Some Improvements to a Parametric Line Clipping Algorithm

You-Dong Liang†, Brian A. Barsky, Mel Slater‡

Computer Science Division
University of California
Berkeley, California 94720
U.S.A.

This paper presents an improved version of our earlier[1, 2] parametric line clipping algorithm.

There are two types of improvements. The first involves an initial trivial reject test based only on comparisons, and the second is the addition of a test to avoid unnecessary computation when the result will not affect the parametric value of the intersection point. For each dimension, the mathematics are derived along with geometrical interpretations and the algorithm is designed. Finally, a performance test is conducted. Both the original and improved versions of our algorithm are compared to the traditional Sutherland-Cohen[3, 4] clipping algorithm.

0. Introduction

Clipping is a basic and important problem in computer graphics. It is the process of removing that portion of an image that lies outside a region called the *clip window* or *visible region*.[5, 6] One way to classify clipping algorithms is according to the type of

---

† Permanent address: Department of Mathematics, Zhejiang University, Hangzhou, Zhejiang People's Republic of China

‡ Permanent address: Department of Computer Science, QMW University of London, Mile End Road, London E1 4NS, U.K.

1 You-Dong Liang and Brian A. Barsky, ''A New Concept and Method for Line Clipping,'' *ACM Transactions on Graphics,* Vol. 3, No. 1, January, 1984, pp. 1-22.

2 You-Dong Liang and Brian A. Barsky, ''Introducing A New Technique for Line Clipping,'' pp. 548-559 in *Proceedings of the International Conference on Engineering and Computer Graphics*, Beijing, 27 August - 1 September 1984. Also in *Journal of Zhejiang University* Special Issue on Computational Geometry, 1984, pp.1-12.

3 James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley Publishing Company, 1990. Second Edition.

4 William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

5 James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley Publishing Company, 1990. Second Edition.

6 William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

primitive upon which they operate; the type of clipping to be considered in this paper is line clipping.

In the early time of computer graphics, the Sutherland/Cohen[7, 8] line clipping algorithm and its coding technique was in common use. In 1984, we proposed a family of efficient line clipping algorithms that are based on a strict mathematical form.[9, 10]

Our line clipping algorithm employed a parametric representation of the line segment to be clipped, used minimum and maximum calculations to determine the parametric values corresponding to the endpoints of the visible line segment, and developed several new definitions for trivial reject to speed up the approach. A similar approach was independently adopted by Cyrus and Beck[11] although that work did not develop the trivial reject cases that are so important for efficiency.

The method for line clipping that we developed describes clipping in an exact and mathematical form. The basic ideas form the foundation for a family of algorithms for two-, three-, and four- (homogeneous coordinates) dimensional line clipping.

One of the features of our algorithm is that it is independent of dimension. The identical coded procedure can be used for clipping in any dimension. The applicability of this algorithm for different dimensions only requires a different calling sequence for this procedure. This feature is both theoretically interesting and practically convenient. For this reason, the computation times for our algorithm are fairly constant over the different dimensions.

At SIGGRAPH'87, Nicholl, Lee, and Nicholl[12] presented a very efficient two-dimensional line clipping algorithm that is based on a case-by-case approach and which improved upon the existing line clipping algorithms. However, their algorithm is only applicable in two dimensions.

In our algorithm, the line segment to be clipped is mapped into a parametric representation. >From this, a set of conditions is derived that describes the interior of the clipping region. Observing that these conditions are all of similar form, they are rewritten such that the solution to the clipping problem is reduced to a simple max/min expression.

---

[7] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley Publishing Company, 1990. Second Edition.

[8] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

[9] You-Dong Liang and Brian A. Barsky, ''A New Concept and Method for Line Clipping,'' *ACM Transactions on Graphics,* Vol. 3, No. 1, January, 1984, pp. 1-22.

[10] You-Dong Liang and Brian A. Barsky, ''Introducing A New Technique for Line Clipping,'' pp. 548-559 in *Proceedings of the International Conference on Engineering and Computer Graphics*, Beijing, 27 August - 1 September 1984. Also in *Journal of Zhejiang University* Special Issue on Computational Geometry, 1984, pp.1-12.

[11] Mike Cyrus and Jay Beck, ''Generalized Two- and Three-Dimensional Clipping,'' *Computers and Graphics,* Vol. 3, No. 1, 1978, pp. 23-28.

[12] Tina M. Nicholl, D. T. Lee, and Robin A. Nicholl, ''An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis,'' pp. 253-262 in *SIGGRAPH '87 Conference Proceedings*, ACM, Anaheim, July 27-31, 1987.

Another advantage of our algorithm is that line segments that are invisible are quickly rejected. In addition, although the three-dimensional and homogeneous coordinate algorithms are presented for perspective viewing volumes, it should be noted that it is equally appropriate for non-perspective viewing volumes, and more generally, for any convex visible region.

Furthermore, the structure of the algorithm is such that the conditions describing the interior of the clipping region are all of similar form; consequently, these visibility calculations could be processed in parallel.

In this paper, we describe an improved version of this algorithm. There are three types of improvements. First, we introduce an initial test for trivial reject, that is, based only on comparisons between the line endpoints and the clipping boundaries. Second, the original algorithm employed two different kinds of tests to accomplish subsequent trivial rejects; in the improved version, we add a third type of test for trivial reject. Third, tests are added so as to avoid unnecessary computation when the result will not affect the parametric value of the intersection point.

For each dimension, the mathematics are discussed and the algorithm is designed, and then a performance test is conducted. Both the original and improved versions of our algorithm are compared to the traditional Sutherland-Cohen[13, 14] clipping algorithm. Using randomly generated data, the original version of our algorithm showed a 25% to 62% improvement compared to the Sutherland-Cohen algorithm and the improved version of our algorithm had a 46% to 68% improvement over the Sutherland-Cohen algorithm.

1. Setting up the Problem in Two, Three, and Four Dimensions

In two-dimensional clipping, lines are clipped against a two-dimensional region called the *clip window* (Figure 1-1). In particular, let the equations of the boundary lines be

$$\begin{aligned} x &= x_{left} \\ x &= x_{right} \\ y &= y_{bottom} \\ y &= y_{top} \end{aligned} \tag{1.1}$$

representing the left, right, bottom, and top boundaries, respectively. Using this notation, a two-dimensional point (x , y) is inside the window if and only if

$$\begin{aligned} x_{left} \le x \le x_{right} \\ y_{bottom} \le y \le y_{top} \end{aligned} \tag{1.2}$$

---

[13] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, 1990. Second Edition.

[14] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979. Second edition.

Figure 1-1: The interior of the clip window.

For three-dimensional clipping, lines are clipped against a viewing pyramid. In this case, the coordinate system is transformed to a right-pyramid coordinate system where the viewing volume is a right pyramid, as shown in Figure 1-2a. The equations of the four bounding planes of the right pyramid are:

$$\begin{aligned} z &= -x \\ z &= x \\ z &= -y \\ z &= y \end{aligned} \tag{1.3}$$

representing the left, right, bottom, and top bounding planes, respectively. >From this, a three-dimensional point (x , y , z) is inside the viewing volume if and only if

$$\begin{aligned} -z \le x \le z \\ -z \le y \le z \end{aligned} \tag{1.4}$$

and $z \ge 0$ is implied.

This can be extended to cover the case of a finite viewing volume. Here, the viewing pyramid is truncated by hither and yon clipping planes to form a frustum of vision, as shown in Figure 1-2b. Note that the location of these planes is completely independent of the position of the picture plane. This requires the addition of constraints for z. If the hither and yon clipping planes are $z=h$ and $z=f$, respectively, then these constraints are:

$$h \le z \le f \tag{1.5}$$

In the case of four-dimensional clipping with perspective depth transformation, a homogeneous coordinate system is employed. Clipping is performed after the application of the perspective transformation, but prior to the perspective depth division. This is called **"clipping in homogeneous coordinates"**. The clipping limits in this formulation are:

$$\begin{aligned} -w \le x \le w \\ -w \le y \le w \\ 0 \le z \le w \end{aligned} \tag{1.6}$$

Figure 1-2: Viewing volumes for three-dimensional clipping.
(a) The right pyramid viewing volume (left).
(b) The frustum of vision (right).

2. Two-Dimensional Line Clipping Algorithm

2.1. Explanation

Recall the two-dimensional clipping limits given by inequalities (1.2):

$$\begin{aligned} x_{left} \le x \le x_{right} \\ y_{bottom} \le y \le y_{top} \end{aligned} \tag{2.1}$$

The line segment to be clipped is mapped into a parametric representation as follows. Recall that our notation for the endpoints of the line segment is $\mathbf{V}_0$ and $\mathbf{V}_1$ with coordinates (x_0, y_0) and (x_1, y_1), respectively. Then the parametric representation of the line is given by

$$\begin{aligned} x = x_0 + \Delta x\ t \\ y = y_0 + \Delta y\ t \end{aligned} \tag{2.2}$$

where

$$\begin{aligned} \Delta x = x_1 - x_0 \\ \Delta y = y_1 - y_0 \end{aligned} \tag{2.3}$$

The parametric values corresponding to $\mathbf{V}_0$ and $\mathbf{V}_1$ are $t=0$ and $t=1$, respectively. As t varies from $t=0$ to $t=1$, the line segment $\mathbf{V}_0\mathbf{V}_1$ is traced out from $\mathbf{V}_0$ to $\mathbf{V}_1$; allowing $-\infty < t < \infty$ generates the line of infinite extent.

Substituting the parametric representation given by equations (2.2) and (2.3) into inequalities (2.1) yields the following conditions for the part of the extended line that is inside the interior of the clip window:

$$\begin{aligned} -\Delta x\ t \le x_0 - x_{left} \quad &\text{and} \quad \Delta x\ t \le x_{right} - x_0 \\ -\Delta y\ t \le y_0 - y_{bottom} \quad &\text{and} \quad \Delta y\ t \le y_{top} - y_0 \end{aligned} \tag{2.4}$$

Note that these conditions (2.4) are inequalities describing the *interior* of the clip window rather than equations defining its *boundary*.

Observe that inequalities (2.4) are all of similar form; thus, they can be rewritten as

$$p_i\, t \le q_i \text{ for } i{=}1,\ 2,\ 3,\ 4 \tag{2.5}$$

where the following notation is used:

$$\begin{aligned} p_1 &= -\Delta x & q_1 &= x_0 - x_{left} \\ p_2 &= \Delta x & q_2 &= x_{right} - x_0 \\ p_3 &= -\Delta y & q_3 &= y_0 - y_{bottom} \\ p_4 &= \Delta y & q_4 &= y_{top} - y_0 \end{aligned} \tag{2.6}$$

To establish a geometrical interpretation of these four inequalities, extend each of the four boundary line segments of the clip window to be a line of infinite extent. Each of these lines divides the plane into two regions; we define the *visible side* to be that side on which the clip window lies, as shown in Figure 2-1.

Figure 2-1: The visible side of a boundary line.

In this way, the clip window can be defined as that region that is on the visible side of all the boundary lines. From this, it can be seen that each of the inequalities (2.5) corresponds to one of these boundary lines (left, right, bottom, and top, respectively), and describes its visible side.

In the improved version of the algorithm, a trivial reject test is performed prior to any other computation. The two endpoints of the line segment are checked against each boundary line in turn to see if they both lie on the invisible side of the boundary line. If they do, for any boundary, then the line segment is trivially rejected as invisible, and no further computation needs to be performed. If the line segment is not trivially rejected against any boundary, then the remainder of the algorithm is based on the following analysis.

Extend $\mathbf{V}_0\mathbf{V}_1$ to be a line of infinite extent. Then each inequality provides the range of values of the parameter t for which this extended line is on the visible side of the corresponding boundary line. Furthermore, the particular parametric value for the point of intersection is $t{=}q_i/p_i$. Also, the sign of q_i indicates on which side of the corresponding boundary line the point $\mathbf{V}_0$ lies. Specifically, if $q_i \ge 0$, then $\mathbf{V}_0$ is on the visible side of the boundary line (inclusive), and if $q_i < 0$, then $\mathbf{V}_0$ is on the invisible side.

Considering the p_i's in equation (2.6), it is clear that each can be negative, positive, or zero. If p_i is negative, the i'th inequality becomes

$$t \ge q_i/p_i \tag{2.7}$$

Observing that the range of parametric values for which the extended line is on the visible side of the corresponding boundary line is at a minimum at the point of intersection, the direction defined by $\overrightarrow{\mathbf{V}_0\mathbf{V}_1}$ is from the invisible side to the visible side of the boundary line.

Analogously, if p_i is positive, the i'th inequality becomes

$$t \leq q_i / p_i \tag{2.8}$$

Since the parametric range is at a maximum at the point of intersection, the direction defined by $\overrightarrow{\mathbf{V}_0\mathbf{V}_1}$ must be from the visible side to the invisible side of the corresponding boundary line.

Finally, if p_i is zero, the i'th inequality becomes

$$0 \leq q_i \tag{2.9}$$

Note that this is now independent of t. Thus, the inequality is satisfied for all t if $q_i \geq 0$, and there is no solution if $q_i < 0$. Geometrically, if p_i is zero, there is no point of intersection of the extended line with the corresponding boundary line; rather, they are parallel. Furthermore, if $q_i \geq 0$, then in this case the entire extended line is on the visible side of the boundary line (inclusive); if $q_i < 0$, then it is entirely on the invisible side of the line. In the former case, there may or may not be a visible segment depending on where $\mathbf{V}_0$ and $\mathbf{V}_1$ lie on the extended line. However, the latter case cannot have a visible segment since the extended line lies outside the visible region and hence this is a trivial reject case. All these cases are summarized in the flowchart shown in Figure 2-2.

Figure 2-2: Flowchart showing the geometrical interpretations for the p_i and q_i values.

Now, the intersection of the four inequalities gives the range of values of the parameter t for which the extended line is inside the clip window. However, the line segment to be clipped ($\mathbf{V}_0\mathbf{V}_1$) is only part of the extended line and is described by $0 \leq t \leq 1$. Hence, the solution to the two-dimensional clipping problem is equivalent to the inequalities (2.5) under the condition $0 \leq t \leq 1$.

Solving these inequalities is actually a max/min problem, which can be seen as follows. Recall that $t \geq q_i/p_i$ for all i such that $p_i < 0$, and that $t \geq 0$. Thus,

$$t \geq max\,(\{q_i/p_i \,|\, p_i < 0,\ \ i = 1,\ 2,\ 3,\ 4\} \cup \{0\}) \tag{2.10}$$

Analogously, $t \leq q_i/p_i$ for all i such that $p_i > 0$, and $t \leq 1$. Thus,

$$t \leq min\,(\{q_i/p_i \,|\, p_i > 0,\ \ i = 1,\ 2,\ 3,\ 4\} \cup \{1\}) \tag{2.11}$$

Finally, if $p_i = 0$ in the i'th inequality for some i, then there are two possibilities. If $q_i \geq 0$, then there is no useful information to be gleaned from this inequality, and this inequality can be discarded. If $q_i < 0$, then this is a trivial reject case, and the clipping problem is solved with no further computation needed.

The righthand sides of equations (2.10) and (2.11) are the values of the parameter t corresponding to the beginning and end of the visible segment, respectively (assuming there is a visible segment).

Denoting these parametric values as t_0 and t_1,

$$\begin{aligned} t_0 &= max\,(\{q_i/p_i \,|\, p_i < 0,\ \ i = 1,\ 2,\ 3,\ 4\} \cup \{0\}) \\ t_1 &= min\,(\{q_i/p_i \,|\, p_i > 0,\ \ i = 1,\ 2,\ 3,\ 4\} \cup \{1\}) \end{aligned} \tag{2.12}$$

If there is a visible segment, it corresponds to the parametric interval

$$t_0 \leq t \leq t_1 \tag{2.13}$$

Hence, a necessary condition for a line segment to be at least partially visible is

$$t_0 \leq t_1 \tag{2.14}$$

This is not a sufficient condition because it ignores the possibility of a trivial reject due to $p_i = 0$ with $q_i < 0$. Nonetheless, this yields a sufficient condition for rejection; specifically, if $t_0 > t_1$, then this is another reject case. The algorithm checks if $p_i = 0$ with $q_i < 0$ or if $t_0 > t_1$, in which case the line segment is immediately rejected without further computation.

The algorithm initializes t_0 and t_1 to 0 and 1, respectively. Then each boundary line is considered successively. In the case of entering the visible region ($p_i < 0$), t_0 is to be determined. Since $p_i < 0$, should $q_i \geq 0$, then the ratio q_i/p_i would be negative and hence not affect the determination of t_0. Figure 2-3(a) shows that if a trivial reject test is performed before this main iteration, as suggested earlier, then it will never be the case that $q_i < p_i < 0$. For, if $q_i < p_i < 0$ then the quotient q_i/p_i would exceed unity, so that both endpoints of the line segment would be on the invisible side of the boundary. Thus, only if $q_i < 0$ is the ratio q_i/p_i actually computed. The quotient is first compared to t_1, for if it is greater, then t_0 will exceed t_1 and this must be a reject case; otherwise, it is compared to t_0, for if it is greater, then t_0 must be updated with this new value.

Figure 2-3(a) A line from the invisible to visible side of a clip boundary

In the case of exiting the visible region ($p_i > 0$), t_1 is to be determined. Since $p_i > 0$ should $q_i \geq p_i$, then the ratio q_i/p_i would be at least unity; however, since the value of t_1 does not exceed unity, this value of the ratio would not affect the determination of t_1. Again, Figure 2-3(b) shows that it will never be the case that $q_i < 0$, for if this is the case then the quotient q_i/p_i would be negative, so that both endpoints of the line would be on the invisible side of the boudnary. Thus only if $q_i > 0$ is the ratio q_i/p_i actually computed. The quotient is first compared to t_0, and if it is less, then this is a reject case; otherwise it is compared to t_1, and if it is less, then t_1 is updated.

Figure 2-3(b) A line from the visible to invisible side of a clip boundary

Finally, if $p_i = 0$, and $q_i < 0$, then this is a reject case. At the last stage of the algorithm, if the line segment was not rejected, the parametric values t_0 and t_1 are used to compute the corresponding points. However, if $t_0 = 0$, then the endpoint is $\mathbf{V}_0$ and need not be computed; similarly, if $t_1 = 1$, then the endpoint is $\mathbf{V}_1$.

The geometric meaning of this process is that the line segment is "squeezed" down by considering where the extended line intersects each boundary line. More specifically, each endpoint of the given line segment $\mathbf{V}_0\mathbf{V}_1$ is used as an initial value for an endpoint of the visible segment $\mathbf{V'}_0\mathbf{V'}_1$. Then the point of intersection of the extended line with each boundary line is considered. (This visibility calculation with respect to each boundary line corresponds to an invocation of the "clipt" procedure in the algorithm.)

Since there has been an initial trivial reject test, the intersection point cannot be further along the line ($q_i < p_i < 0$) than $\mathbf{V}_1$, since this can only happen when the line segment terminates before the line enters the visible region. Hence, the line segment terminates on the visible side of the boundary line. For this situation, the point of intersection is compared to $\mathbf{V}_0$. Should $\mathbf{V}_0$ be further along the line than the point ($q_i \geq 0$), then the line segment starts on the visible side of the boundary line; thus, no further computation would be necessary with respect to this boundary line to determine $\mathbf{V'}_0$. For this reason, the algorithm checks to see if the point of intersection is further along the line than $\mathbf{V}_0$ ($q_i < 0$), that is, to see if the line segment starts on the invisible side of the boundary line. If this is the case then $\mathbf{V'}_0$ will be further along the line than $\mathbf{V}_0$, and needs to be determined. In this case, the point of intersection is first compared to $\mathbf{V'}_1$. If the point is further along the line, then $\mathbf{V'}_1$ (and thus $\mathbf{V'}_0\mathbf{V'}_1$) must be on the invisible side of the line. In this case, the line segment terminates before the line enters the visible region, and thus the line segment is rejected. Otherwise, the point of intersection is compared to $\mathbf{V'}_0$, and if the point is further along the line, then $\mathbf{V'}_0$ is moved up to this point.

On the other hand, if the direction is from the visible side to the invisible side of the line, ($p_i > 0$), then this point of intersection is first compared to $\mathbf{V}_0$. $\mathbf{V}_0$ cannot be further along the line than the point of intersection ($q_i < 0$) since such a line segment would earlier have been rejected. Hence, the line segment begins on the visible side of the boundary line. For this situation, $\mathbf{V}_1$ is compared to the point of intersection. Should this point be further along than $\mathbf{V}_1$ ($q_i \geq p_i > 0$), then the line segment terminates on the visible side of the boundary line; thus, no further computation would be necessary with respect to this boundary line to determine $\mathbf{V'}_1$. For this reason, the algorithm examines $\mathbf{V}_1$ to see if it is further along than the point of intersection ($0 \le q_i < p_i$), that is, to check that the line segment terminates on the invisible side of the boundary line. If this is the case then $\mathbf{V'}_1$ will precede $\mathbf{V}_1$ along the line, and needs to be determined. In this case, the point of intersection is first compared to $\mathbf{V'}_0$. If $\mathbf{V'}_0$ is further along the line than the point, then $\mathbf{V'}_0$ (and thus $\mathbf{V'}_0\mathbf{V'}_1$) must be on the invisible side of the boundary line. In this case, the line segment does not begin until after the line exits the visible region, and thus the line segment is rejected. Otherwise, the point of intersection is compared to $\mathbf{V'}_1$, and if $\mathbf{V'}_1$ is further along the line than the point, then $\mathbf{V'}_1$ is moved down to this point.

Finally, if the extended line is parallel to the boundary line and on its invisible side, the line segment is rejected. At the end of the algorithm, if the line segment was not rejected, $\mathbf{V'}_0$ and $\mathbf{V'}_1$ are the required the endpoints of the visible segment. One programming detail is that assuming $\mathbf{V'}_0$ overwrites $\mathbf{V}_0$, then $\mathbf{V'}_1$ must be assigned prior to $\mathbf{V'}_0$ since the computation of $\mathbf{V'}_1$ is dependent on $\mathbf{V}_0$.

On a final mathematical note, although it is not necessary for the algorithm, inequality (2.14) can be modified to be a necessary and sufficient condition for a line segment to be at least partially visible. This requires the inclusion of the p_i=0 case (line segment parallel to the corresponding boundary line). This case can be combined with that for p_i<0 if q_i/p_i is defined as follows:

$$q_i/p_i = \begin{cases} -\infty & \text{if } p_i = 0 \text{ and } q_i < 0 \\ +\infty & \text{if } p_i = 0 \text{ and } q_i \ge 0 \end{cases} \tag{2.15}$$

Then the necessary and sufficient condition is

$$t_0 \le t_1^* \tag{2.16}$$

where

$$t_1^* = min\,(\{q_i/p_i \mid p_i \ge 0,\ \ i = 1,\ 2,\ 3,\ 4\} \cup \{1\}). \tag{2.17}$$

2.2. Algorithm

The improved algorithm to perform two-dimensional line clipping is as follows:

```
(* Clip 2-d line segment with endpoints (x0, y0) and (x1, y1).            *)
(* The clip window is xleft <= x <= xright and ybottom <= y <= ytop.       *)

   procedure clip2d (var x0, y0, x1, y1: real; var visible: Boolean);
   label 1;
   var t0, t1: real;
      deltax, deltay: real;

         function clipt ((* value *) p, q: real; var t0, t1: real): Boolean;
            var
               r : real;
               accept : Boolean;
            begin (* clipt *)
               accept := true;
```

```
      if p < 0 then begin (* entering visible region, so compute t0 *)
         if q < 0 then begin (* t0 will be nonnegative, so continue *)
            r := q / p;
            if r > t1 then accept := false (* t0 will exceed t1, so reject *)
            else if r > t0 then t0 := r (* t0 is max of r's *)
         end (* t0 will be nonnegative, so continue *)
      end (* entering visible region, so compute t0 *)

      else if p > 0 then begin (* exiting visible region, so compute t1 *)
         if q < p then begin (* t1 will be <= 1, so continue *)
            r := q / p;
            if r < t0 then accept := false (* t1 will be <= t0, so reject *)
            else if r < t1 then t1 := r (* t1 is min of r's *)
         end (* t1 will be <= 1, so continue *)
      end (* exiting visible region, so compute t1 *)

      else (* p=0 *) if q < 0 then accept := false; (* line parallel and outside *)

      clipt := accept
   end (* function clipt *);

begin (* clip2d *)
    if (((x0 < xleft) and (x1 < xleft)) or
      ((x0 > xright) and (x1 > xright)) or
        ((y0 < ybottom) and (y1 < ybottom)) or
        ((y0 > ytop) and (y1 > ytop))) then
    begin (* sc trivial reject *)
      visible := false;
      goto 1;
   end; (* sc trivial reject *)

   begin (* not trivial reject *)
       visible := false;
     t0 := 0;
     t1 := 1;
     deltax := x1 - x0;

     if clipt (- deltax, x0 - xleft, t0, t1) (* left *) then
        if clipt (deltax, xright - x0, t0, t1) (* right *) then begin
           deltay := y1 - y0;
           if clipt (- deltay, y0 - ybottom, t0, t1) (* bottom *) then
              if clipt (deltay, ytop - y0, t0, t1) (* top *) then begin (* compute coordinates *)
                 if t1 < 1 then begin (* compute V1' *)
                    x1 := x0 + t1 * deltax;
                    y1 := y0 + t1 * deltay
                 end (* compute V1' *);
                 if t0 > 0 then begin (* compute V0' *)
                    x0 := x0 + t0 * deltax;
```

```
                y0 := y0 + t0 * deltay
              end (* compute V0' *);
              visible := true
            end; (* compute coordinates *)
      end
    end; (* not trivial reject *)
1:    null;
  end (* clip2d *);
```

3. Three-Dimensional Line Clipping Algorithm

3.1. Explanation

The three-dimensional line clipping algorithm is analogous to the method developed in Section 2. First, a trivial accept test is performed. This involves checking each endpoint of the line segment to be clipped against each bounding plane of the viewing volume. If both endpoints lie on the outside of at least one bounding plane, then the line segment is trivially rejected. Otherwise, the line segment is mapped into a parametric representation. Denoting the endpoints of the line segment as $\mathbf{V}_0$ and $\mathbf{V}_1$ with coordinates (x_0, y_0, z_0) and (x_1, y_1, z_1), respectively, then the parametric representation of the line is given by equations (2.2) and (2.3) with the addition of the following equation:

$$z = z_0 + \Delta z\, t \tag{3.1}$$

where

$$\Delta z = z_1 - z_0 \tag{3.2}$$

Recall the clipping limits for the right-pyramid viewing volume given by inequalities (1.4):

$$\begin{aligned} -z \le x \le z \\ -z \le y \le z \end{aligned} \tag{3.3}$$

Substituting the parametric representation for *x* and *y* given by equations (3.1) and (3.2) into inequalities (3.3) yields the following conditions for the part of the extended line that is inside the *volume* of the clip window:

$$\begin{aligned} -(\Delta x + \Delta z)\, t \le x_0 + z_0 \quad \text{and} \quad (\Delta x - \Delta z)\, t \le z_0 - x_0 \\ -(\Delta y + \Delta z)\, t \le y_0 + z_0 \quad \text{and} \quad (\Delta y - \Delta z)\, t \le z_0 - y_0 \end{aligned} \tag{3.4}$$

The inequalities (3.4) can be rewritten in the form of inequalities (2.7). Specifically,

$$p_i\, t \le q_i \quad \text{for } i = 1,\ 2,\ 3,\ 4 \tag{3.5}$$

where the following notation is used:

$$\begin{aligned} &p_1 = -(\Delta x + \Delta z) && q_1 = x_0 + z_0 && \textit{left} \\ &p_2 = \Delta x - \Delta z && q_2 = z_0 - x_0 && \textit{right} \\ &p_3 = -(\Delta y + \Delta z) && q_3 = y_0 + z_0 && \textit{bottom} \\ &p_4 = \Delta y - \Delta z && q_4 = z_0 - y_0 && \textit{top} \end{aligned} \tag{3.6}$$

The geometrical interpretation of these four inequalities is analogous to that given for the two-dimensional case. Extend each of the four bounding sides of the viewing pyramid to be an (infinite) plane. Then, each of these bounding planes divides the three-space into two regions. The *visible side* is the side on which the viewing pyramid lies. In this way, the viewing pyramid can be defined as that volume that is on the visible side of all the bounding planes. From this, it can be seen that each of the inequalities (3.5) corresponds to one of these planes (left, right, bottom, and top, respectively), and describes its visible side. To be more specific, extend $\mathbf{V}_0\mathbf{V}_1$ to be a line of infinite extent. Then each inequality provides the range of values of the parameter t for which this extended line is on the visible side of the corresponding plane. Furthermore, the particular parametric value for the point of intersection is $t = q_i / p_i$. Also, the sign of q_i indicates on which side of the corresponding bounding plane the point $\mathbf{V}_0$ lies. Specifically, if $q_i \geq 0$, then $\mathbf{V}_0$ is on the visible side of the bounding plane (inclusive), and if $q_i < 0$, then $\mathbf{V}_0$ is on the invisible side.

Now, it is readily verified that three of the p_i coefficients are linearly independent since they satisfy only one linear relation:

$$p_1 + p_2 - p_3 - p_4 = 0$$

This equation can be solved for any one of the coefficients in terms of the other three independent ones. Thus, there is no restriction on the sign of any of the four p_i's.

If p_i is negative, the i'th inequality becomes

$$t \geq q_i / p_i \tag{3.7}$$

Since the range of parametric values for which the extended line is on the visible side of the corresponding plane is at a minimum at the point of intersection, the direction defined by $\overrightarrow{\mathbf{V}_0\mathbf{V}_1}$ is from the invisible side to the visible side of the plane.

Analogously, if p_i is positive, the i'th inequality becomes

$$t \leq q_i / p_i \tag{3.8}$$

and the direction defined by $\overrightarrow{\mathbf{V}_0\mathbf{V}_1}$ is from the visible side to the invisible side of the plane.

Finally, if p_i is zero, the i'th inequality becomes

$$0 \leq q_i \tag{3.9}$$

Since inequality (3.9) is independent of t, it is satisfied for all t if $q_i \geq 0$, and there is no solution if $q_i < 0$. Geometrically, if p_i is zero, there is no point of intersection of the extended line with the corresponding plane; rather, the extended line is parallel to the plane. Furthermore, if $q_i \geq 0$, then in this case the entire extended line is on the visible side of the plane (inclusive); if $q_i < 0$, then it is entirely on the invisible side of the plane. In the former case, there may or may not be a visible segment depending on where $\mathbf{V}_0$ and $\mathbf{V}_1$ lie on the extended line. However, the latter case cannot have a visible segment since the extended line lies outside the viewing volume and hence this is a trivial reject case.

Now, the intersection of the four inequalities gives the range of values of the parameter t for which the extended line is inside the viewing pyramid. However, the line segment to be clipped ($\mathbf{V}_0\mathbf{V}_1$) is only part of the extended line and is described by $0 \leq t \leq 1$. Hence, the solution to the three-dimensional clipping problem is equivalent to the inequalities (3.5) under the condition $0 \leq t \leq 1$. The algorithm to accomplish this is identical to that developed in Section 2 except the definitions of the p_i's and q_i's are given by equation

(3.6).

As was described in Section 1, hither and yon clipping planes can be optionally added to form a frustum of vision. This is accomplished by recalling the clipping limits engendered by the addition of hither and yon clipping planes which were given by inequalities (1.5):

$$h \le z \le f \tag{3.10}$$

Substituting the parametric representation for z given by equations (3.1) and (3.2) into inequalities (3.10) yields the following additional two constraints:

$$-\Delta z \; t \le z_0 - h \tag{3.11}$$

and

$$\Delta z \; t \le f - z_0 \tag{3.12}$$

These inequalities can be rewritten in the form of inequalities (3.5) where

$$\begin{array}{ll} p_5 = -\Delta z & q_5 = z_0 - h \\ p_6 = \Delta z & q_6 = f - z_0 \end{array} \tag{3.13}$$

The geometric meaning of the three-dimensional clipping process is that the line segment is "squeezed" down by considering where the extended line intersects each bounding plane. More specifically, each endpoint of the given line segment $\mathbf{V}_0\mathbf{V}_1$ is used as an initial value for an endpoint of the visible segment $\mathbf{V'}_0\mathbf{V'}_1$. Then the point of intersection of the extended line with each bounding plane is considered.

Since there has been an initial trivial reject test, the intersection point cannot be further along the line ($q_i < p_i < 0$) than $\mathbf{V}_1$, since this can only happen when the line segment terminates before the line enters the viewing volume. Hence, the line segment terminates on the visible side of the bounding plane. For this situation, the point of intersection is compared to $\mathbf{V}_0$. Should $\mathbf{V}_0$ be further along the line than the point ($q_i \ge 0$), then the line segment starts on the visible side of the bounding plane; thus, no further computation would be necessary with respect to this bounding plane to determine $\mathbf{V'}_0$. For this reason, the algorithm checks to see if the point of intersection is further along the line than $\mathbf{V}_0$ ($q_i < 0$), that is, to see if the line segment starts on the invisible side of the bounding plane. If this is the case then $\mathbf{V'}_0$ will be further along the line than $\mathbf{V}_0$, and needs to be determined. In this case, the point of intersection is first compared to $\mathbf{V'}_1$. If the point is further along the line, then $\mathbf{V'}_1$ (and thus $\mathbf{V'}_0\mathbf{V'}_1$) must be on the invisible side of the plane. In this case, the line segment terminates before the line enters the viewing volume, and thus the line segment is rejected. Otherwise, the point of intersection is compared to $\mathbf{V'}_0$, and if the point is further along the line, then $\mathbf{V'}_0$ is moved up to this point.

On the other hand, if the direction is from the visible side to the invisible side of the plane ($p_i > 0$), then this point of intersection is first compared to $\mathbf{V}_0$. $\mathbf{V}_0$ cannot be further along the line than the point of intersection ($q_i < 0$), since such a line segment would earlier have been rejected. Hence, the line segment begins on the visible side of the bounding plane. For this situation, $\mathbf{V}_1$ is compared to the point of intersection. Should this point be further along than $\mathbf{V}_1$ ($q_i \ge p_i > 0$), then the line segment terminates on the visible side of the bounding plane; thus, no further computation would be necessary with respect to this bounding plane to determine $\mathbf{V'}_1$. For this reason, the algorithm examines $\mathbf{V}_1$ to see if it is further along than the point of intersection ($0 \le q_i < p_i$), that is, to check that the line segment terminates on the invisible side of the bounding plane. If this is the case then $\mathbf{V'}_1$ will precede $\mathbf{V}_1$ along the line, and needs to be determined. In this case, the point of intersection is first compared to $\mathbf{V'}_0$. If $\mathbf{V'}_0$ is further along the line than the point, then $\mathbf{V'}_0$ (and thus $\mathbf{V'}_0\mathbf{V'}_1$) must be on the invisible side of the plane. In this case, the line segment does not begin until after the line exits the viewing volume, and thus the line segment is rejected. Otherwise, the point of intersection is compared to $\mathbf{V'}_1$, and if $\mathbf{V'}_1$ is further along the line than the point, then $\mathbf{V'}_1$ is moved down to this point.

Finally, if the extended line is parallel to the bounding plane and on its invisible side, the line segment is rejected. At the end of the algorithm, if the line segment was not rejected, $\mathbf{V'}_0$ and $\mathbf{V'}_1$ are the required the endpoints of the visible segment.

3.2. Algorithm

The improved algorithm to perform three-dimensional line clipping is as follows:

```
(* Clip 3-d line segment with endpoints (x0, y0, z0) and (x1, y1, z1).     *)
(* The viewing volume is a frustum of vision with hither and yon clipping. *)
(* A viewing volume of a right pyramid with its apex at the origin         *)
(* without hither and yon clipping can be handled simply by removing       *)
(* the lines indicated in the algorithm.                                   *)

   procedure clip3d (var x0, y0, z0, x1, y1, z1: real; var visible: Boolean);
   label 1;
   var
         t0, t1: real;
         deltax, deltay, deltaz: real;

   begin (* clip3d *)

         if (((x0 > z0) and (x1 > z1)) or
           ((y0 > z0) and (y1 > z1)) or
           ((x0 < -z0) and (x1 < -z1)) or
           ((y0 < -z0) and (y1 < -z1))
(* *)       or ((z0 < h) and (z1 < h))
(* *)       or ((z0 > f) and (z1 > f))
            ) then
       begin (* trivial reject *)
         visible := false;
         goto 1;
       end; (* trivial reject *)

       begin (* not trivial reject *)
            visible := false;
          t0 := 0;
          t1 := 1;
          deltax := x1 - x0;
          deltaz := z1 - z0;
```

```
      if clipt (- deltax - deltaz, x0 + z0, t0, t1) (* left *) then
        if clipt (deltax - deltaz, z0 - x0, t0, t1) (* right *) then begin
          deltay := y1 - y0;
          if clipt (- deltay - deltaz, y0 + z0, t0, t1) (* bottom *) then
            if clipt (deltay - deltaz, z0 - y0, t0, t1) (* top *) then
(* *)           if clipt (- deltaz, z0 - h, t0, t1) (* hither *) then
(* *)             if clipt (deltaz, f - z0, t0, t1) (* yon *) then
                    begin (* compute coordinates *)
                    if t1 < 1 then begin (* compute V1' *)
                      x1 := x0 + t1 * deltax;
                      y1 := y0 + t1 * deltay;
                      z1 := z0 + t1 * deltaz
                    end (* compute V1' *);
                    if t0 > 0 then begin (* compute V0' *)
                      x0 := x0 + t0 * deltax;
                      y0 := y0 + t0 * deltay;
                      z0 := z0 + t0 * deltaz
                    end (* compute V0' *);
                    visible := true
                  end (* compute coordinates *)
        end
    end; (* not trivial reject *)
1:    null;
  end (* clip3d *);
```

where function clipt is as was defined in Section 2. Hither and yon clipping corresponds to the lines in the algorithm indicated by "(* *)" in the left margin. These lines should be removed if the viewing volume is not bounded by hither and yon clipping planes.

4. Line Clipping Algorithm for Homogeneous Coordinate Systems

4.1. Explanation

Recall that for line clipping in homogeneous coordinates, clipping is performed after the application of the perspective transformation, but prior to the perspective depth division.

The method of Section 3 can be easily extended to handle this case. First, a trivial reject test is performed. If both endpoints lie on the outside of at least one bounding plane, then the line segment is trivially rejected. Otherwise, the line segment is mapped into a parametric representation. Denoting the endpoints of the line segment as $\mathbf{V}_0$ (x_0, y_0, z_0, w_0) and $\mathbf{V}_1(x_1, y_1, z_1, w_1)$, then the parametric representation of the line is given by equations (2.2), (2.3), (3.1), and (3.2) with the addition of the following equation:

$$w = w_0 + \Delta w \; t \tag{4.1}$$

where

$$\Delta w = w_1 - w_0 \tag{4.2}$$

Substituting this parametric representation into inequalities (1.6) yields the following conditions for the part of the extended line that is inside the frustum of vision:

$$\begin{aligned} -(\Delta x+\Delta w)\ t \le x_0 + w_0 \ \text{ and } \ (\Delta x-\Delta w)\ t \le w_0 - x_0 \\ -(\Delta y+\Delta w)\ t \le y_0 + w_0 \ \text{ and } \ (\Delta y-\Delta w)\ t \le w_0 - y_0 \\ -\Delta z\ t \le z_0 \ \text{ and } \ (\Delta z-\Delta w)\ t \le w_0 - z_0 \end{aligned} \tag{4.3}$$

Again, these conditions are inequalities describing the *interior* of the clipping volume rather than equations defining its *boundary*. The six inequalities (4.3) can be rewritten in the form of inequalities (3.4) with i = 1, 2, ..., 6. Specifically,

$$p_i\ t \le q_i \quad \text{for} \quad i=1,\ 2,\ ...,\ 6 \tag{4.4}$$

where

$$\begin{array}{lll} p_1 = -(\Delta x + \Delta w) & q_1 = x_0 + w_0 & \textit{left} \\ p_2 = \Delta x - \Delta w & q_2 = w_0 - x_0 & \textit{right} \\ p_3 = -(\Delta y + \Delta w) & q_3 = y_0 + w_0 & \textit{bottom} \\ p_4 = \Delta y - \Delta w & q_4 = w_0 - y_0 & \textit{top} \\ p_5 = -\Delta z & q_5 = z_0 & \textit{hither} \\ p_6 = \Delta z - \Delta w & q_6 = w_0 - z_0 & \textit{yon} \end{array} \tag{4.5}$$

It can be shown that there are exactly two linear relations satisfied by the p_i coefficients:

$$\begin{aligned} p_1 + p_2 - p_5 - p_6 = 0 \\ -p_3 - p_4 + 2p_5 + 2p_6 = 0 \end{aligned} \tag{4.6}$$

Thus, four of the coefficients are independent and hence there is no restriction on the sign of any of the six p_i ’s.

Analogous to the method explained in Section 3, clipping using homogeneous coordinate systems is accomplished by solving the inequalities (4.4) under the condition $0 \le t \le 1$. The algorithm is similar to that given in Section 3, except now there are six inequalities to be considered, and the p_i’s and q_i’s are defined by equation (4.5).

4.2. Algorithm

The improved algorithm to perform homogeneous line clipping is as follows:

```
(* Clip 4-d line segment with endpoints (x0,y0,z0,w0) and (x1,y1,z1,w1).   *)
(* The viewing volume is a frustum of vision in three-space with hither    *)
(* and yon clipping.                                                       *)

   procedure clip4d (var x0, y0, z0, w0, x1, y1, z1, w1: real;
                 var visible: boolean);
   label 1;
   var
       t0, t1: real;
       deltax, deltay, deltaz, deltaw: real;

   begin (* clip4d *)
```

```
    if (((x0 > z0) and (x1 > z1)) or
      ((y0 > z0) and (y1 > z1)) or
      ((x0 < -z0) and (x1 < -z1)) or
      ((y0 < -z0) and (y1 < -z1)) or
      ((z0 < h) and (z1 < h)) or
      ((z0 > f) and (z1 > f))) then
   begin (* trivial reject *)
    visible := false;
    goto 1;
   end; (* trivial reject *)

   begin (* not trivial reject *)
      visible := false;
     t0 := 0;
     t1 := 1;
     deltax := x1 - x0;
     deltaw := w1 - w0;
     if clipt (- deltax - deltaw, x0 + w0, t0, t1) (* left *) then
       if clipt (deltax - deltaw, w0 - x0, t0, t1) (* right *) then begin
         deltay := y1 - y0;
         if clipt (- deltay - deltaw, y0 + w0, t0, t1) (* bottom *) then
           if clipt (deltay - deltaw, w0 - y0, t0, t1) (* top *) then begin
             deltaz := z1 - z0;
             if clipt (- deltaz, z0, t0, t1) (* hither *) then
               if clipt (deltaz - deltaw, w0 - z0, t0, t1) (* yon *) then
               begin (* compute coordinates *)
               if t1 < 1 then begin (* compute V1' *)
                 x1 := x0 + t1 * deltax;
                 y1 := y0 + t1 * deltay;
                 z1 := z0 + t1 * deltaz;
                 w1 := w0 + t1 * deltaw
               end (* compute V1' *);
               if t0 > 0 then begin (* compute V0' *)
                 x0 := x0 + t0 * deltax;
                 y0 := y0 + t0 * deltay;
                 z0 := z0 + t0 * deltaz;
                 w0 := w0 + t0 * deltaw
               end (* compute V0' *);
                   visible := true;
             end (* compute coordinates *)
           end
       end
   end (* not trivial reject *);
1:    null;
  end (* clip4d *);
```

where function clipt is as was defined in Section 2.

5. Performance Tests

The computational complexity of the algorithm is linear in the number of line segments to be clipped. The exact requirements are dependent upon the particular data. In general, the worst case occurs when a line segment is not trivially rejected nor parallel to any of the boundary lines or planes, and has both of its original endpoints outside the visible region. In this case, two intersection points must be computed. This requires a total of 12 additions/subtractions, 4 multiplications, and 4 divisions for the two-dimensional algorithm; or 19 additions/subtractions, 6 multiplications, and 4 divisions for the three-dimensional algorithm using a viewing pyramid; or 25 additions/subtractions, 8 multiplications, and 6 divisions for the homogeneous algorithm using a finite viewing frustum.

The improved algorithm was compared to the original version of our algorithm as well as to the traditional Sutherland-Cohen line clipping algorithm for two-dimensional, three-dimensional, and homogeneous clipping, respectively. In the two-dimensional case, there was also a comparison with the Nicholl-Lee-Nicholl algorithm using Pascal code supplied by the authors of that algorithm. Pascal code for the two-dimensional Sutherland-Cohen algorithm was copied verbatim from pages 66-67 of.[15] However, it was found that the Pascal code for the traditional Sutherland-Cohen three-dimensional clipping algorithm as it appears on page 345 of[16] sometimes enters an infinite loop. This is due to the fact that this program was based on an assumption that is not always true. Once it is recognized that the source of this problem is simply this erroneous assumption, then it is a straightforward matter to correct the program. This was done in[17] and it is this corrected version that was used in this comparison test. For the Sutherland-Cohen homogeneous line clipping algorithm, Pascal code was copied verbatim from page 360 of[18]

For each dimension, the three algorithms were executed on four different sets of data, corresponding to different window sizes, and each containing a thousand randomly generated line segments. The data used for the homogeneous case was identical to that for the three-dimensional case, except transformed to the homogeneous formulation. Each line segment was clipped a thousand times to reduce the effects of random variation, resulting in a total of four million clips.

Tests were first performed on a Sun 3/160 with a floating point coprocessor in Pascal under Berkeley UNIX. In the two-dimensional case, the Sutherland/Cohen algorithm used an average of 1486 seconds, while the original version of our algorithm required an average of 1213 seconds (an 18% improvement) and the improved version of our algorithm used an average of 1074 seconds (a 28% improvement). The Nicholl-Lee-Nicholl algorithm was slightly faster (1059 seconds), in percentage terms also 28%. For clipping in three-dimensions against a viewing pyramid, the Sutherland/Cohen algorithm used an average of 1486 seconds, while the original version of our algorithm required an average of 1266 seconds (a 15% improvement) and the improved version of our algorithm used

---

[15] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

[16] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

[17] You-Dong Liang and Brian A. Barsky, ‘‘A New Concept and Method for Line Clipping,’’ *ACM Transactions on Graphics,* Vol. 3, No. 1, January, 1984, pp. 1-22.

[18] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

an average of 1046 seconds (a 30% improvement). For homogeneous clipping using a finite viewing frustum, the Sutherland/Cohen algorithm used an average of 3319 seconds, while the original version of our algorithm required an average of 1364 seconds (a 59% improvement) and the improved version of our algorithm used an average of 1078 seconds (a 68% improvement). It is interesting to note that the computation times for the parametric algorithm are fairly constant over the different dimensions because the same "clipt" algorithm is used independent of dimension. These performance test results are summarized in Table 1. The numbers enclosed in parentheses are the percentage improvements over the Sutherland-Cohen algorithm.

The same timing experiments were carried out on a DECStation 5000 (MIPS architecture) and are shown in Table 2. In this case the percentage improvements are much less, because of the overall much greater performance of this machine, and, in the case of 2D line clipping, the Nicholl-Lee-Nicholl algorithm is actually worse than Sutherland-Cohen. This may be because the version of the algorithm supplied by the authors results in three procedure calls on the average, with each call involving between 8 and 12 parameters, which is at least double the number of parameters for the other algorithms. Some simple benchmarking experiments showed that procedure call parameters were relatively more expensive on the MIPS architecture (although in absolute terms many times faster) than on the 68000 based CISC architecture. This may explain the relative change in performance of the algorithms.

		Dimension (Times in Seconds)	
Algorithm	2	3	4
Sutherland-Cohen	1486	1486	3319
Original Liang-Barsky	1213 (18%)	1266 (15%)	1364 (59%)
Improved Liang-Barsky	1074 (28%)	1046 (30%)	1078 (68%)
Nicholl-Lee-Nicholl	1059 (28%)	-	-

Table 1. Comparison of the performance of the algorithms (SUN3).

		Dimension (Times in Seconds)	
Algorithm	2	3	4
Sutherland-Cohen	36	28	89
Original Liang-Barsky	39 (-8%)	31 (-11%)	35 (61%)
Improved Liang-Barsky	33 (8%)	25 (11%)	24 (73%)
Nicholl-Lee-Nicholl	42 (-17%)	-	-

Table 2. Comparison of the performance of the algorithms (DEC/MIPS).

† These figures do reflect the startling difference in performance between the two machines.

6. Conclusion

An improved version of our earlier parametric line clipping algorithm is presented. There are three types of improvements. First, the original algorithm had no provision for an initial trivial reject which is now introduced. Second, the original algorithm employed two different kinds of tests to accomplish trivial rejects; in the improved version, we add a third type of test for trivial reject. Third, tests are added so as to avoid unnecessary computation when the result will not affect the parametric value of the intersection point.

One of the advantages of our algorithms is that they quickly determine if a line segment is to be rejected. There are three different categories of lines that will be trivially rejected by the improved algorithm. The first type are those line segments that either terminate prior to the line entering the visible region or do not begin until after the line exits the visible region. Such cases are discovered in the initial trivial reject test involving only comparisons. The second category comprises those lines that are parallel to and outside one of the window boundaries; this is detected when p_i=0 with q_i<0. The third kind are those lines that do not intersect the window boundary at all; this occurs when the value about to be stored in t_0 exceeds t_1, or when the value about to be stored in t_1 is less than t_0.

Although a line in the second category is quickly rejected by our algorithms, the Sutherland-Cohen algorithm may perform a significant amount of computation before rejecting it. As an example, consider a diagonal line segment that lies outside a two-dimensional window passing through the lower right region. The Sutherland-Cohen algorithm will clip this line segment so that it begins at the top boundary of the window, and will then execute another pass through the loop, clipping the line segment yet again so that it now begins at the right boundary of the window. It should be noted that each of these clip operations requires the computation of an intersection point. On the other hand, our algorithm would simply compute t_1 to be the parametric value at which the line segment intersects the left boundary. Then, considering the right boundary, it would be in the process of computing t_0 when the reject case would show that the value about to be stored in t_0 already exceeds the computed value of t_1. This information is sufficient to conclude that the line segment bypasses the window and the algorithm determines that the line segment should be rejected. This early determination of reject cases results in a computational savings for this algorithm.

Note that the "heart" of our algorithm, codified in the procedure "clipt", is identical independent of dimension. The applicability of this algorithm for different dimensions only requires a different calling sequence for this procedure. This feature is both theoretically interesting and practically convenient. Consequently, the computation times for this algorithm are fairly constant over the different dimensions.

Performance tests were conducted on the traditional Sutherland-Cohen[19] algorithm and on both the original and improved versions of our algorithm. All were executed on randomly generated data. In these tests, the original version of our algorithm showed a 18% to 59% improvement compared to the Sutherland-Cohen algorithm and the improved version of our algorithm had a 28% to 68% improvement over the Sutherland-Cohen algorithm, as was illustrated in Table 1. The percentage improvements are less on the

---

[19] William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill, 1979. Second edition.

DECStation 5000 MIPS machine.

Although the three-dimensional and homogeneous coordinate algorithms were presented for perspective viewing volumes, it should be noted that it is equally appropriate for non-perspective viewing volumes, and more generally, for any convex viewing volume. Since the algorithm computes visibility by taking the intersection of the visible sides of a set infinite lines or planes, this approach is applicable to any convex visible region.

Since the structure of the algorithm is such that the conditions describing the interior of the clipping region are all of similar form, these visibility calculations could be processed in parallel.

Acknowledgements

The authors are grateful to Simon Hui, Loretta Willis, and Francis W. Yun for their programming assistance, to Geoff Voelker for carrying out the simulation experiments, and to Walter N. Brown, Caryn Dombroski and Geoff Voelker for their excellent work in figure preparation. This work was supported in part by a National Science Foundation Presidential Young Investigator Award (number CCR-8451997), and in part by the National Science Foundation under grant number CCR-9015 874.

rm: remove /usr/tmp/grefer4826?

---

[20] James F. Blinn and Martin E. Newell, ‘‘Clipping Using Homogeneous Coordinates,’’ pp. 245-251 in *SIGGRAPH ’78 Conference Proceedings*, ACM, August, 1978.

[21] Mike Cyrus and Jay Beck, ‘‘Generalized Two- and Three-Dimensional Clipping,’’ *Computers and Graphics,* Vol. 3, No. 1, 1978, pp. 23-28.

[22] Eric A. Brewer and Brian A. Barsky, ‘‘Clipping After Projection: An Improved Perspective Pipeline.’’ Submitted for publication.

[23] You-Dong Liang and Brian A. Barsky, ‘‘A New Concept and Method for Line Clipping,’’ *ACM Transactions on Graphics,* Vol. 3, No. 1, January, 1984, pp. 1-22.

[24] You-Dong Liang and Brian A. Barsky, ‘‘An Analysis and Algorithm for Polygon Clipping,’’ *Communications of the ACM,* Vol. 26, No. 11, November, 1983, pp. 868-877.Corrigendum in *Communications of the ACM*, Vol. 27, No. 4, April 1984, p. 383.

[25] You-Dong Liang and Brian A. Barsky, ‘‘Introducing A New Technique for Line Clipping,’’ pp. 548-559 in *Proceedings of the International Conference on Engineering and Computer Graphics*, Beijing, 27 August - 1 September 1984. Also in *Journal of Zhejiang University* Special Issue on Computational Geometry, 1984, pp.1-12.

[26] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley Publishing Company, 1990. Second Edition.

[27] Tina M. Nicholl, D. T. Lee, and Robin A. Nicholl, ‘‘An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis,’’ pp. 253-262 in *SIGGRAPH ’87 Conference Proceedings*, ACM, Anaheim, July 27-31, 1987.

[28] Robert F. Sproull and Ivan E. Sutherland, ‘‘A Clipping Divider,’’ pp. 765-775 in *Proceedings of the Fall Joint Computer Conference, Vol. 33,* AFIPS, Thompson Books, Washington, D.C., 1968.