturally transparent in that foreground does not obscure background, which makes them ideal for seeing into representations of molecular structure.

3.3.1.5.2. *Optimization of line drawings*

A line drawing consisting of n line segments may be specified by anything from $(n + 1)$ to $2n$ position vectors depending on whether the lines are end-to-end connected or independent. Appreciable gains in both processing time and storage requirements may be made in complicated drawings by arranging for line segments to be end-to-end connected as far as possible, and an algorithm for doing this is outlined below. For further details see Diamond (1984*a*).

Supposing that a list of nodal points (atoms if a covalent skeleton is being drawn) exists within a computer with each node appearing only once and that the line segments to be drawn between them are already determined, then at each node there are, generally, both forward and backward connections, forward connections being those to nodes further down the list. A quantity D is calculated at each node which is the number of forward connections minus the number of backward connections. At the commencement of drawing, the first connected node in the list must have a positive D, the last must have a negative D, the sum of all D values must be zero and the sum of the positive ones is the number of strokes required to draw the drawing, a 'stroke' being a sequence of end-to-end connected line segments drawn without interruption. The total number of position vectors required to specify the drawing is then the number of nodes plus the number of strokes plus the number of rings minus one.

Drawing should then be done by scanning the list of nodes from the top looking for a positive D (usually found at the first node), commencing a stroke at this node and decrementing its D value by 1. This stroke is continued from node to node using the specified connections until a negative D is encountered, at which point the stroke is terminated and the D value at the terminating node is incremented by 1. This is done even though this terminating node may also possess some forward connections, as the total number of strokes required is not minimized by keeping a stroke going as far as possible, but by terminating a stroke as soon as it reaches a node at which *some* stroke is bound to terminate.

The next stroke is initiated by resuming the scan for positive D values at the point in the node list where the previous stroke began. If this scan encounters a zero D value at a node which has not hitherto been drawn to, or drawn from, then the node concerned is isolated and not connected to any other, and such nodes may require to be drawn with some special symbol. The expression already given for the number of vectors required is valid in the presence of isolated nodes if drawing an isolated node is allowed one position vector, this vector not being counted as a stroke.

The number of strokes generated by this algorithm is sensitive to the order in which the nodes are listed, but if this resembles a natural order then the number of strokes generated is usually close to the minimum, which is half the number of nodes having an odd number of connections. For example, the letter E has six nodes, four of which have an odd number of connections, so it may be drawn with two strokes.

3.3.1.5.3. *Representation of surfaces by lines*

The commonest means of representing surfaces, especially contour surfaces, is to consider evenly spaced serial sections and to perform two-dimensional contouring on each section. Repeating this on serial sections in two other orientations then provides a good representation of the surface in three dimensions when all such contours are displayed. The density is normally cited on a grid with submultiples of **a**, **b** and **c** as grid vectors, inverse linear interpolation being used between adjacent grid points to locate points on the contour. For vector-graphics applications it is expedient to connect such points with straight lines; some equipment may be capable of connecting them with splines though this is burdensome or impossible if real-time rotation of the scene is required. Precalculation of splines stored as short vectors is always possible if the proliferation of vectors is acceptable. For efficient drawing it is necessary for the line segments of a contour to be end-to-end connected, which means that it is necessary to contour by following contours wherever they go and not by scanning the grid. Algorithms which function in this way have been given by Heap & Pink (1969) and Diamond (1982*a*). Contouring by grid scanning followed by line connection by the methods of the previous section would be possible but less efficient. Further contouring methods are described by Sutcliffe (1980) and Cockrell (1983).

For raster-graphics devices there is little disadvantage in using curved contours though many raster devices now have vectorizing hardware for loading a line of pixels given only the end points. For these devices well shaped contours may be computed readily, using only linear arithmetic and a grid-scanning approach (Gossling, 1967). Others have colour-coded each pixel according to the density, which provides a contoured visual impression without performing contouring (Hubbard, 1983).

3.3.1.5.4. *Representation of surfaces by dots*

Connolly (Langridge *et al*., 1981; Connolly, 1983*a*,*b*) represents surfaces by placing dots on the surface with an approximately uniform superficial density. Connolly's algorithm was developed to display solvent-accessible surfaces of macromolecules and provides for curved concave portions where surface atoms meet. Pearl & Honegger (1983) have developed a similar algorithm, based on a grid, which generates only convex portions which meet in cusps, but is faster to compute than the Connolly surface. Bash *et al.* (1983) have produced a van der Waals surface algorithm fast enough to permit real-time changes to the structure without tearing the surface.

It has become customary to use a dot representation to display computed surfaces, such as the surface at a van der Waals radius from atomic centres, and to use lines to represent experimentally determined surfaces, especially density contours.

3.3.1.5.5. *Representation of surfaces by shading*

Many techniques have been developed, mainly for raster-graphics devices, for representing molecular surfaces and these have been very well reviewed by Max (1984).

The simplest technique in this class consists in representing each atom by a uniform disc, or high polygon, which can be colour-coded and area-filled by the firmware of the device. If such atoms are sorted on their z coordinate and drawn in order, furthest ones first, so that nearer ones partly or completely overwrite the further ones then the result is a simple representation of the molecule as seen from the front. This technique is fast and has its uses when a rapid schematic is all that is required. In one sense it is wasteful to process distant atoms when they are going to be overwritten by foreground atoms, but front-to-back processing requires the boundaries of visible parts

of partially obscured atoms near the front to be determined before they can be painted or, alternatively, every pixel must be tested before loading to see if it is already loaded. Not only does this approach give a uniform rendering over the whole area of one atom, it also gives a boundary between overlapping atoms with almost equal z values which completes the circle of the nearer atom, though it should be an arc of an ellipse when the atoms are drawn with radii exceeding their covalent radii.

Greater realism is achieved by establishing a z buffer, which is an additional area of memory with one word per pixel, in which is stored the z value of the currently loaded feature in each pixel. Treatments which take account of the sphericity are then possible and correct arcs of intersection for interpenetrating spheres and more complicated entities arise naturally through loading a colour value into a pixel only if the z coordinate is less than that of the currently loaded value. This z buffer and the associated x, y coordinates should be in picture space or screen space rather than display space since only after the application of perspective can points with the same x/w and y/w coordinates obscure one another.

It is usual in such systems to vary the intensity of colour within one atom by darkening it towards the circumference on the basis of the z coordinate. Some systems augment this impression of sphericity by highlighting. The simplest form of highlighting is an extension of the uniform disc treatment in which additional, brighter discs, possibly off centre, are associated with each atom. More general highlighting (Phong, 1975) is computed from four unit vectors, these being the normal to the surface, the direction to a light source, the direction to the viewer and the normalized vector sum of these last two. Intensity levels may then be set as the sum of three terms: a constant, a term proportional to the scalar product of the first two vectors (if positive) and a term proportional to a high power of the scalar product of the first and last vectors; the higher the power the glossier the surface appears to be. This final term normally adds a white term, rather than the surface colour, supposing the light source to be white.

Shadows may also be rendered to give even greater realism. In addition to the z buffer and (x, y) frame buffer a second z buffer for z' values associated with x' and y' is also required. These coordinates are then related by $x' = x + \alpha z$, $y' = y + \beta z$, $z' = z$. The second buffer is a ray buffer since $x'y'$ are the coordinates with which an illuminating ray passing through (xyz) passes through the $z = 0$ plane, and z', stored at x', y', records the depth at which this ray encounters material. Thus any two pixels $(x_1y_1z_1)$ and $(x_2y_2z_2)$ are on the same illuminating ray if their x' and y' values are equal and the one with smaller z' shadows the other. Processing a pixel at $(x_1y_1z_1)$ therefore involves first determining its visibility on the basis of the z buffer, as before, then, whether or not it is visible, setting $z'_1 = z_1$ and considering the value of z' currently stored at $x'y'$, which we call z'_2.

If $z'_1 < z'_2$ then $x_1y_1z_1$ is in light and must be loaded accordingly. From z'_2 we find the previously processed pixel $(x_2y_2z_2)$ which is now in shade and which was in light when originally processed, so that the colour value stored at x_2y_2 needs to be altered *unless* the pixel at x_2y_2 is now $(x_2y_2z_3)$ with $z_3 < z_2$, in which case the pixel $(x_2y_2z_2)$ which has now become shadowed by $(x_1y_1z_1)$ has, in the meantime, been obscured by $(x_2y_2z_3)$ which is not shadowed by $(x_1y_1z_1)$ and no change is therefore needed. In either event z'_1 then replaces z'_2.

If $z'_1 > z'_2$ then $(x_1y_1z_1)$, if visible, is in shade and must be coloured accordingly, and in this case z'_2 is not superseded.

This shadowing scheme corresponds to illumination by a light source at infinity in picture space or, equivalently, with a z coordinate equal to that of the eye in display space. For its implementation x, y and z may be in any convenient coordinate system, *e.g.* pixel addresses, but if x and y are expressed with the range -1 to 1 and z with the range 0 to 1 corresponding to the window then they may be identified as the quantities $x/w, y/w$ and z/w of picture space (Section 3.3.1.3.1).

If, in the notation of Section 3.3.1.3.5, the light source is placed at (P, Q, E, V) in display space and a ray leaves it in the direction (p, q, r, V) then

$$x' = \frac{p}{r} \cdot \frac{2(S-E)}{(R-L)} + \frac{2(S-E)(P-C)}{(N-E)(R-L)} + \frac{2C-R-L}{R-L},$$

which varies only with beam direction,

$$\alpha = \frac{2(S-E)(F-N)(P-C)}{(F-E)(N-E)(R-L)}$$

and similarly for y' and β.

3.3.1.5.6. *Advanced hidden-line and hidden-surface algorithms*

Hidden surfaces may be handled quite generally with the z-buffer technique described in the previous section but this technique becomes very inefficient with very complicated scenes. Faster techniques have been developed to handle computations in real time (*e.g.* 25 frames s^{-1}) on raster machines when both the viewpoint and parts of the environment are moving and substantial complexity is required. These techniques generally represent surfaces by a number of points in the surface, connected by lines to form panels. Many algorithms require these panels to be planar and some require them to be triangular. Of those that permit polygonal panels, most require the polygons to be convex with no re-entrant angles. Yet others are limited to cases where the objects themselves are convex. Some can handle interpenetrating surfaces, others exclude these. Some make enormous gains in efficiency if the objects in the scene are separable by the insertion of planes between them and degrade to lower efficiency if required, for example, to draw a chain. Some are especially suited to vector machines and others to raster machines, the latter capitalizing on the finite resolution of such systems. In all of these the basic entities for consideration are entire panels or edges, and in some cases vertices, point-by-point treatment of the entire surface being avoided until after all decisions are made concerning what is or is not visible.

All of these algorithms strive to derive economies from the notion of 'coherence'. The fact that, in a cine context, one frame is likely to be similar to the previous frame is referred to as 'frame coherence'. In raster scans line coherence also exists, and other kinds of coherence can also be identified. The presence of any form of coherence may enable the computation to be concerned primarily with changes in the situation, rather than with the totality of the situation so that, for example, computation is required where one edge crosses in front of another, but only trivial actions are involved so long as scan lines encounter the projections of edges in the same order.

The choice of technique from among many possibilities may even depend on the viewpoint if the scene has a statistical anisotropy. For example, the depiction of a city seen from a viewpoint near ground level involves many hidden surfaces. Distant buildings may be hidden many times over. The same scene depicted from an aerial viewpoint shows many more surfaces and fewer overlaps. This difference may swing the balance of advantage between an algorithm which sorts first on z or one which leaves that till last.

These advanced techniques have, so far, found little application in crystallography, but this may change. Ten such techniques are critically reviewed and compared by Sutherland *et al.* (1974), and three of these are described in detail by Newman & Sproull (1973).

references