Graphics and Image Processing
W. Newman Editor

# Reentrant Polygon Clipping

Ivan E. Sutherland and Gary W. Hodgman
Evans and Sutherland Computer Corporation

**A new family of clipping algorithms is described. These algorithms are able to clip polygons against irregular convex plane-faced volumes in three dimensions, removing the parts of the polygon which lie outside the volume. In two dimensions the algorithms permit clipping against irregular convex windows.**

**Polygons to be clipped are represented as an ordered sequence of vertices without repetition of first and last, in marked contrast to representation as a collection of edges as was heretofore the common procedure. Output polygons have an identical format, with new vertices introduced in sequence to describe any newly-cut edge or edges. The algorithms easily handle the particularly difficult problem of detecting that a new vertex may be required at a corner of the clipping window.**

**The algorithms described achieve considerable simplicity by clipping separately against each clipping plane or window boundary. Code capable of clipping the polygon against a single boundary is reentered to clip against subsequent boundaries. Each such reentrant stage of clipping need store only two vertex values and may begin its processing as soon as the first output vertex from the preceeding stage is ready. Because the same code is reentered for clipping against subsequent boundaries, clipping against very complex window shapes is practical.**

**For perspective applications in three dimensions, a six-plane truncated pyramid is chosen as the clipping volume. The two additional planes parallel to the projection screen serve to limit the range of depth preserved through the projection. A perspective projection method which provides for arbitrary view angles and depth of field in spite of simple fixed clipping planes is described. This method is ideal for subsequent hidden-surface computations.**

**Key Words and Phrases: computer graphics, hidden-surface, clipping**

**CR Categories: 5.31, 6.32, 6.35**

## Introduction

In this paper we are concerned with the computer processing of three-dimensional data to produce realistic two-dimensional pictures. The art of producing such pictures has become fairly widespread, and the variety of approaches available has become manifold. Among all new approaches and the multitude of algorithms on which they depend, there are some common steps which all systems must perform. Polygon clipping is one such step, which we hope to simplify in the discussion which follows.

Polygon clipping reduces a polygonal surface extending beyond the boundary of some three-dimensional viewing volume to a surface which does not extend beyond the boundary. The process "clips off" those parts of the polygon which lie outside the volume. Although some hidden-surface algorithms manage without an explicit polygon clipping step, all such algorithms must somehow eliminate off-screen objects.

For a long time we believed that polygon clipping should be a simple extension of line clipping. Indeed, Archuleta's polygon clipping algorithm [1], long used for all of the hidden-surface pictures at the University of Utah, is just such an extension. Such an algorithm considers a polygon to be enclosed by $N$ edges, and by clipping each edge against all the clipping boundaries, seeks to define a smaller polygon entirely within the boundaries. Unfortunately, this approach involves complicated reasoning to determine where to add edges along the boundary. For example, if a given polygon surrounds a corner of the viewing area, then two new edges must be added which share a vertex at the corner. It is a somewhat tedious process to compute whether or not a polygon surrounds a corner of the viewing area.

The algorithm described in this paper abandons the notion of defining a polygon in terms of its edges, and instead considers it to be defined solely by its vertices. Thus, the first vertex of the polygon need not be mentioned again in the definition of the final edge. The algorithm also abandons the notion of clipping each separate edge against all clipping boundaries simultaneously in favor of clipping the entire polygon against each clipping boundary in sequence. With this new approach it is no longer necessary to reassemble a polygon out of a disjoint collection of clipped edges. These changes in approach have resulted in dramatic simplification of the polygon clipping process.

The simplification of the polygon clipping algorithm has been achieved without paying a price in capability or storage requirement. The algorithm described is



Authors' address: Evans and Sutherland Computer Corporation, 3 Research Road, Salt Lake City, UT 84112.

applicable to any polygon, convex or concave, planar or non-planar. The algorithm proceeds to use each acceptable vertex found by one stage of clipping immediately in subsequent stages. There is no need to complete processing of an entire polygon nor to store an entire polygon between stages of clipping. Because the algorithm is reentrant, code is required for a single clipping stage only; reentrant calls will provide subsequent clipping stages. We are somewhat chagrined that the obvious extension of work on line clipping with which we have been involved [11] kept us so long from seeing the simplicity of the present approach.

## Perspective Projection

When producing perspective pictures of opaque objects we use depth information to compute which of two objects is nearer to the observer. We might do these computations before making the perspective projection, using the true depth of objects for comparison but we would then be troubled by the difficulty of computing the relative lateral position of objects considering the parallax of the viewpoint. It is generally much more convenient to make a three-dimensional perspective projection first since this projection puts all of the objects into their correct positions on the screen while preserving not only depth ordering but also the fact that straight lines project into straight lines [8]. In effect, the application of a proper perspective projection to a set of three-dimensional objects enables us to treat the objects as if they were being viewed from infinity.

Roberts [10] published a formulation of the three-dimensional perspective projection of the point $[X\ Y\ Z\ 1]$ into the point $[X'\ Y'\ Z'\ 1]$:

$$[X\ Y\ Z\ 1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 1/D\\ 0 & 0 & 0 & 1\end{bmatrix} = [x'\ y'\ z'\ w'],$$

$$X' = \frac{x'}{w'} = \frac{X}{(Z/D)+1},$$

$$Y' = \frac{y'}{w'} = \frac{Y}{(Z/D)+1}, \qquad (1)$$

$$Z' = \frac{z'}{w'} = \frac{Z}{(Z/D)+1}.$$

where $X$, $Y$, and $Z$ are measured from an origin at the center of the screen and $D$ is the viewing distance; viewing is assumed to be from a position on the negative $Z$ axis. If we remember that the $Z$ displacement of any point from the observer's eye is $Z + D$, we can easily confirm that this formulation is the familiar one of dividing lateral displacement by viewing range. Notice that as in all perspective projections, a division step is required.

The formulation of eq. (1) assumes the origin of coordinates to be at the center of the screen rather than at the observer's eye. This may seem strange and unnatural but it has two important benefits which amply justify the assumption. First, with the origin at the screen, the eye may be moved as far from the screen as desired without introducing difficulties in the equations. As $D$ in eq. (1) becomes infinite, the transformation reduces gracefully to the identity transformation which would be appropriate for orthographic projection. Second, the formulation of eq. (1) is entirely symmetric in $X$, $Y$, and $Z$, a property which simplifies not only the perspective computations but also our understanding of the perspective process. Equation (1) exhibits explicitly that the perspective process treats all axes identically.

## Clipping Before Division

The perspective computations alone are inadequate to meet most needs because they will erroneously project points behind the observer upside down and backwards onto the screen in front of him. The final division process in the perspective projection destroys one bit of sign information, and this is precisely the bit that keeps objects in front of an observer in front of him (see Appendix A). If a line connects a point in front of an observer with a point behind him, it will project into a very strange line indeed if its ends are simply projected by the ordinary rules. Moreover, a point "alongside" the observer, i.e. with the same value of $Z$ as his eye, projects into a point with infinite values of $X'$, $Y'$, and $Z'$, clearly a difficult point to represent. As we learned in three-dimensional line clipping earlier [11], we must limit the permissible values of $x'$, $y'$, $z'$, and $w'$ *prior* to doing the division.

If we are careful to limit the values of $x'$, $y'$, $z'$, and $w'$ appropriately, we can make the range of possible values for $X'$, $Y'$, and $Z'$ come out to be very simple. For example, it would be nice to have $X'$ and $Y'$ in the range $-1$ to $+1$ because they would then be easy to scale to any desired screen coordinate system. It would also be nice to have $Z'$ in the range 0 to 1, because it would then be easy to scale to any available precision in a subsequent depth computation. This suggests that we always clip on the limits:

$$\begin{aligned} -w' &\le x' \le +w',\\ -w' &\le y' \le +w',\\ 0 &\le z' \le +w'. \end{aligned} \qquad (2)$$

These clipping limits correspond to six planes which we will call "left," "right," "bottom," "top," "hither," and "yon."

The six clipping planes of eq. (2) form a truncated pyramid. In raw form, the transformation of eq. (1) and the clipping of eq. (2) result in a fixed field of view with half angle defined by $\tan \alpha = 1/D$. It would be

nice if we were not limited to a fixed field of view, and if the hither and yon clipping planes could be placed at will.

We can, in fact, clip to any field of view and place the hither and yon clipping planes at will. The transformation:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{S}{F}+\frac{S}{D} & \frac{S}{D} \\ 0 & 0 & 0 & S \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{S}{F}+\tan\alpha & \tan\alpha \\ 0 & 0 & 0 & S \end{bmatrix} \quad (3)$$

places the six clipping planes at any desired locations, as shown in Figure 1. The viewing distance, $D$, can even be made infinite, as can the position of the "yon" plane, without causing any anomalous behavior of the transformation matrix. It is nice that the transformation matrix can contain all information for an arbitrary truncated pyramid, because the clipping process can then be standardized to the very simple clipping limits.

Fig. 1. The clipping volume is a truncated pyramid with dimensions as shown. Note that the origin of coordinates is in the "hither" clipping plane and the eye is located at $[0, 0, -D]$.

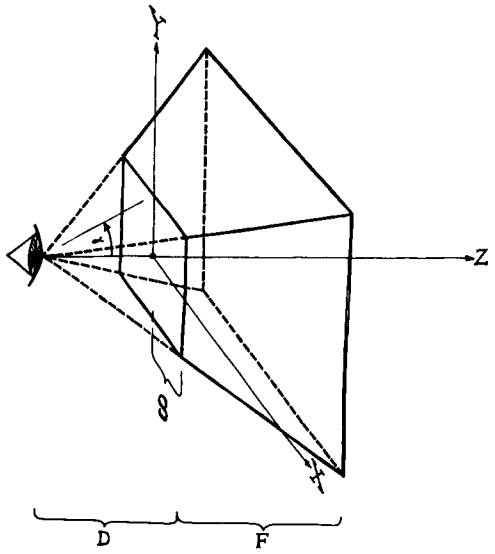
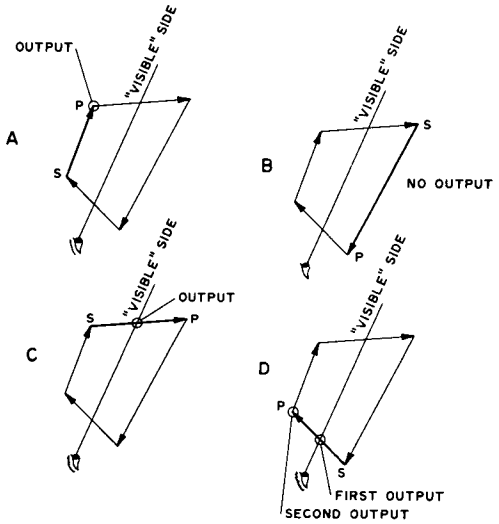



FIGURE 1

Fig. 2. A polygon edge can relate to a clipping plane in four ways: (a) both ends visible; (b) neither end visible; (c) edge leaves visible region; and (d) edge enters visible region.




## Clipping a Polygon Against a Single Plane

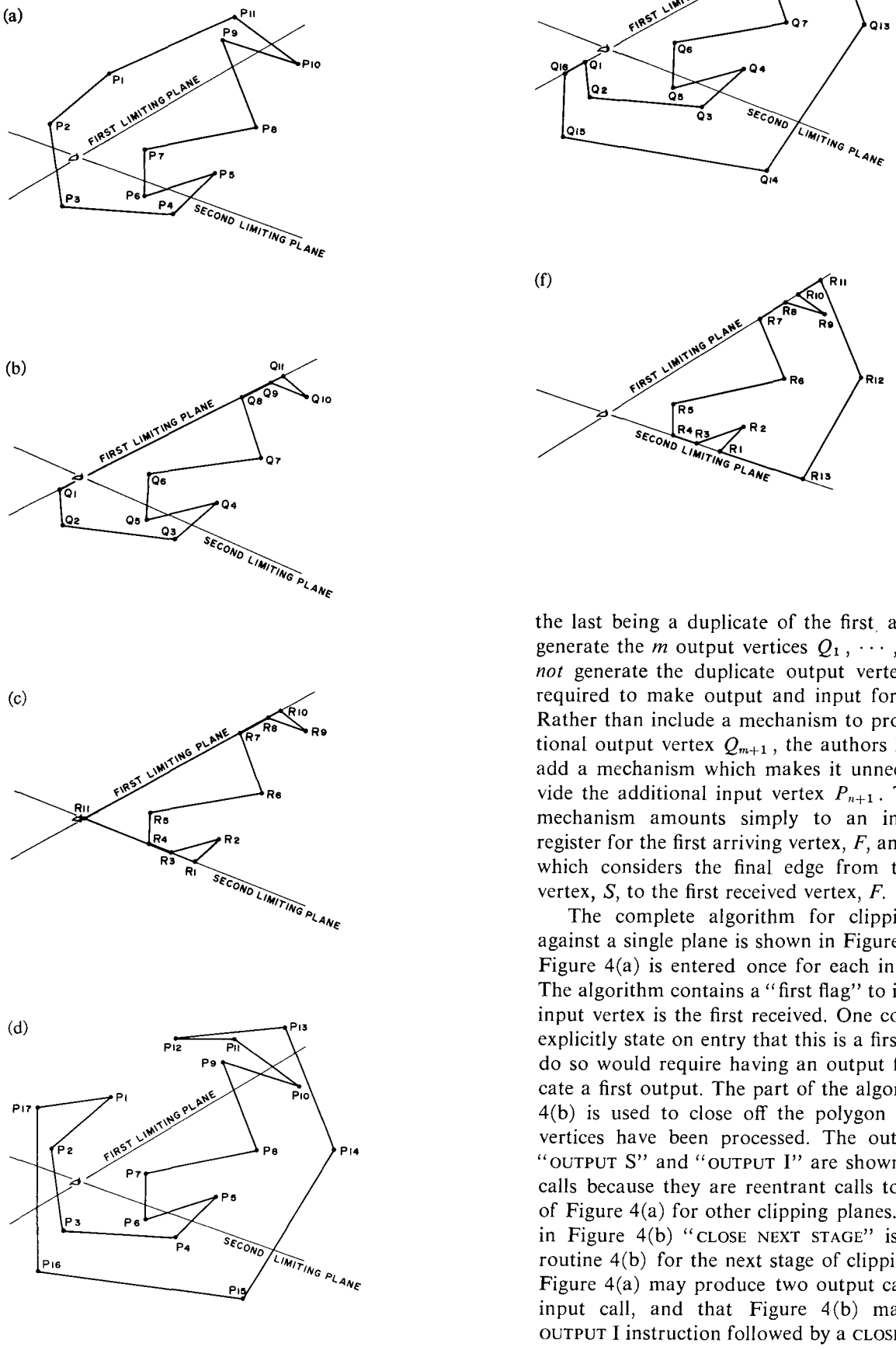
We will define a polygon as an ordered set of vertices $P_1, \cdots, P_n$. It will not be necessary to duplicate the first vertex at the end of the set because we will automatically close every polygon by producing edges $P_1P_2$, $P_2P_3, \cdots, P_{n-1}P_n$, $P_nP_1$. We will not insist that the polygon be planar in any sense at all, although the meaning of a nonplanar polygon may not be entirely clear. Our objective is to produce a new set of points $Q_1, \cdots, Q_m$ which defines another polygon identical to the first on the "visible" side of the clipping plane but which does not include any points on the other side of the clipping plane.

The clipping algorithm for a polygon against a single clipping plane considers the input vertices $P_1, \cdots, P_n$ one at a time. For each such input vertex, zero, one or two of the output vertices $Q_1, \cdots, Q_m$ will be generated, depending on the positions of the input vertices with respect to the clipping plane. Each input vertex, $P$, except the first, is considered to be the terminal vertex of an edge: namely, the edge defined by the input vertex value, $P$, and the position, $S$, of the just previous input vertex saved internally in a register. The algorithm will generate output vertices from the set $Q_1, \cdots, Q_m$ depending on the relationship between the input vertex, $P$, the saved vertex, $S$, and the clipping plane.

The four possible relationships between the edge and the clipping plane are shown in Fig. 2. If the edge is entirely on the visible side of the clipping plane, only its terminal vertex $P$, need be output since its initial vertex, $S$, will already have been output. If the edge is entirely on the other side of the clipping plane, nothing need be output. If the edge leaves the visible side of the clipping plane, with $S$ visible and $P$ not, then only the intersection of the edge and the clipping plane need be output because the initial vertex, $S$, will already have been output. If the edge enters the visible side of the clipping plane, with $P$ visible and $S$ not, then two output vertices must be generated: the intersection of the edge and the clipping plane, followed by the terminal vertex, $P$. The reader may wish to check how these four cases apply in the more complicated examples of Figure 3.

Some special care must be taken at the beginning and end of each polygon processed. The algorithm outlined above might be given $n + 1$ input vertices,

Fig. 3. Sequential clipping applied to a complex polygon. Figures (a), (b), and (c) show a polygon surrounding the observer. Figures (d), (e), and (f) show an almost identical polygon which does not surround the observer.




the last being a duplicate of the first, and would then generate the $m$ output vertices $Q_1, \cdots, Q_m$. It would *not* generate the duplicate output vertex $Q_{m+1} = Q_1$ required to make output and input formats identical. Rather than include a mechanism to produce the additional output vertex $Q_{m+1}$, the authors have chosen to add a mechanism which makes it unnecessary to provide the additional input vertex $P_{n+1}$. This additional mechanism amounts simply to an internal storage register for the first arriving vertex, $F$, and a closing call which considers the final edge from the final saved vertex, $S$, to the first received vertex, $F$.

The complete algorithm for clipping a polygon against a single plane is shown in Figure 4. The part in Figure 4(a) is entered once for each input vertex $P_i$. The algorithm contains a "first flag" to indicate that an input vertex is the first received. One could, of course, explicitly state on entry that this is a first vertex, but to do so would require having an output flag bit to indicate a first output. The part of the algorithm in Figure 4(b) is used to close off the polygon after all of its vertices have been processed. The output statements "OUTPUT S" and "OUTPUT I" are shown as subroutine calls because they are reentrant calls to the algorithm of Figure 4(a) for other clipping planes. The statement in Figure 4(b) "CLOSE NEXT STAGE" is a call to the routine 4(b) for the next stage of clipping. Notice that Figure 4(a) may produce two output calls for a single input call, and that Figure 4(b) may produce an OUTPUT I instruction followed by a CLOSE NEXT STAGE.

## Computing the Intersection

Computing the intersection of a line and a plane need not be difficult. If one computes any convenient distance measure between the end points and the plane, one can use the ratios of such measures to compute at which point along the line it intersects the plane. The only requirement is that the two distance measures be computed parallel to each other. In the edge view of Figure 5, dotted lines show two possible distance measures. Notice that each one results in a pair of similar triangles from which one can easily compute the fraction of the line consumed before the line hits the plane. Having this fraction in hand (call it $\alpha$), the intersection point is given by:

$$I = \alpha\overrightarrow{P_2} + (1-\alpha)\overrightarrow{P_1} - \overrightarrow{P_1} + \alpha(\overrightarrow{P_2} - \overrightarrow{P_1}) \quad (4)$$

where

$$|P_1R_1|/|P_2R_2| = |P_1I|/|P_2I|,$$
$$\alpha = |P_1I|/|P_1P_2| = |P_1R_1|/|P_1R_1| + |P_2R_2|. \quad (5)$$

It is convenient in our case to use a distance measure parallel to the axis rather than perpendicular to the plane. Therefore, $\alpha$ is computed simply by subtracting and dividing some simple coordinate values.

For example, if we wish to limit the value of $x$ to the range $-w \leq x \leq w$, we can use $x-w$ as our distance measure for the right clipping plane and $x-(-w)$ for the left clipping plane. Because we compute an intersection only when the edge crosses the clipping plane, the distance measures for the two vertices will always be opposite in sign. In order to get the sum of their magnitudes, therefore, we can subtract rather than taking absolute magnitudes before adding. Thus, for the right clipping plane, $\alpha = (x_1-w_1)/(x_1-w_1 - (x_2-w_2))$, a form guaranteed to produce $0 \leq \alpha < 1$ if the two points are on opposite sides of the clipping plane. Notice that the sign of any convenient distance measure serves to answer the questions posed by the algorithm about which side of the clipping plane each vertex is on.

## Clipping Against Multiple Planes

If a collection of the simple clipping algorithms are connected together in tandem, they will clip to a succession of planes. As was implied in Figure 3, the output of each simple algorithm is passed along as the input to the next. If clipping is to be done to six planes, six algorithms will be required. A total of twelve vertex storage locations will be required, six for $F$'s and six for $S$'s. Six "first point" flags will also be required.

The six algorithms are, of course, identical except for the specific locations in which they store $F$, $S$, and the "first point" flag, and except for the distance measure that they use in computing $\alpha$. It is, therefore, easy to produce them all using the same reentrant code.

Fig. 4. Flowchart for the polygon clipping algorithm. Part (a) is used for each polygon vertex; part (b) is used to close off the polygon.

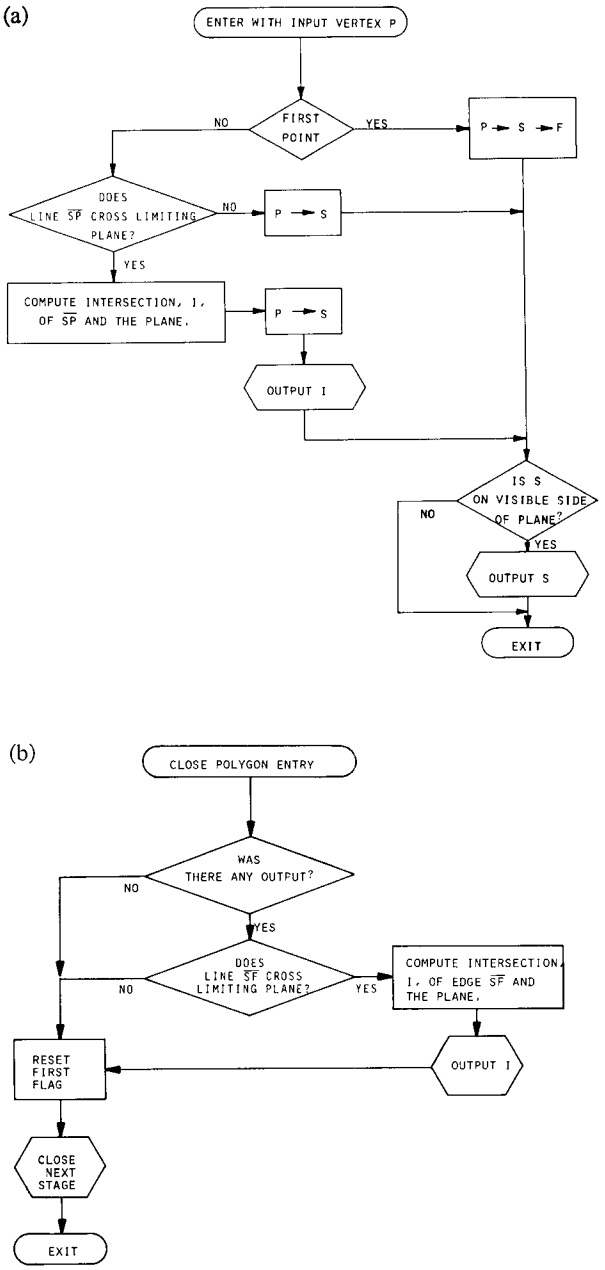



Fig. 5. The similar triangles $P_1R_1I$, and $P_2R_2I$ seen by looking edge-on at a line penetrating a limiting plane make it easy to compute the point of intersection. Any convenient measure of the distance from the ends of the line to the plane will serve equally well.

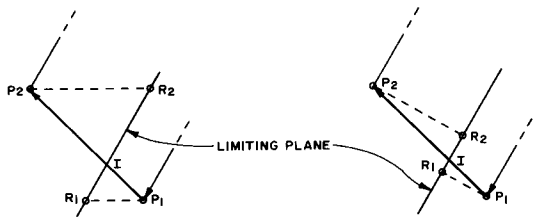
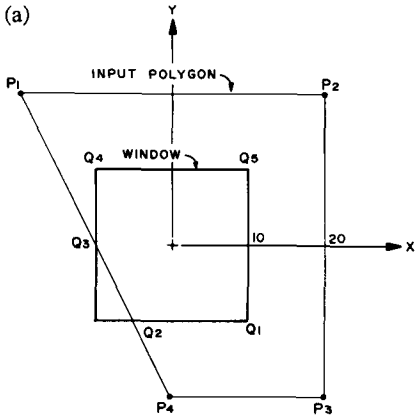


Fig. 6. An example of polygon clipping: the polygon $P_1P_2P_3P_4$ almost completely surrounds the field of view or "window." The table in (b), patterned after our test program's output, shows the points progressing through the various stages of clipping, as shown by the "level" column.

(a)




(b)

| LEVEL | X | Y | COMMENTS |
|---|---|---|---|
| Given | −20 | 20 | |
| 1 (top) | No output | | Point is out, becomes $F$ and $S$ |
| Given | 20 | 20 | |
| 1 (top) | No output | | Point is out, becomes $S$ |
| Given | 20 | −20 | |
| 1 (top) | 20 | 10 | Point is in, becomes $S$, output $I$ first |
| 2 (bottom) | 20 | 10 | Point is in, becomes $S$ and $F$ |
| 3 (left) | 20 | 10 | Point is in, becomes $S$ and $F$ |
| 4 (right) | No output | | Point is out, becomes $S$ and $F$ |
| 1 (top) | 20 | −20 | Output $S$ point |
| 2 (bottom) | 20 | −10 | Point is out, becomes $S$, output $I$ |
| 3 (left) | 20 | −10 | Point is in, becomes $S$ |
| 4 (right) | No output | | Point is out, becomes $S$ |
| Given | 0 | −20 | |
| 1 (top) | 0 | −20 | Point is in, becomes $S$ |
| 2 (bottom) | No output | | Point is out, becomes $S$ |

NOTICE THAT NOTHING HAS REACHED HITHER OR YON CLIPPING STAGES YET.

| | | | |
|---|---|---|---|
| Given | Flush | | |
| 1 (top) | −15 | 10 | $S$ is in, $F$ is out, output $I$ |
| 2 (bottom) | −5 | −10 | Point is in, becomes $S$, output $I$ first |
| 3 (left) | −5 | −10 | Point is out, becomes $S$, output $I$ |
| 4 (right) | 10 | −10 | Point is in, becomes $S$, output $I$ first |
| 5 (hither) | 10 | −10 | |
| 6 (yon) | 10 | −10 | $Q1$ is output |

Figure 6(a)

| LEVEL | X | Y | COMMENTS |
|---|---|---|---|
| 4 (right) | −5 | −10 | Output $S$ point |
| 5 (hither) | −5 | −10 | |
| 6 (yon) | −5 | −10 | $Q2$ is output |
| 2 (bottom) | −15 | 10 | Output $S$ point |
| 3 (left) | −10 | 0 | Point is "out", output $I$ |
| 4 (right) | −10 | 0 | |
| 5 (hither) | −10 | 0 | |
| 6 (yon) | −10 | 0 | $Q3$ is output |
| 1 (top) | Flush | | Output flush next |
| 2 (bottom) | Flush | | $F$ and $S$ both in |
| 3 (left) | −10 | 10 | $S$ is on, $F$ is out, output $I$ |
| 4 (right) | −10 | 10 | |
| 5 (hither) | −10 | 10 | |
| 6 (yon) | −10 | 10 | $Q4$ is output |
| 3 (left) | Flush | | Output flush next |
| 4 (right) | 10 | 10 | $S$ is in, $F$ is out, output $I$ |
| 5 (hither) | 10 | 10 | |
| 6 (yon) | 10 | 10 | $Q5$ is output |
| 4 (right) | Flush | | |
| 5 (hither) | Flush | | |
| 6 (yon) | Flush | | End of output polygon |

Figure 6(b)

A six level stack will be required for the first point flag and for $F$ and $S$, and different distance measurement parameters must be used at each level of reentrance.

It is a little more difficult to comprehend what happens as a polygon is processed through the resulting composite process. If a point is found to be "in" by any clipping level, it will be passed on to the next level, and thus, points tend to progress directly through the process to the first level where they are found to be "out." Because each level may produce two outputs for one input, processing tends to repeat the sequence of levels starting again at some intermediate level which produces its second output. For the relatively simple case of Figure 6(a), the complete sequence of processing is shown in Figure 6(b). Notice that, in this example, no output is generated until after the CLOSE NEXT STAGE (or "flush") call is given to the first clipping level.

## Accuracy in Depth

The problem of available depth resolution has been plaguing development of hidden-surface algorithms for some time. To get better speed or to utilize simpler hardware, many hidden-surface algorithms use fixed point arithmetic in their depth computations. Because the perspective depth computation involves a division, however, these hidden-surface programs have often found that the range of perspective depth numbers generated by the perspective transformation exceeded the range provided for by the number of bits used for perspective depth numbers. Moreover, the depth resolution realized by a particular scaling changes rapidly with range, and thus, where adequate depth resolution was available in a foreground, not enough resolution was available in the background! Heretofore, the solution to this problem was simply to provide as many bits for perspective depth as one could afford, and then hope that the objects shown would not require greater depth resolution.

With hither and yon clipping, on the other hand, the available range of perspective depth is distributed as uniformly as possible over the space between the hither and yon clipping planes. As one might expect, the closer the hither and yon clipping planes are to each other, the more depth resolution will be available in the space between them. Thus, just as one would place the left and right clipping planes to just encompass the width of the object to be portrayed, so one should place the hither and yon clipping planes to just encompass it in depth.

It is interesting to know within this general rule, however, just how much depth resolution will be provided by any particular number of bits used to represent perspective depth. If $n$ bits are used to represent the perspective depth number, then the actual depth

Fig. 7. Depth resolution is a function of the hither clipping distance, $D$. Making the hither clipping distance bigger, as shown in the lower figure, improves the depth resolution.

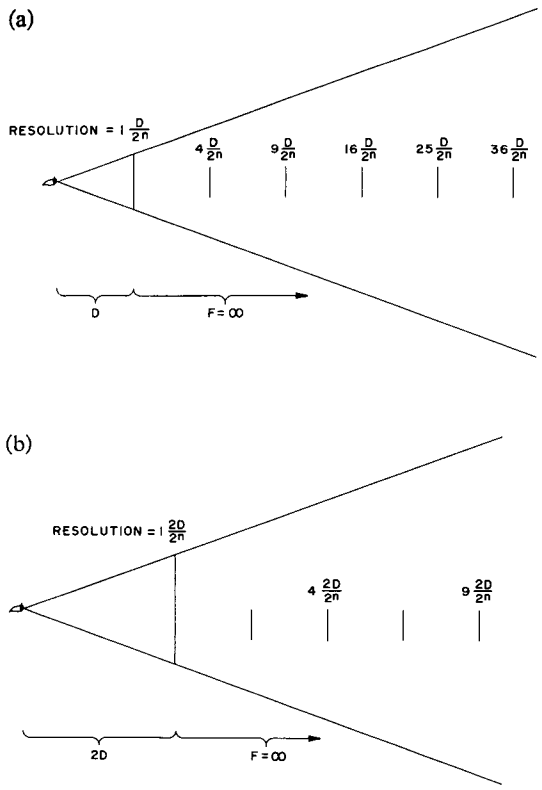
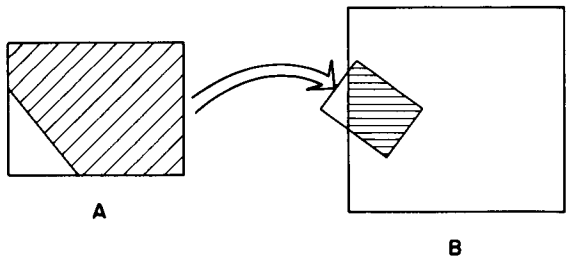



Fig. 8. Polygon clipping permits one to handle clipping properly for subpictures which are rotated. If an object within frame $A$ is used as a subpicture within frame $B$, only part of the object within the crosshatched area will be visible; thus, only the crosshatched part of frame $A$ should be used for clipping.




resolution in object space adjacent to the hither plane is always $D/2^n$. This resolution deteriorates as the square of the distance from the observer as shown in Figure 7(a). This available resolution is improved markedly by moving the hither clipping plane further from the observer. As shown in Figure 7(b), doubling the hither distance improves the depth resolution throughout the remaining viewing volume by a factor of two. The available resolution is also improved somewhat by bringing the yon plane closer to the observer. Thus, one should be prepared to compute a sensible position for the hither and yon planes, probably not more than a factor of 10 away from the minimum or maximum distances that they could assume without clipping the data.

In practical use, the programmer will establish a distance for the hither and yon planes either by doing a maximum/minimum search on his data, or by prior knowledge of the geometry of the situation. For example, in a flight simulator, the hither limit may well be established at 20 feet from the pilot, since any object coming closer to him than that will inevitably have a traumatic result on his aircraft. For a single aircraft situation, the hither distance might alternatively be established as equal to aircraft altitude, since no part of the ground can be closer than that. The yon distance might be established either according to horizon fading, or according to the maximum distance at which detail is visible, or by other geometric considerations. Because one pays only a slight penalty in resolution for placing the yon plane at infinity, that may become the preferred mode of operation.

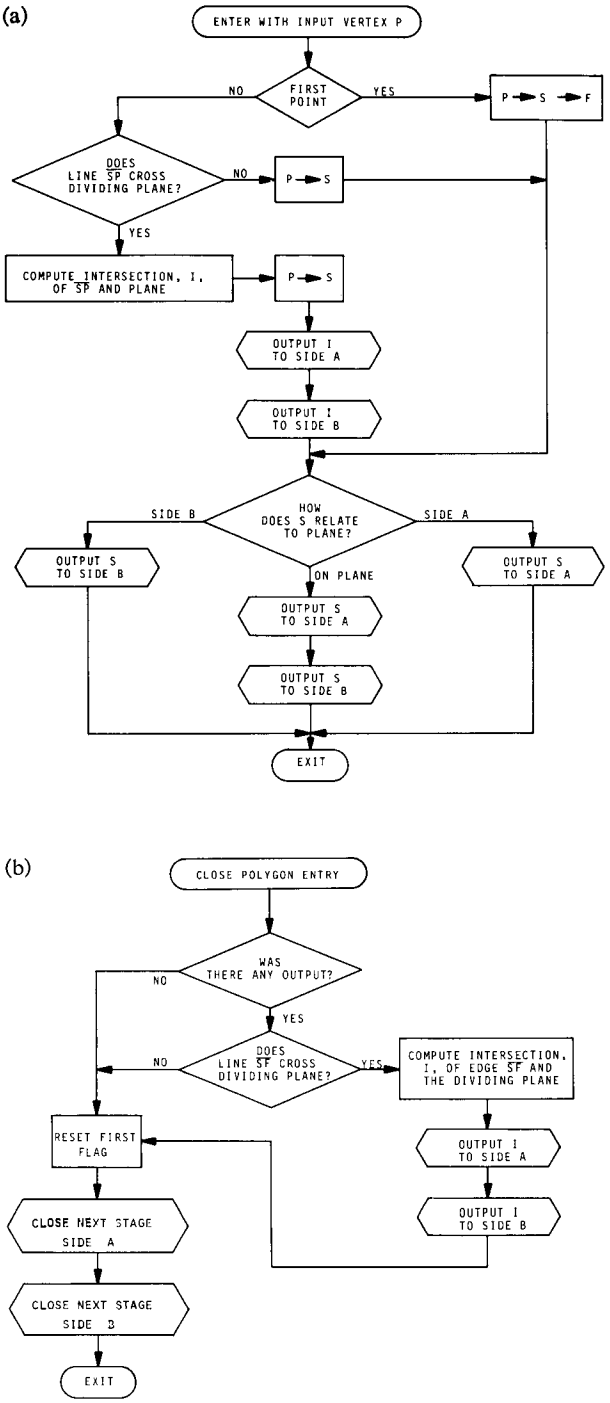
## Application of the Algorithm to Line Clipping

The algorithm can be used with minor modifications to clip lines rather than polygons. The parametric method of computing the intersection between the line and the clipping boundary given in eq. (4) remains applicable as does the method for finding the parameter $\alpha$ outlined in eq. (5). The modified algorithm need only distinguish "set points," the beginnings of lines and the intersections of lines entering the visible region, from "end points," the ends of lines and the intersections of lines leaving the visible region.

There are, however, simpler ways of making a line clipper than as six separate stages of clipping (or four stages for a two-dimensional clipper). To clip a given line against any number of clipping planes requires storage of only two points, not storage of two points per clipping plane as outlined above. To clip a line against an arbitrary number of clipping planes, one merely saves the two end points of the remaining visible segment of the line, replacing either the beginning or the end of the line with new data corresponding to any clipping plane which cuts the line. This is substantially simpler than the reentrant approach.

Moreover, if one expects the line to be clipped by more than one plane, one should save not the computed end points of the reduced line segment, but rather the values of the parameter $\alpha$ (see eq. 4) for the beginning and end of the visible segment. If a given clipping plane clips off some more of the line near its beginning, the new value of $\alpha$ will be greater than the remembered value of $\alpha$ at the start of the visible piece. Thus, the clipping computation involves merely the computation of various values of $\alpha$ and a comparison to retain a minimum and maximum value of $\alpha$ between which the line is visible. If the minimum value of $\alpha$ is ever forced to exceed the maximum value of $\alpha$, then the line is entirely outside the window. Only after obtaining

Fig. 9. A modified clipping algorithm which divides a polygon into two parts at a division plane. Such an algorithm is useful in the Warnock algorithm [13] or for simultaneous clipping for two adjacent viewing windows such as often appear in flight simulation systems.




values of $\alpha$ for the start and end of the visible segment, would one compute the actual positions of the start and end of the line by applying eq. (4).

The algorithm unmodified has a much more significant role to play in two-dimensional line clipping, however, for it enables us to easily handle windows of arbitrary shape and complexity. The need to handle windows of complex shapes arises when two-dimensional pictures are specified by means of instances of subpictures. It is convenient to place such instances in the picture by specifying a bounding rectangle for the area in which the instance is to appear and a bounding rectangle for the area of the subpicture to be used. One would like, in general, to be able to clip such subpictures before transforming them. If, however, the instance is rotated, as shown in Figure 8, the clipping region may not remain rectangular. The polygon clipping algorithm is directly applicable to the task of clipping the bounding polygons themselves. This algorithm determines the irregular shape of the composite bounding polygon, which is subsequently used for clipping the subpicture. This application was first brought to our attention by W.M. Newman [9].

## Dividing a Polygon into Two Parts

A minor modification of the algorithm described here makes it possible to divide polygons into two parts rather than preserving only the part which lies on one side of a clipping plane. Such a computation is useful if clipping must be done for two adjacent windows of a multiple window presentation. For example, in a flight training simulator there might be front, left, and right views all simultaneously being computed. Instead of throwing out information clipped off by the right boundary of the front window, we prefer to use it as valid output in the right window. Thus, the planes which separate the windows should be treated as dividing the polygons into two parts rather than as merely clipping them. The computed intersection of an edge with this division plane can be used as a vertex in each of the two output polygons.

The modified algorithm is shown in Figure 9. Notice that this modification also refers to points which lie "on" the division plane, thus avoiding computation of an intersection whenever the edge in question touches but does not actually cross the division boundary. The introduction of points on the division plane is necessary to avoid recomputation of intersections for polygons previously divided. When only the "visible" side polygon is saved, intersection points can be treated as "visible," but when both sides are saved, special provision for intersection points is required.

The reentrant nature of the algorithm, even in its modified form, makes it practical to think of clipping for a multitude of windows simultaneously. In a particular case of eight adjacent 45° viewing windows

covering a 360° field of view, there are four principal bounding planes between the windows. The first stage of division separates information on the first of these bounding planes, thus determining which group of four windows the polygon will go to. The second stage of division separates polygons destined for each of four pairs of windows. The third stage determines uniquely into which window the polygon will fall. Obviously, the bounding plane to be used in the third stage depends on which of the window pairs is being treated. Thus the output data from "SIDE A" and "SIDE B" of each such stage of division informs the next stage of clipping which planes to use. After the third stage, ordinary clipping is used to clip on the top, bottom, hither, and yon planes of the eight individual windows. Although there are 28 different clipping or division planes, each polygon need pass through only seven stages of clipping, a mere 16 percent increase over the six stages required for a single window; and, except for a slight increase in the storage required, the algorithm is the same.

*Acknowledgments.* We are indebted to Barry Wessler who suggested the reentrant coding approach, and to Steve Coons for his help in the early stages of formulating these ideas. Robin Forrest was extremely helpful in defining the crucial matrix formulation which permits constant and simple internal clipping planes. Ted Glaser suggested the names "hither" and "yon" for which the rest of us had long searched in vain.

## Appendix A. Negative Points

The polygon clipper described in this paper depends for its operation on the use of the homogenous representation for coordinates. We think of representing a point $X\ Y\ Z$ in real three-space by any vector of the form $[x\ y\ z\ w] = [wX\ wY\ wZ\ w]$ where $w$ is any non-zero value. The scale factor, $w$ is often chosen to be unity, but need not be. To go from homogeneous coordinates to real coordinates a division is required: $X = x/w = wX/w$, $Y = y/w = wY/w$, $Z = z/w = wZ/w$. This division turns out to be exactly the division required for perspective projection and so we have lost nothing by using the homogeneous representation. The homogeneous representation also permits one to think of any three-dimensional transformation, including rotation, translation, scaling, and perspective, as a single four-by-four matrix, vastly simplifying thought about three-dimensional motion [10].

Throughout this paper we have implicitly been assuming a positive value for $w$. Negative values for $w$ are, however, possible. In fact, the vectors $[x\ y\ z\ w]$ and $[-x\ -y\ -z\ -w]$ represent the same point in real three-dimensional space because the division by the homogeneous term, $w$ or $-w$, will remove the minus signs. Points with negative $w$ values are not necessarily behind the observer. Only if the point has an opposite sign for $z$ and $w$ is it behind the observer.

Because we have implicitly assumed positive values for $w$, the clipping limits described in this paper will automatically eliminate points with negative $w$ values even though such points might also have negative $z$ values and therefore be visible to the observer. A point with negative $w$ value cannot satisfy $-w \leq x \leq w$ regardless of its $x$ value. In this regard the clipping limits of eq. (2) are deficient, but, as we shall see, adequate for ordinary use and precisely correct if we give a more subtle interpretation to points with negative $w$ values.

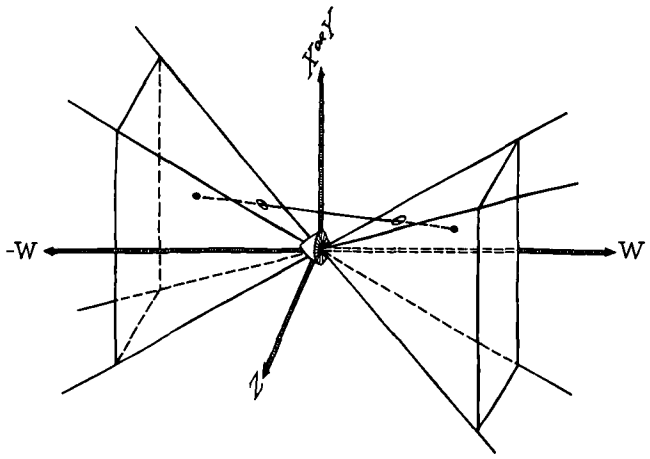
If we were clipping points only, we could quite easily modify the clipping limits of eq. (2) to account for negative $w$ values. We would simply take the magnitude of $w$ rather than $w$ itself as the limit so that points with negative $w$ values would be eliminated only if they lay outside the obvious clipping limits. In the four-dimensional space of the homogeneous coordinates, this amounts to having two volumes in which points are visible. In ordinary three-dimensional space, there is still only the single truncated pyramid described in the main text.

If we consider lines as connecting end points, however, we have a more complicated situation. What does it mean, for example, if a line begins in one of the two four-dimensional viewing regions and terminates in the other? Such a line will pass through the boundary of the visible region twice! If each such segment is presented on the screen, a single input line will result in two output lines. Figure 10 shows such a line portrayed in a two-dimensional projection of the four-dimensional homogeneous space. The two dotted portions of the line are each in the visible volume and thus will show, one at the top of the screen and one at the bottom.

The resolution of this issue is straightforward if we are willing to consider what it means to connect two points with a line segment. There are actually two possible line segments connecting two points: one is the obvious direct connection, the other is the rest of the infinite line defined by the two points. The dual output implied by Figure 10 is more palatable if one thinks of it in three dimensions as a representation of the outer line segment connecting the two points.

If we accept that lines connecting the two separate visible regions in homogeneous space represent the outer line segment between their two points, then points with negative $w$ values can serve a real purpose. If we wish to represent the line segment between two three-dimensional points $X_1Y_1Z_1$ and $X_2Y_2Z_2$, we represent each point in homogeneous coordinates as: $[w_1X_1,\ w_1Y_1,\ w_1Z_1,\ w_1]$ and $[w_2X_2,\ w_2Y_2,\ w_2Z_2,\ w_2]$. If we choose $w_1$ and $w_2$ to have the same sign, we represent the internal line segment. If we choose $w_1$ and $w_2$ to have opposite signs, we are representing the external line segment.

Fig. 10. In homogeneous coordinates the clipping volume is actually a skew pyramid. Note that this is a four-dimensional figure showing the $W$-axis explicitly. Hither clipping occurs in the $Z=0$ plane. Yon clipping occurs when $Z/W$ exceeds some limit as shown by the vertical plane passing at an angle through the origin. Two completely separate viewing volumes exist, and a line such as the one shown might have two separate visible segments.



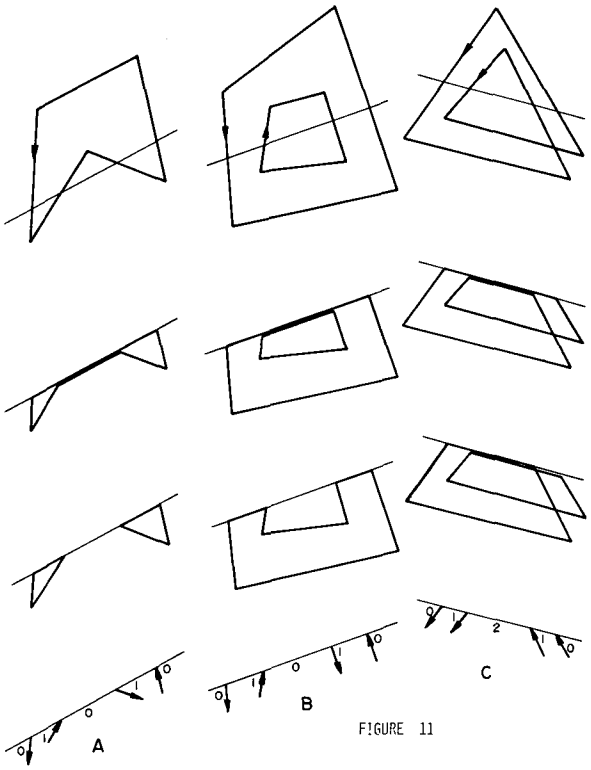

Points with negative $w$ values will not ordinarily arise in the definition of data patterned after real three-dimensional objects. Certain mathematical representations of curves, such as Coons' rational cubics, however, generate points with negative $w$ values. Such points arise when the curve being represented has "wrapped around" the point infinitely distant. For example, if one represents a hyperbola with the rational cubic technique, one sheet of the hyperbola will have positive $w$ values and the other will have negative $w$ values. In homogeneous space, the hyperbola is but a single curve somehow "wrapped around" infinity so that two parts of it appear locally. The choice of representation implied by Figure 10 is exactly right for such situations.

We have chosen in the hardware implementation of the polygon clipping algorithm to use the clipping limits as described in the main body of the paper. This implies that for most situations where only positvie $w$'s are encountered, the equipment can proceed with maximum ease. If some points are expected with negative $w$ values, the equipment will require a second pass, and will generate the second output implied by Figure 10 during such a pass. During the second pass, the matrix used is multiplied by $-1$ to invert the sign of all homogeneous vectors processed.

## Appendix B. Concave Polygons and Degenerate Boundaries

With a modest addition which will be described in this appendix, the polygon clipping algorithm can be used to divide concave, multiply-connected or self-overlapping polygons into simple convex polygons. This process can prove valuable in preparing data for hidden surface algorithms which require use of only

Fig. 11. Some examples of clipping which produce degenerate cases: (a) concave polygon; (b) multiply-connected polygon; (c) self-overlapping polygon. These cases produce the outputs in the middle row unless the extra sorting step described in Appendix B is used. Correct output with the sorting step is seen in the lower center. Direction and count of edges in the clipping plane are shown in the bottom row.




convex polygons. The methods employed are interesting in themselves because they point out the differences between the topological computations required and the geometric computations already implemented in the algorithm.

The boundary of a polygon is a sequence of edges described as one or more strings of vertices. In the case of multiply-connected polygons, this boundary has at least two vertex strings, one to describe the outline and one to describe each hole. The fact that the vertex strings are ordered places a natural direction to the boundary. In the case of multiply-connected polygons, consistency demands that holes be described in the opposite sense from the outline. A counter-clockwise vertex string inside a clockwise outline represents a hole, but a clockwise vertex string inside a clockwise boundary describes a two-layer situation.

Our task in resolving these more complicated topological cases is, given an input vertex string or strings, to produce an output vertex string or strings. These more complicated cases can be identified by looking for vertices with concave angles or for pairs of edges which cross. In either case we can simplify the polygon by dividing it along one of the offending edges. As can be seen from Figure 11, one input string may produce two output strings (11(a) and 11(c)) or two input strings

may produce one output string (11(b)). For the sake of clarity, Figure 11 shows each polygon cut along an arbitrary line rather than along an edge.

If the polygon boundary intersects the clipping plane in four or more places, there will be more than one way in which to connect those places with new edges. Each such choice of new edges will result in a different set of output vertex strings. We would prefer as output the "simplest" such set of output vertex strings, by which we mean a set which avoids degenerate overlap of edges along the clipping plane (see Figure 11(a) and (b) and within that condition has the shortest possible output strings (see Figure 11(c)).

The points where a polygon boundary intersects the clipping plane can be categorized as incoming or outgoing points according to the boundary direction at the point of intersection. In constructing new edges between these points, we must connect an outgoing point with an incoming point if we are correctly to preserve the directionality or sense of the boundary. Within this limitation we may connect any outgoing point to any incoming point and have a plausible output. How do we select which connections to make?

To avoid degenerate overlap of edges we will need to know the geometric order of the intersection points along the line of intersection. We therefore must *sort* them into this order. Starting at the first sorted point we can count $+1$ for outgoing points and $-1$ for incoming points along the line of intersection cumulatively, and thus discover the minimum number of edges which are needed in each segment of the intersection line. If the count is 0, an equal number of incoming and outgoing points has been passed, and no edge need pass by. If the count is $+1$, a single edge is needed in the direction of our progress; if $-1$, a single edge is needed in the opposite direction. If the count is 2 or more, the input polygon was self-overlapping. These counts are shown at the bottom of Figure 11. The fact that edges are not required in the segments with 0 count may be enough to resolve any question about what the output string must be (Fig. 11(a) and (b)).

Where ambiguity remains after eliminating degenerate edge cases, we wish to produce the shortest output strings. This is easily done by using Moore's algorithm [6]. The paths to use for Moore's algorithm are all of the edges not in the line of intersection, plus any path within the intersection from an outgoing point to an incoming point between which no 0 count value is found. We start Moore's path at any vertex and search for the shortest path which closes on itself. The closed part is used as the first output string, and what remains is again treated in the same way until all edges have been accounted for. This procedure will also produce correct answers in the simpler cases of Figure 11 (a) and (b).

It is interesting that the notions of simplest output string require both topological and geometric considerations. In order to avoid degenerate overlap of edges in the intersection, we have to sort the intersection points into geometric order. In order to get the shortest possible strings, we have to resort to Moore's topological method. Fortunately for our geometric sort, all of the intersection points lie in a straight line, namely the line of intersection between the plane of the polygon and the clipping plane. Were the polygon being clipped nonplanar, these intersection points might be arrayed in the clipping plane and thus be impossible to sort unambiguously. This inability to produce an unambiguous sorting order corresponds exactly to the ambiguity inherent in what is meant by the "skin" of a nonplanar polygon.

Received January 1973; revised August 1973

**References**


1. Archuleta, Michael. Computer Science Technical Memos 7002, 7102, 7105, and 7203 pertaining to Watkins' Hidden-line processing, U. of Utah, Salt Lake City, Utah, Nov. 1970, Oct. 1971, May 1971, and Mar. 1972 respectively.
2. Bouknight, W.J. A procedure for generation of three-dimensional half-toned computer graphics representations. *Comm. ACM 13*, 9 (Sept. 1970), 527–536.
3. Galimberti, R. and Montanari U. An algorithm for hidden-line elimination. *Comm. ACM 12*,4 (Apr. 1969), 206–211.
4. Loutrel, P.P. A solution to the hidden-line problem for computer-drawn polyhedra. NYU Eng. and Sci. Dept. of EE Rept. 400-167, Sep. 1967. (Also IEEE Tran. on Computers EC-19[3], Mar. 1970.)
5. Mathematical Applications Group Inc. (MAGI). 3-D simulated graphics. *Datamation* (Feb. 1968).
6. Moore, E. Shortest Path Through a Maze. Ann. of Computation Laboratory of Harvard U., Vol. 30, Harvard U. Press, Cambridge, Mass. pp. 285–292, 1959.
7. Newell, M.E., Newell, R.G., and Sancha, T.L. A new approach to the shaded picture problem. Proc. ACM Nat. Conf., Boston, Aug. 1972.
8. Newman, W.M., and Sproull, R.F. *Principles of Interactive Computer Graphics*. McGraw-Hill, New York, 1973. (See especially Chap. 13).
9. Newman, W.M. An informal graphics system based on the LOGO language. Proc. AFIPS 1973 FJCC, Vol. 42, AFIPS Press, Montvale, N.J., pp. 651–655.
10. Roberts, L.G. Homogeneous matrix representation and manipulation of N-dimensional constructs. MIT Lincoln Laboratory MS 1405, May 1965.
11. Sproull, R.F., and Sutherland I.E. A clipping divider. Proc. 1968 AFIPS FJCC, Vol. 33, AFIPS Press, Montvale, N.J., pp. 765–775.
12. Sutherland I.E. A head-mounted three-dimensional display. Proc. 1968 AFIPS FJCC, Vol. 33, AFIPS Press, Montvale, N.J., pp. 757–764.
13. Warnock, J.E. A hidden-surface algorithm for computer-generated halftone pictures. Computer Science Dept., U. of Utah, TR 4-15, June 1969.
14. Weiss, R.A. BE VISION, A package of IBM 7090 FORTRAN programs to draw orthographic views of combinations of planes and quadric surfaces. *J.ACM 13*, 2 (Apr. 1966), 194–204.
15. Zajac, E.E., and Behler, B.L. A generalized window-shield routine. UAIDE Proc. 8th Ann. Meeting, Nov. 1969, pp. 351–388.