Drawing opaque solids using an incremental plotter

J. G. Griffiths

Department of Mathematics, University of Salford, Salford M5 4WT

This paper describes an ALGOL program which provides shaded drawings of any opaque polyhedron viewed from three mutually perpendicular directions.

Since the projection of a face onto a plane perpendicular to the direction of vision is a plane polygon, much of the program consists of procedures which perform operations on polygons. The main operation involves processing two polygons that overlap to derive the visible part of the one taken to be underneath. Before this calculation takes place, it is necessary to know that the two polygons chosen do in fact overlap, and to have a criterion for determining which of the two is to be treated as the one on top.

Section 1 of the paper describes the main program except for the procedures, which are described in Section 2. Specimen results are given in Section 3.

(Received July 1970)

1. The main program

A polyhedron is described by a list of $p$ unique vertex co-ordinates stored in an array $xyz[1:3, 1:p]$, together with a list of $f$ faces. A face with $n$ sides whose vertices in cyclic order are the points

$$(xyz[1, i_1], xyz[2, i_2], xyz[3, i_3]),$$
$$(xyz[1, i_2], xyz[2, i_3], xyz[3, i_1]),$$
$$
... (xyz[1, i_n], xyz[2, i_1], xyz[3, i_2])$$

is described by the integers

$$n, i_1, i_2, ... i_n.$$ 

The points from which the polyhedron is observed are taken to be on the co-ordinate axes far distant from the origin. To obtain views from any three mutually perpendicular directions it is necessary first to rotate the polyhedron about the origin. The matrix of such a transformation is obtained by forming the product of the three matrices corresponding to rotation about each individual axis. Using the transformed vertices and the information about each face, an array $s[1:3, 1:f; 1:v]$ is assembled such that $s[i, j, k]$ contains the $j$th co-ordinate of the $k$th vertex on the $i$th face. $u$ is chosen large enough to accommodate the face with the largest number of vertices, and the number of vertices on each face is recorded in an array $pts[1:f]$. The equation of each face is of the form $ax + by + cz = p$, where $a^2 + b^2 + c^2 = 1$, and a further array $eq[1:4, 1:f]$ is created to contain the coefficients. The equation of a face is needed to decide whether or not the face is partially or completely hidden by some other face and to determine how densely the face should be shaded in the final picture.

A projection of the polyhedron onto one of the co-ordinate planes is obtained from $s$ simply by ignoring the co-ordinate in $s$ corresponding to the axis perpendicular to that plane.

Thus from $s$ is obtained a list of overlapping co-planar polygons. The list is processed so that every projection in it is replaced by the part of the projection corresponding to that part of the original face not hidden by any other face. Since each view requires two of the three co-coordinates of points in $s$, calculations are performed on copies of the co-ordinates, stored in arrays $pcc, pcy [1:f'; 1:v']$, where $f'$ and $v'$ are discussed later. When part of a polygon is removed because the corresponding portion of face is hidden, the remaining part does not in general have the same number of vertices as the original polygon. Because it is necessary to preserve the contents of $pts[1:f]$ for use with other projections, a working copy is made, $rel[1:f']$. $v'$ is chosen so that the rows of $pcc$ and $pcy$ are long enough to accommodate the greatest number of vertices that is likely to occur in the projection of a partially hidden face. $v' = 3v$ is found to be adequate for most cases. The visible part of a partially hidden face may comprise several separate pieces. The projections of these pieces are treated as independent polygons each with a row of pcc and pcy to itself. $f'$ is chosen so that there is sufficient room to accommodate the greatest number of independent polygons that is likely to occur in the final drawing. If the pessimistic view is taken that every face is split into two pieces, then $2f$ is a reasonable value to choose for $f'$.

Before the main processing begins, a preliminary check is made to determine whether or not any face is hidden by virtue of its being parallel to the axis along which observation takes place. Any such face is marked as hidden by setting to unity the integer corresponding to it in the array $rel[1:f']$.

Every polygon stored in pcc and pcy except the last is compared with every other polygon lower down in the list, so that all possible pairs are considered. The procedure ‘overlap’ determines whether or not the two polygons cover a common area on the plane of projection. If they do not, a new pair of polygons is selected. If the polygons do overlap, the procedure chooses a point $P$ on the common area. The projection derived from the nearer observer is determined by comparing the points at which the line through $P$ parallel to the axis of observation pierces the two faces.

This method of discrimination places some restriction upon the ways in which the faces of a polyhedron may be described if ambiguity is to be avoided. For example: $ABCD$ and $A'B'C'D'$ are the opposite faces of a cube, whose centres are $O$ and $O'$ respectively. Suppose that the diagonal planes $AA'C'C$ and $DD'B'B$ form part of a polyhedron. In general, projections of $AA'C'C$ and $DD'B'B$ cover common area, and the procedure ‘overlap’ finds a point $P$ on the common area. Since each plane hides part of the other, the plane picked out as the one nearer the observer depends upon the position of $P$ relative to the projection of the line $OO'$. The ambiguity is resolved if the configuration is described by means of the four rectangles $AA'O'O$, $CC'O'O$, $DD'O'O$ and $BB'O'O$, instead of the original two. If a composite solid is formed by placing face to face two polyhedra, it is immaterial which of the two touching faces is taken as nearer the observer, since the area common to the two faces is hidden by the rest of the object.

The procedure ‘blot’ obtains from the two polygons one or more independent polygon $R_1, R_2, R_3, ...$ corresponding to the projection of the part of the face more distant from the observer not masked by the nearer face. If the distant face is completely hidden, then it is marked as such by setting to unity the corresponding element of $rel[1:f']$. If the distant face is not completely hidden, then its projection in pcc and pcy is
replaced by \( R_1 \), and the number of vertices in \( R_1 \) is stored in the appropriate element of \( rel[1:1] \). If \( R_2, R_3 \ldots \) exist, they are inserted into the first available unused rows at the bottom of the list, and subsequently processed if necessary.

After every possible pair has been considered, \( pXC \) and \( pCY \) contain only the projections of the visible parts of the polyhedron. These are outlined on the graph plotter and shaded if desired.

2. The procedures

The following four procedures are used in the program

\[
\begin{align*}
\text{procedure } & \text{overlap} \ (px, py; pp, qx, qy; pq, bo, x_o, y_o), \\
& \quad \text{real } x_o, y_o, \\
& \quad \text{integer } pp, pq; \\
& \quad \text{array } px, py, qx, qy; \\
& \quad \text{boolean } bo; \\
\end{align*}
\]

The procedure determines whether or not the polygon \( P \) whose \( pp \) vertices are stored in arrays \( px \) and \( py \) overlaps the polygon \( Q \) whose \( pq \) vertices are stored in arrays \( qx \) and \( qy \). If it does, a point \((x_o, y_o)\) common to both interiors is picked out, and the parameter \( bo \) takes the value \text{true}. If \( P \) and \( Q \) do not overlap then \( bo \) takes the value \text{false} and no values are assigned to \( x_o \) and \( y_o \).

Before the two polygons are examined in detail a rough check is made to see if overlapping could not possibly occur. If the smallest rectangle which contains \( P \) and whose sides are parallel to the co-ordinate axes does not overlap the corresponding rectangle for \( Q \), then \( P \) and \( Q \) cannot overlap. If the two rectangles do overlap, then there is at least a possibility that \( P \) and \( Q \) do too.

If there are any points common to both boundaries, then their abscissae are stored in an array \( o \), together with the abscissae of the vertices \( P \) and \( Q \). The elements of \( o \) are sorted into numerical order. If there are no points common to both boundaries, then either one polygon completely encloses the other, or the polygons do not overlap. If there are points common to both boundaries, then the polygons either overlap or touch. The rectangles containing \( P \) and \( Q \) determine two values of \( x \) between which overlapping must occur if it occurs at all. Any straight path parallel to the \( y \) axis cutting the \( x \) axis within this range crosses the boundaries of both \( P \) and \( Q \) a non-zero even number of times.

If the path cuts the \( x \) axis at a point \((x_o, 0)\) midway between two unequal consecutive abscissae in \( o \), then the path never crosses the boundaries of \( P \) or \( Q \) at a vertex or a point common to the two boundaries. The intersections of the path and the boundary of \( P \) are formed into a list of points \( P_1, P_2, \ldots, P_{2m} \), whose ordinates are in ascending order. The \( n \) straight line segments \( P_iP_{i+1}, P_{i+2}P_{i+1}, \ldots, P_{2m}P_{2m} \) consist only of points that lie inside or on the boundary of \( P \). A set of \( m \) straight line segments \( Q_1Q_2, Q_2Q_3, \ldots, Q_{2m-1}Q_{2m} \) is similarly derived for \( Q \). A search is made for a pair of line segments \( P_iP_{i+1}, Q_jQ_{j+1} \) which have a section of the path in common. If such a pair is found, then \( P \) and \( Q \) overlap and the mid-point of the common section of path is chosen as the point common to the interiors of \( P \) and \( Q \). If such a pair of line segments is not found, then another search is made on a new path parallel to the old one and passing midway between the next consecutive pair of unequal abscissae in \( o \).

The process continues until either a suitable point \((x_o, y_o)\) is found, or the path avoids one of the polygons completely, in which case \( P \) and \( Q \) do not overlap.

\[
\begin{align*}
\text{procedure } & \text{decide} \ (x, y; n, x_o, y_o); \\
& \quad \text{real } x_o, y_o, \\
& \quad \text{integer } n; \\
& \quad \text{array } x, y; \\
\end{align*}
\]

The procedure determines the position of the point \((x_o, y_o)\) relative to a polygon with \( n \) vertices stored in \( x \) and \( y \). The principle invoked is that any path joining a point inside the polygon to a point outside crosses the boundary of the polygon an odd number of times, and that any path joining two points outside the polygon crosses the boundary an even number of times. In the present case the path chosen is the half-line joining \((x_o, y_o)\) to \((x_o, \infty)\).

If a vertex lies on the path then the crossing is counted twice because each of the sides meeting at the vertex is crossed, and the wrong conclusion is drawn. One way of avoiding this difficulty is to define the side of a polygon as the open set of all points that lie on the line segment joining two consecutive vertices, together with the vertex at the beginning of the line, but excluding the vertex at the end. Although this definition is adequate for unexceptional cases, a simple diagram shows that it does not yield the wanted result if the two sides meeting at the offending vertex happen to lie entirely to one side of the path. Further complications arise if part of the boundary coincides with the path.

The simplest way of avoiding these pitfalls is to choose a new path whenever a vertex is encountered, and to begin the calculation again. Since a path parallel to one of the co-ordinate axes is computationally simpler to deal with, the old path is retained but the polygon is rotated about \((x_o, y_o)\) by an angle incommensurate with a complete revolution, say one radian. A path with no difficulties on it will always be found if the polygon is rotated sufficiently often, and in practice one rotation is almost certain to suffice.

\[
\begin{align*}
\text{procedure } & \text{blop} \ (px, py; pp, qx, qy; pq, rx, ry, pr); \\
& \quad \text{integer } pp, pq, pr, \\
& \quad \text{array } px, py, qx, qy, rx, ry; \\
\end{align*}
\]

The first six parameters in this procedure have the same meaning as the corresponding parameters in the procedure 'overlap' except that in this case the areas bounded by the polygons \( P \) and \( Q \) are known to be overlapped. The area common to \( P \) and \( Q \) is removed from \( P \), and the polygon (or polygons) \( R \) bounding the remaining area are stored in \( rx \) and \( ry \). If the remaining area comprises several fragments, the vertices describing one fragment are separated from the vertices describing the next by a point that could not arise in any other context. The integer \( pr \) contains the total number of vertices and separators left in \( rx \) and \( ry \) by the procedure. If there is no remaining area, \( pr \) is set to unity.

From \( P \) an augmented polygon is formed whose vertices comprise the vertices of \( P \) together with the points common to the boundaries of \( P \) and \( Q \). To ensure that the intersections and vertices are stored in the correct cyclic order within the augmented arrays, a number \( \lambda = i + 1 \) for \( P \) and \( P_{i+1} \) is associated with a point of intersection \( i \) on the \( i \)th side \( P_iP_{i+1} \). Using this device, the intersections are found in any order, and correctly sorted afterwards. \( Q \) is treated in the same way. The two augmented polygons together form a set of line segments whose members can be divided into two classes: those which form part of the boundary of \( R \) and those which do not. The line segments are sorted into their respective classes by considering the position of one test point on each line segment relative to the polygon to which the line segment does not belong. The test point is taken as the mid-point of the line segment under consideration. If a test point on \( P \) lies inside \( Q \), then the line segment is hidden and does not form part of \( R \), but if it lies outside \( Q \) then the line segment is visible and does form part of \( R \). If a test point on \( Q \) lies inside \( P \), then the line segment forms part of \( R \), but if the test point lies outside \( P \) then the line segment does not form part of \( R \).

If a test point is found to lie on both boundaries then \( P \) and \( Q \) have a line-segment in common. To decide which class the shared line segment belongs to, a new test point is chosen sufficiently near to the old one to ensure that the straight path
joining the old test point to the new one does not cross the boundary of either polygon. If the position of this new test point relative to either polygon is the same, then \( P \) and \( Q \) have common interior points on one side of the common boundary segment, which does not therefore form part of \( R \). If the position of the new test point relative to \( P \) is different from its position relative to \( Q \), then \( P \) and \( Q \) have no common interior points in the immediate vicinity of the shared part of the boundary. Consequently the common line segment belongs to \( R \) in this case. The new test point is chosen to be the point midway between the old one and the intersection of either boundary and the perpendicular bisector of the common line segment, nearest to the old point.

This elaborate process need only be performed upon one of the polygons, say \( P \). When a segment of \( Q \) is found whose midpoint lies on the boundary of \( P \) a decision has been made on a previous occasion. If the decision was that the line segment of \( P \) does not form part of \( R \), then neither does its duplicate in \( Q \). If the decision was that the line segment of \( P \) does form part of \( R \), then the inclusion of its copy would cause \( R \) to have a repeated line segment. Such a repition could make the procedure 'decide' reach the wrong conclusion at some future stage in the calculation. Thus in either case, the line segment belonging to \( Q \) is not incorporated into \( R \).

If the boundaries of \( P \) and \( Q \) intersect at vertices, as they often do when \( P \) and \( Q \) are projections of two faces of a polyhedron, then repeated points are introduced into the augmented arrays. These are equivalent to the end-points of line segments having zero length, and would cause program failure if an attempt were made to set up the equation of the perpendicular bisector. All zero line segments are therefore counted as hidden.

When the class of every line segment of \( P \) and \( Q \) is known, all those belonging to \( R \) are picked out and stored in any order in an array \( poly \) which has five columns and a sufficiently large number of rows. Each row stores a line segment of \( R \). The first four columns contain the co-ordinates of the end points. The fifth column is used to denote whether or not the line has been used up at some point in the assembly of \( R \), and prevents its being used a second time. The construction of \( R \) is started by placing in \( rx[1] \) and \( ry[1] \) respectively the co-ordinates of one end of the line segment stored in the first row of \( poly \). \( Poly \) is then searched for a line segment one of whose end points coincides with the end of the previous segment found not placed in \( rx \) and \( ry \). When such a segment is found, the co-ordinates of the coinciding end points are inserted in \( rx \) and \( ry \). The process continues until no line segment is found that fits onto the last one found. If there are still some unused line segments left in \( poly \), then fragmentation has occurred. In this case the separator point is inserted in \( px \) and \( py \), and another fragment is assembled from the remaining line segments of \( poly \).

procedure shade \((x, y, n, th, sh)\);
real \( th, sh\);
integer \( n \);
array \( x, y \);

The procedure shades a simply connected plane closed polygon with \( n \) vertices stored in \( x \) and \( y \) by instructing a graph plotting device to draw the segments of equally spaced parallel lines of the form \( y \cos \theta - x \sin \theta = h \) cut off by the boundary of the polygon. Any side parallel to the shading is ignored. The co-ordinates of the point of intersection \( P_i \) of the \( ith \) side \( P_i P_{i+1} \) and a shading line with \( h \) set at its initial value are stored in the \( ith \) elements of arrays \( px \) and \( py \), and the ratio \( P_i P_{i+1}:P_{i+1}P_{i+1} \) in the \( ith \) element of an array \( \lambda \). Every time \( h \) is decreased by the spacing \( sh \), the ratios \( \lambda \) and points of intersection \((px, py)\) alter by fixed amounts stored in arrays \( dx, dy, dx, \) and \( dy \). If \( \lambda \) lies between 0 and 1 then \( P_1 \) lies between \( P_i \) and \( P_{i+1} \), and is therefore an end point of a shading line segment. Every time \( h \) is altered, \( \lambda, px \) and \( py \) are modified, and a list of \( p \) end points is picked out of \( px \) and \( py \), and stored in arrays \( wx \) and \( wy \).

If the polygon is convex, then \( p = 2 \). Otherwise \( p \) may be any non-zero positive even integer. The \( p \) end points are sorted with respect to their \( x \) or \( y \) co-ordinates according as \( th < \pi/4 \) or \( th > \pi/4 \) and joined in consecutive pairs. To minimise unproductive pen movement, the points are joined starting with the first point stored for one shading line and starting with the last point stored for the next shading line. The shading is complete when arrays \( wx \) and \( wy \) contain no points to join up.

3. Results

The program was used to draw views of five different objects: a cube, a dodecahedron, a small stellated dodecahedron, a dodecahedral framework, and a small stellated dodecahedron inscribed in a dodecahedral framework. Statistics relating to the five examples are displayed for comparison in Table 1, and a shaded drawing of a framework enclosing a small stellated dodecahedron is given as Fig. 1.

Table 1

<table>
<thead>
<tr>
<th></th>
<th>CUBE</th>
<th>DODECAHEDRON</th>
<th>STELLATED DODECAHEDRON</th>
<th>FRAMEWORK AND DODECAHEDRON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faces</td>
<td>6</td>
<td>12</td>
<td>60</td>
<td>110</td>
</tr>
<tr>
<td>Vertices</td>
<td>8</td>
<td>20</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>Calculation time (secs)</td>
<td>3</td>
<td>10</td>
<td>51</td>
<td>186</td>
</tr>
<tr>
<td>Calls of 'Decide'</td>
<td>96</td>
<td>287</td>
<td>1788</td>
<td>4439</td>
</tr>
<tr>
<td>Calls of 'Overlap'</td>
<td>13</td>
<td>50</td>
<td>1036</td>
<td>3683</td>
</tr>
<tr>
<td>Calls of 'Blot'</td>
<td>8</td>
<td>18</td>
<td>158</td>
<td>283</td>
</tr>
</tbody>
</table>

Fig. 1
The number of procedure calls depends to a small extent upon the new orientation and on the order in which the data is read. Different orientations give rise to numbers of procedure calls differing by not more than about 4 per cent from the typical ones given in the table. Calculation times are for single unshaded drawings. The inclusion of shading can cause the calculation time to rise steeply, especially if the shading is dense. The example shown in Fig. 1 took 437 seconds to calculate.

Acknowledgement
The author wishes to thank Dr. A. Wragg for reading the manuscript and for pointing out a number of obscurities.

References

Book review


Potentially, the publication of a book with this title could have been a very exciting event in Computer Science and Computer Education. Semantics is the new growth industry in University departments and IBM have even supported a laboratory in Vienna, Austria (not Australia as many of their US correspondents appear to believe) whose main mission has been the formal semantics of PL/I and on whose methodology J. A. N. Lee is totally dependent.

Hitherto, publications in Computer Semantics have either been isolated papers in the learned journals or distributed by IBM, Vienna, or at best collections of Symposia papers. All these publications are fragmentary, being research reports on forays into the boundaries of the field. It would be timely to have an introductory text that covered the objectives of workers in the field and expounded and criticised their work.

Unfortunately this is not the book that J. A. N. Lee has given us. He presents us with the Vienna Language, a notation for programming predicates and transformations of tree-structured data. He then uses this as a tool to describe algorithms, e.g. addition of multi-digit numerals, data-structures, e.g. FORTRAN arrays, computers, e.g. the PDP8 and languages, e.g. BASIC.

It is valuable to show that the same tools, and the same kind of explanation gives equally the semantics of algorithms, data, machines and languages. It is also valuable to observe that the discipline imposed by the Vienna programs that one writes sharpens the precision with which one thinks about algorithms and data and also reveals lacunae in the published description of the PDP8 and BASIC. But, what is lacking throughout Mr Lee's text is any kind of context within which to set the definitions he presents and the methods he uses. Clearly, a method of semantic exposition is to give a function from text to meaning and clearly, to do so, this function must be specified as a program in a programming language. But why this method, and why use the Vienna Language rather than LISP or even ALGOL, FORTRAN, or BASIC? There is a reason, but I could not find any clues to it in Mr Lee's book.

I cannot see who can usefully buy this book. For those who are interested in technical problems of using 'undefined' as a value for 'the evaluation never terminates' the following will confuse: 'if p is T then the value of the expression is F otherwise T'. When p = u (undefined), it is obvious [sic] that the value of p is not T (not to be confused with not-T or F (Author)) and thus the first proposition is not applicable (how do you know (Reviewer)). However, the second proposition contains a tautology, and thus the value of the expression is T.

J. G. LASKI (Colchester)