

BACKGROUND OF THE INVENTION

1. Field of the Invention:

This invention relates to computer graphics systems and, more particularly, to arrangements for more efficiently clipping polygons to lie within windows on computer output displays.

2. History of the Prior Art:

Modern computer systems are able to process a plurality of programs at the same time. This is usually accomplished by the various programs running on a computer system essentially timesharing the facilities of central processing unit. When a plurality of programs are being run by a central processing unit at the same time, graphic output displays for each of the programs may be presented together on the computer output display in separate windows. A window in a two-dimensional computer output display is usually a rectangle which contains the graphic information for the particular program being displayed in that window. In any case, the information being displayed in a window is often less than the entire graphic presentation which is available; and it is necessary to clip the graphics image to fit within a particular output display.

Graphics figures are often described on an output display in the form of polygons. Usually the method for clipping a polygon to fit within a window requires that the lines joining the vertices of the polygon to be displayed be compared with the lines describing the window boundaries to determine which portions of the polygon lie within the window boundaries. A number of algorithms have been provided for accomplishing this comparison of lines joining the vertices of polygons with the computer boundaries. The most used of these algorithms, the Sutherland-Hodgman algorithm is described in a paper entitled "Re-entrant Polygon Clipping", Sutherland and Hodgman, Communications of the Association for Computing Machinery, Volume 17, No. 1, January, 1974. The Sutherland-Hodgman algorithm computes, for each window boundary a new polygon by clipping each of the sides of the "input"

5 polygon against a particular window boundary. This produces an "output" polygon. This "output" polygon then becomes the "input" polygon for the next window boundary, producing a new "output" polygon. This sequence is repeated until all the window boundaries are scanned and the last "output" polygon will fit in the window. The implementation of the Sutherland-Hodgman algorithm requires a very substantial amount of memory and requires a significant amount of central processing unit time in order to accomplish its objects.

10 Since so much time and memory is required by the Sutherland-Hodgman algorithm, other algorithms have been suggest. Weiler and Atherton, in an article entitled "Hidden Surface Removal Using Polygon Area Sorting", published in the proceedings of Siggraph, 1977, pages 214 - 222, present an algorithm for polygon hidden surface and hidden line removal. The algorithm
15 recursively subdivides the image into polygon-shaped windows until the depth order within the window is found. The algorithm is especially accurate but is so complex that it takes a great deal of computer time and memory to operate. Furthermore, the algorithm is not optimized for a rectangular window and, consequently, is relatively slow running.

20 Liang and Barsky, in an article entitled "An Analysis and Algorithm for Polygon Clipping", Communications of the ACM, November, 1983, Volume 26, pages 868-877, published an algorithm which speeds the operation of the Sutherland-Hodgman algorithm by approximately two times.
25 Unlike the Sutherland-Hodgman algorithm the Liang & Barsky algorithm is limited to two dimensional and rectangular windows. The Liang and Barsky algorithm computes the intersections of a line which is a side of a polygon extended infinitely with all clip planes also extended infinitely. It then uses these intersections to determine points lying off the polygonal sides but
30 necessary to be included within the clipped polygon (known as turning points or turning vertices). These points are the four corners of the clipped window which in the clipped polygon often become portions of the polygon to be presented. The Liang and Barsky algorithm, although faster than the

Sutherland-Hodgman algorithm, requires the use of a substantial number of floating point computations which are difficult and very time consuming unless the particular computer system includes a floating point co-processor. Many microprocessor based computer systems do not include such a co-processor
5 and the calculations with these machines are extremely complicated and time consuming.

Finally, Sutherland-Cohen devised a method for clipping lines only by quickly determining cases in which a particular line segment lies
10 entirely within the clipping window and should be completely drawn and those in which a particular line segment lies entirely outside of the clipping window within a particular area in which it should not be drawn; see Newman & Sproull, Principles of Interactive Computer Graphics 2d. Ed., McGraw-Hill, pp. 65-69. These are referred to as trivial acceptance and trivial rejection cases and do
15 not require the implementation of an entire intersection calculation. The Sutherland-Cohen algorithm provides a rapid method for determining these trivial acceptance and rejection cases and greatly speeds the implementation of a clipping algorithm. However, the Sutherland-Cohen algorithm does not provide a way of clipping polygons and, therefore, does not generate the
20 additional intersections to determine points lying off the polygonal sides but to be included within the clipped polygon (turning points).

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to accelerate the presentation of graphic objects on computer output displays.

5

It is another object of the present invention to increase the speed at which the vertices of a polygon may be determine and clipped to fit within a window on a computer output display.

10

These and other objects of the present invention are realized in a method in which the Sutherland-Cohen line clipping algorithm is first applied to a polygon to be clipped to determine trivial acceptance and rejection cases, then the information determined in the Sutherland-Cohen trivial acceptance and rejection tests is utilized to determine from the regions in which the vertices of polygonal side lie the existence of additional points to be included in the polygon which do not lie on the sides of the polygon but are corners of the clipping window. This is accomplished very simply without the need to compute intersections with the window boundaries external to the visible window in most cases through the use of Boolean operations using the vertices of the end-points and various look-up tables generated for the particular window.

15

20

These and other objects and features of the invention will be better understood upon a consideration of the detailed description which follows taken together with the drawings in which like elements are referred to by like designations throughout the several views.

25

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a diagram illustrating the region codes used in the Sutherland-Cohen algorithm;

5

FIGURE 2 is a diagram illustrating the meaning of turning point as utilized in the present invention;

FIGURE 3 illustrates various generic configurations for lines with respect to a clipping window which may be used in explaining the present invention;

10

Figure 4 illustrates a particular turning point which is to be generated for a particular side of a polygon with respect to a particular clipped window in carrying out this invention;

15

Figure 5 illustrates another set of turning points which are generated with respect to a particular side of a polygon in a particular clipping window in carrying out the invention; and

20

Figure 6 illustrates an ambiguous condition which must be handled in carrying out the invention.

NOTATION AND NOMENCLATURE

Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms, such as adding or comparing, which are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary or desirable in most cases in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases the distinction between the method operations in operating a computer and the method of computation itself should be borne in mind. The present invention relates to method steps for operating a computer in processing electrical or other (e.g. mechanical, chemical) physical signals to generate other desired physical signals.

The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The
5 algorithms presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these
10 machines will appear from the description given below.

DETAILED DESCRIPTION OF THE INVENTION

Referring now to Figure 1, there is shown a diagram illustrating the implementation of the present invention. This diagram illustrates the designations for the areas surrounding a visible window to be presented on an output display in the format provided by the Sutherland-Cohen algorithm. Four lines which designate a two dimensional clipping window are illustrated in Figure 1, two parallel vertical lines designated Xmin and Xmax and two parallel horizontal lines designated Ymin and Ymax. The area lying in the central portion bounded by the four lines is the area of the window to be displayed on the computer output display. The other areas are those lying above, below, to the left, and to the right of the computer display window.

The Sutherland-Cohen algorithm codes each of these areas using four binary bits. A one in the leftmost bit indicates that a point lies below the clipping window, a one in the second bit from the left indicates that the point lies to the left of the window, a one in the third bit from the left indicates that the point lies above the window, and a one in the rightmost bit indicates that the point lies to the right of the window. Consequently, the window itself is designated as 0000 and the other regions are designated as shown in Figure 1. Thus, the area above and to the left of the window is designated 0110, for example.

The Sutherland-Cohen clipping algorithm is designed to identify efficiently lines which can be trivially accepted or rejected by checking whether both vertices lie in a single particular one of these regions. For example, referring to Figure 3, there is shown a number of different individual lines which may be considered in explaining the operation of the Sutherland-Cohen algorithm. The line designated 10 which lies completely within the window area has each of its end-points lying within the 0000 region. A check by the Sutherland-Cohen algorithm which compares the areas in which each the two vertices of a line segment lie determines that this is a trivial case of a

line which should be accepted without conducting any intersection determinations or any other checks.

In like manner, the line 12 has both end-points lying within the region 0010 immediately above the window, and it may be trivially rejected after comparing the areas in which its vertices lie as requiring no intersection points to be computed by the algorithm. In like manner, the line 14 may be trivially rejected as having both end-points lying within the region 0011 above and to the right of the window.

10

However, none of the other lines illustrated in Figure 3 may be either trivially or rejected and require the use of the Sutherland-Cohen algorithm for the determination of various intersection points in order to completely define a clipped polygon. The manner in which this is accomplished using the Sutherland-Cohen algorithm is as follows. A line such as the line 16 has a first intersection with one of the extended lines defining the window boundaries determined. For example, using the algebraic definitions of the lines Y_{max} and the line 16, the intersection point 18 in Figure 3 may be computed. Then, one portion of the line 16 is discarded. The Sutherland-Cohen algorithm picks the portion to be discarded by selecting a portion which lies above, below, to the right, or to the left of the window in a particular order which is followed throughout the application of the algorithm. In the case of the line 16, that portion lying above the window is discarded; and the remainder of the line 16 is checked to determine whether it may be trivially accepted or rejected. If neither applies, a new intersection of the line 16 and the next window boundary X_{min} is determined at point 20. Once the intersection 20 is determined, the portion of the line to the left of the window is discarded; and the remainder of the line between the points 18 and 20 is tested to see if it may be trivially accepted or rejected. This portion, obviously, may be trivially accepted since both end points lie within the boundaries of the visible window in the region 0000; and the computations for clipping that particular line 16 are complete.

When applied to other lines, such as the line 22, the determination accomplished by the algorithm becomes more complex. For example, the line 22 may complete a polygon which lies partially within the window. In such a case (for example, in the case of the polygon joining the end-points of the lines 5 22 and 10 shown by the dotted lines in Figure 3), the intersection of those lines with the edges of the clip window at points 24 and 26 may be determined in a straight forward manner as explained above. However, the polygon is only completed by including within the clip window a portion of the polygon which extends to the upper left hand corner of the clip window. This point 28 does not 10 lie on any of the lines defining the polygon and must be determined from the position of the line 22 with respect to the window boundaries. There is no way to find this point using the Sutherland-Cohen algorithm since it is a line clipping algorithm. The Sutherland-Cohen algorithm proceeds to compute the various intersections of the line 22 extended with the Ymax and Xmin lines defining the clip window extended and reaches a conclusion which rejects the entire line. 15

The Liang-Barsky algorithm to some extent overcomes this problem by utilizing a scheme of determining all the intersections and assigning clip point values from the determination of those intersections. Although the 20 Liang-Barsky algorithm is twice as fast as the basic Sutherland-Hodgman algorithm, it has to compute intersections for all of the clip window boundaries before it can accomplish the assignment of the turning points; and it requires the computation of the intersections by means of floating point calculations. In many microprocessor-based computer systems, such computations are 25 extremely slow and therefore detract from the presentation of the image on the graphical output display.

The present invention offers a substantial improvement in determining the points lying off the sides of a polygon, the so-called turning 30 points, without the necessity of computing intersections with the infinitely extended window boundaries in most cases. The present invention operates as follows. First, the method recognizes that the region codes assigned by the Sutherland-Cohen algorithm each contain two one bits if they lie at the

diagonal corners of the visible window and a single one bit if they lie to the right, left, above, or below the visible window. The method uses this recognition to determine the turning points of a visible polygon.

5 The Sutherland-Cohen algorithm is first applied to determine whether a trivial accept or reject condition exists for the first line segment defining a side of a polygon to be clipped. If such a condition does not exist, then the Sutherland-Cohen algorithm determines the intersections of the sides with the edges of the visible window. If these do not occur, the information
10 provided by the Sutherland-Cohen algorithm about the regions in which the vertices lie is sorted into categories based on the beginning and ending points of the line segment. Each side is considered from a beginning point to an ending point at which the next side is considered. If the end-point of the line lies in a region in which the four bit code designation includes two one bits (i.e.,
15 an area at a corner of the window), then the corner of the window of the visible window touching that region is assigned to the visible polygon as an additional point in the polygon to be clipped, i.e., a turning point. As may be seen in Figure 1 this means that if an end-point of a side of polygon lies within the regions numbered 0110, 0011, 1001, or 1100 at the diagonal corners of the
20 visible window, then this first (so-called general rule) will apply; and the window corners closest to each of these regions are assigned as additional points for the clipped polygon.

 This general rule does not cover the cases where the end-points
25 lie within the regions immediately above, below, and to the left and right of the window. It also does not cover all turning points for lines which end in the diagonal regions. For this reason, line segments are categorized depending on the regions in which their beginning and end-points lie. If both points lie in regions above, below, and to the left and right of the window where the codes
30 for the regions have a single one bit, the sides are designated as 1-1 bit lines and treated in one manner. On the other hand, if the beginning and ending points lie in coded regions one of which has a single one bit and the other of which has a pair of one bits (a so-called 1-2 bit line), the line is treated in a

different manner. Finally, lines which have beginning and ending points both of which lie in regions the codes for which includes a pair of one bits (so-called 2-2 bit lines) are treated in a third manner.

5 It will be seen that in the case of a 1-1 bit line, if both vertices lie within the same region, then no turning point needs to be assigned; for this line will already have been trivially rejected in the first step of the Sutherland-Cohen algorithm as a trivial rejection case. However, if both vertices of a 1-1 bit side lie within different regions, then the two codes are combined in a logical
10 OR operation; and the results of that calculation are used in applying the general rule. For example, if a line starts in region 0100 and ends in 0010, such as the line 30 shown in Figure 3, then a logical OR calculation will provide the binary number 0110 which is the same number as the code for the upper left hand region. This number, using the general rule, requires the
15 assignment of the upper left hand corner of the visible window as a turning point to be included within the polygon to be described within the clipped window.

 In the case of lines with vertices falling within the 1-2 bit designation, four possibilities occur. First, if the end-point of the line lies in a 1
20 bit region and the logical AND of the two codes of the regions is not zero, then no turning point is generated. For example, a line which proceeds from the 0110 area to the 0010 area, needs to have no turning point generated because the upper left hand corner will have already been designated as a turning point
25 by a line ending at the vertex in the 0110 region under the general rule; and once a point has been designated as within a clipped polygon, it need not be so designated again in order to describe completely the polygon which lies within the clipped window.

30 However, in the case in which the end-point of the line lies in a 1 bit region and the ANDing of the codes designating the regions of the vertices provides a zero result, such as a line 32 running from the region 0110 to the region 0001 illustrated in Figure 4, a turning point 40 is generated that

corresponds to the upper right hand corner of the window. In the present invention this designation is accomplished by a look-up table.

5 If the line ends in a 2 bit region and the ANDing of the two codes does not result in zero, then the case is handled by the general rule and produces as a turning point the window corner closest to the end-point of the line.

10 If, on the other hand, the line ends in a 2 bit region and the ANDing of the two codes results in a zero, such as a line 34 beginning in the area 0001 and ending in the area 0110 as shown in Figure 5, then two turning points must be generated, one point 42 by the general rule at the upper left hand corner of the window and one point 44 at the upper right hand corner of the window. This second turning point 44 is also handled by a look-up table in
15 the present implementation of the invention.

In the case of sides designated as 2-2 bit cases, three situations occur. First, where both the beginning and ending points lie in the same region, no turning point needs to be generated since this will have been
20 handled as a trivial acceptance or rejection case by the Sutherland-Cohen algorithm. If not a trivial case and the ANDing of the two code regions for the end-points of the line does not produce a zero, the general rule is applied for the end-point of the side. Such sides might be lines running, for example, from region 0011 to region 1001 and from region 1001 to region 1100.

25 If, however, the ANDing of the codes for the two regions in which the ends of the lines lie produces a zero (for example, the lines 36 and 38 in Figure 6 running from region 0011 to an opposite diagonal region 1100), then the line may lie to one or the other side of the visible window; and an
30 ambiguous situation is produced. In this situation, the line (36 or 38 in Figure 6) is divided into two lines, by determining a so called mid-point (39 in Figure 6 for example on line 38). The present implementation of the invention then determines if this point lies in the same region as one of the two original points.

If this is the case, the corresponding original point is replaced by the mid point and a new mid point is evaluated. This operation is repeated until the midpoint lies in a region different from the end points. If or when the mid point is not in one of the regions of the end points, the present implementation of the invention
5 determines the adequate turning point by applying one of the rules previously described for the 1-1 bit or 1-2 bits case.

In implementing the look-up tables in accordance with the present invention, it has been determined that such a table is easily implemented in a
10 relatively simple manipulation. For example, to determine the turning point labelled 40 in Figure 4, the code of the area 0110 at which the line segment 32 begins is simply added in a non-Boolean arithmetic computation to the content of a look-up table indexed by the code 0001 (which defines the end point) to provide a code number 0110. Thus, it will be seen by comparison with the
15 regions of Figure 4 that the turning point at the upper right hand corner of the window will be indicated by the general rule if minus three is added to the code 0110 at the beginning vertex (the binary representation of 6). Consequently a binary stated minus three is provided in the look-up table, at the indexed position 0001, and the general rule is then applied. In like manner the look-up
20 table may hold a number plus or minus six or plus three in order to realize the turning points for the other corners or lines which variously start or end in a bit region and start or end in a 2 bit region.

A line segment beginning in code region 1100 and ending in the
25 same region 0001 in like manner uses a minus three from the look-up table to reach a turning point in code region 1001. A line segment beginning in regions 1100 or 1001 and ending in code region 0010 is indexed to and a minus six to reach turning points in regions 0110 or 0011, respectively. A line segment beginning in either code region 0011 or 0110 and ending in code region 1000
30 is indexed to and adds a plus six to reach turning points in regions 1100 or 1001, respectively. And a line segment beginning in code regions 0011 or 1001 and ending in region 0100 is indexed to and adds a plus three to reach turning points in regions 0110 or 1100, respectively.

In devising the look-up tables for the invention disclosed herein, a first look-up table is provided for each of the 2 bit codes generated by either an end-point which has a vertex ending in a 2 bit region or for a Boolean OR operation which provides a code designating such a region. The look-up table produces an X value and a Y value for each of these turning points (corners of the window) which are then added to the clipped polygon to completely define the polygon.

The second look-up table provided in implementing this invention is one in which the code for the ending vertex is provided as an offset to the address and a value to be added or subtracted from the other code for the beginning vertex to reach the value for the particular corner of the window is provided by the table. This value is either subtracted or added (depending on the sign of the number) to one of the codes, depending on the different cases, and the first look-up table is consulted under the address of the result to provide a turning point. Again, this turning point is provided as an X value and a Y value to indicate an additional point to be added to the clipped polygon.

A general algorithm for accomplishing the invention is as follows:

```
Calculate the code of the first point of the polygon.
for all subsequent points:
    Calculate the code of the point,
    Clip the line according to the two codes,
    if the line is outside the clipping region:
        Test for 1-1, 1-2 and 2-2 cases.
    endif
    Test for the general case.
    Set "first point" = "current point".
endfor
```

A specific implementation of this algorithm in the C language is provided in the following paragraphs. Not included in this implementation is the Sutherland-Cohen algorithm which is composed of the following two functions:

- Cp_move_clip() performs a simple coding of the first point of the polygon. This function returns SEGM if the point is inside the clipping region, NOSEGM if the point is outside.

- Cp_draw_clip() performs the clipping of a line coded in the two structures Cp_move and Cp_draw which respectively represent the start and end point of one polygon edge. Cp_draw_clip provides the following information:

- The returned status, NOSEGM, SEGM, SEGM | CLIP, which represents the visibility characteristic of the edge,
- M_move and M_draw, which are the computed codes of the start and end point of the edge, respectively.
- Cp_move and Cp_draw contain the clipped line coordinates at the end of the algorithm.

```

15 typedef struct {
    float x;          /* x coordinate (could be int).*/
    float y;          /* y coordinate (could be int) */
    } Upoint;          /* User point */

20 typedef struct {
    Upoint Point;      /* The point's coordinates */
    int code;          /* The code computed for this point */
    } Ppoint;          /* Internal representation */

#define MAXTEST 1

#define NOSEGM 0      /* The line is rejected */
#define SEGM 1        /* The line is visible (even partially) */
25 #define CLIP 2       /* The line has been "clipped" (left) */
#define TWOBITS 0x100 /* A flag to indicate a two bits code */

Ppoint Cp_move;      /* The start point of the line */
int M_code;

30 Ppoint Cp_draw;     /* The end point of the line */
int D_code;

Ppoint C_exchange;

/* These are two look_up tables */
/* used in finding the turning point */
/* in the case 1-2. They should be */
/* modified with the regions' codes.*/
35 int Tcc[16] = {0,-3,-6,1,3,0,1,0,6,1,0,0,1,0,0,0};
int Vpt[16] = {-1,-1,-1,2,-1,-1,3,-1,-1,1,-1,-1,0,-1,-1,-1};
/* Tcc is used to compute a correct */
/* offset, while Vpt gives an index */
40 /* in the Viewport array, for the */
/* turning coordinates */

Upoint Viewport[4]; /* Viewport coordinates ll,lr,ur,ul */

```

The function `CP_space_code()` returns the code associated with a given point. The returned code can be a single value, or an union between a single value and a flag that indicates a two bits code.

```

5      CP_space_code(point_to_code)
      Upoint      *point_to_code;

      {
          if (point_to_code->x < Lower_left.x) {
              if (point_to_code->y < Upper_right.y) return (6 | TWOBITS);
10             if (point_to_code->y < Lower_left.y) return (12 | TWOBITS);
              return (4);
          }
          if (point_to_code->x > Upper_right.x) {
              if (point_to_code->y < Upper_right.y) return (3 | TWOBITS);
15             if (point_to_code->y < Lower_left.y) return (9 | TWOBITS);
              return (1);
          }
          if (point_to_code->y < Upper_right.y) return (2);
          if (point_to_code->y < Lower_left.y) return (8);
20             return (0);
      }

```

The `CP_2D_polygon_clip()` function accepts an array of vertices as input and clips the vectors against a rectangular viewport. Turning points are generated
25 when necessary to keep the polygon structure and ensure a correct visualization. The function generates the resulting polygons in an output array of points.

```

30      CP_2D_polygon_clip(nin,in,nout,out)
      int      nin;
      Upoint      *in;
      int      *nout;
      Upoint      *out;
      {

35      register int      i, j, k;
      register Ppoint *pt_Cp_move      = &Cp_move;
      register Ppoint *pt_Cp_draw      = &Cp_draw;
      /*
      * Temporary data used in the case of 2-2 bits.
40      */
      Upoint      Cp_t_move;
      Upoint      Cp_t_draw;
      Upoint      Cp_A_point;
      int      A_code;

45

      /*
      * Be sure to close the polygon.
      */
50      k = nin + 1;
      in[nin] = in[0];
      /*
      * Compute the first point' status.
      * If visible, then store the first point in the output array.
55      */
      if (CP_move_clip(in)) {
          out[*nout] = pt_Cp_move->Point;
          *nout += 1;
      }

```

```

    }
    /*
    * Next polygon's points... We build a vector from the "move"
    * point to the "draw" point.
    5  * Clip the line with a standard 2D line clipping method.
    */
    for (i = 1; i < k; i++) {
        j = CP_draw_clip(in + i);
    }
    /*
    10 * If the line is visible, then store the computed point(s), and
    * jump to the general case.
    */
    Cp_move.Code = D_code;
    if (j & SEGM) {
    15     if (j & CLIP) {
        out[*nout] = pt_Cp_move->Point;
        *nout += 1;
    }
    out[*nout] = pt_Cp_draw->Point;
    20 *nout += 1;
    /*
    * Here the line has been rejected... See what we can do!
    */
    } else {
    25 /*
    * Begin with a 2 bits end point.
    */
    if (D_code & TWOBITS) {
        if (!(M_code & D_code)) {
    30 /*
    * If the start point is also a 2 bits... Need some more to
    * make a decision!
    */
    if (M_code & TWOBITS) {
    35     j = 1;
    Cp_t_move = pt_Cp_move->Point;
    Cp_t_draw = pt_Cp_draw->Point;
    while (j) {
    40     Cp_A_point.x = (Cp_t_move.x + Cp_t_draw.x) / 2.;
    Cp_A_point.y = (Cp_t_move.y + Cp_t_draw.y) / 2.;
    A_code = Cp_space_code(&Cp_A_point);
    if (A_code & TWOBITS) {
        if (A_code == D_code) Cp_t_draw = Cp_A_point;
        else if (A_code == M_code) Cp_t_move = Cp_A_point;
    45     else j = 0;
    } else {
        if (A_code & D_code) A_code = D_code + Tcc[M_code &
            0xf];
        else A_code = M_code + Tcc[D_code &
    50     0xf];
        j = 0;
    }
    }
    } else {
    55 /*
    * This is for a 1 bit start point (2 bits end point).
    */
    A_code = D_code + Tcc[M_code];
    }
    out[*nout] = Viewport[Vpt[A_code & 0xf]];
    60 *nout += 1;
    }
    } else {
    65 /*
    * Here we have a 1 bit end point.
    */
    if (M_code & TWOBITS) {
        if (!(M_code & D_code)) D_code = M_code + Tcc[D_code];
    } else {
    70     D_code = M_code;
    if (Tcc[D_code] == 1) D_code = TWOBITS;
    }
    }
    }

```




CLAIMS

1. A method for clipping a polygon for display on a computer output display, said polygon comprising beginning and ending vertices connected by a substantially straight line forming a side of said polygon, said side being clipped for display in a substantially rectangular display window having four corners, said method comprising the steps of:

assigning beginning and ending codes to the beginning and ending vertices, each of the codes indicating whether each of the vertices lies in a region that is above, below, to the right of, to the left of, above and to the right of, above and to the left of, below and to the right of, below and to the left of, or inside said display window, each of the codes comprising at least four bits, a first bit indicating whether the vertex is located in one of the three regions below said display window, a second bit indicating whether the vertex is located in one of the three regions left of said display window, a third bit indicating whether the vertex is located in one of the three regions above said display window, and a fourth bit indicating whether the vertex is located in one of the three regions right of said display window;

including said side for display if both of said vertices lie in the same region inside said display window;

excluding said side for display if both of said vertices lie in the same region outside said display window;

including a portion of said side for display if said vertices lie in different regions and said side intersects with one or two edges of said display window, said portion being determined by clipping one or both ends of said side which lie outside the display window;



if said side does not intersect with any edge of said display window, said vertices lie in different regions and said different regions wherein said beginning and ending vertices lie comprise two of four side regions, said side regions being the four regions that are above, below, to the right and to the left of said display window, determining a first result by performing a Boolean OR operation on the assigned codes of the beginning and ending vertices and replacing said side with a first corner, said first corner being the corner of said display window closest to a first derived region, said first derived region having an assigned code equal to the first result;

if said side does not intersect with any edge of said display window, said vertices lie in different regions and at least one of said vertices lies in one of four corner regions, said corner regions being the four regions that are above and to the left, above and to the right, below and to the left and below and to the right of said display window, determining a second result by performing a Boolean AND operation on the assigned codes of the beginning and ending vertices, and

(i) if said second result is not equal to zero and said ending vertex lies in one of the four corner regions, replacing said side with a second corner, said second corner being the corner of said display window closest to the region of the ending vertex,

(ii) if said second result is equal to zero, and if said ending vertex lies in one of said side regions, deriving a second derived region and replacing said side with a third corner, said third corner being the corner of said display window closest to the second derived region,

(iii) if said second result is equal to zero, and if said beginning vertex lies in one of said side regions, determining a replacement edge and replacing



said side with the replacement edge, said replacement edge having a start point and a stop point and being an edge of said display window, said start point being the corner of said display window closest to said second derived region, said stop point being the corner of said display window closest to the region of the ending vertex,

(iv) if said second result is equal to zero, and if said beginning vertex lies in one of said corner regions and said ending vertex lies in a different one of said corner regions, deriving a third derived region and replacing said side with a fourth corner, said fourth corner being the corner of said display window closest to the third derived region.

2. The method for clipping a polygon for display as set forth in claim 1, wherein said second derived region is derived as follows:

determining first and second lookup regions, said first lookup region being the region containing the ending vertex, said second lookup region being the region containing the beginning vertex, and

setting the first derived region to be equal to the sum of an arithmetic increment and the code for said second lookup region, said arithmetic increment being looked up from a table using said first lookup region, said table comprising the assigned codes for said first lookup region and a plurality of corresponding arithmetic increments.

3. The method for clipping a polygon for display on a computer output display as set forth in claim 2, wherein said arithmetic increments corresponding to said first lookup region are -3, -6, +3 and +6.

4. The method for clipping a polygon for display on



- 23 -

a computer output display as set forth in claim 1, wherein said third derived region is resolved to one of two opposing said corner regions by applying the method of the present invention to a truncated version of said side generated by a mid-point subdivision process.

5. The method of clipping polygons for display on a computer output display substantially as hereinbefore described with reference to the accompanying drawings.
