DEVELOPMENT OF THE BRESENHAM LINE ALGORITHM FOR A FIRST COURSE IN COMPUTER SCIENCE

Alfred L. McKinney
K.K. Agarwal
Department of Computer Science
Louisiana State University in Shreveport
Shreveport, LA 71115

ABSTRACT

The Bresenham line algorithm has found wide applicability for drawing straight lines rapidly on raster-scan display devices in both software and hardware. This paper presents the significance of this algorithm and provides its complete derivation for use in a first course in computer graphics. The line drawing procedure is introduced with a simpler algorithm, namely, the Digital Differential Analyzer (DDA). Pseudocode and a Turbo Pascal 6.0 procedure with sample output for both the DDA and the Bresenham algorithms are included.

INTRODUCTION

In a first course in Computer Graphics, it is appropriate to introduce the students to simple straight line drawing algorithms early in the course. The simplest of these is the Digital Differential Analyzer algorithm [Foley 1990, Hearn 1986]. Unfortunately, this algorithm uses floating point arithmetic which is very inefficient for raster-based computer graphics devices. Efficiency is of extreme importance since complex drawings may involve hundreds or thousands of lines. Bresenham's algorithm relies only on integer arithmetic and removes this deficiency. Therefore, it is used frequently in graphics software and some manufacturers have incorporated its use in graphics hardware. Recent improvements to Bresenham's algorithm which rely on using symmetry can be found in [Glassner 1990]. In examining several commonly used computer graphics texts (those listed as references at the end of this paper), we have not found a complete derivation of the Bresenham algorithm. Stevens [1989] gives a C program for the complete algorithm in which we discovered errors. The pixels selected were different than the ones which should have been selected by both the DDA and the true Bresenham algorithm.

THE DDA PROCEDURE

The DDA is an algorithm for calculating pixel positions along the line using the slope-intercept form for the equation of a straight line, namely, $y = m x + b$, where m is the slope of the line and b is the y intercept. This algorithm selects the pixel which is closest to the actual line. For analog devices, this algorithm forms the basis for determining voltages applied to the line drawing mechanism.

It is appropriate to introduce students to this algorithm because of its simplicity. To clarify the working of this algorithm, let us consider drawing a straight line from (5,3) to (10,6). Figure 1 shows resulting pixel positions that are selected by the algorithm. The end-points (5,3) and (10,6) are selected because the line passes through them exactly. The pixel

(6,4) is selected because it is closer to the line than the pixel (6,3). Similarly, at each step the pixel closest to the line is selected.

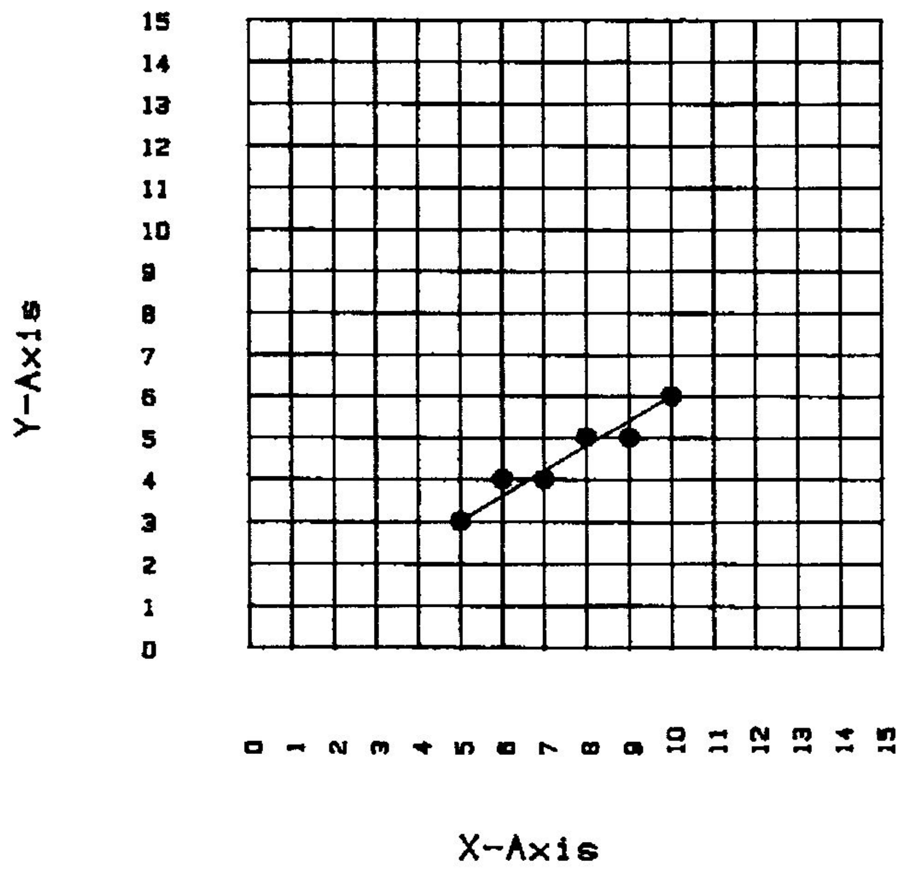

Figure 1 DDA example for a line from (5,3) to (10,6)

A Turbo Pascal 6.0 procedure based on this algorithm is given below:

```
procedure DDA (x1, y1, x2, y2 : integer; c : integer);
  var
    length, i : integer;
    x, y, xincrement, yincrement : real;
  begin
    length := abs(x2-x1);
    if abs(y2-y1) > length then length := abs(y2-y1);
    xincrement := (x2 - x1) / length;
    yincrement := (y2 - y1) / length;
    x := x1 + 0.5;
    y := y1 + 0.5;
    for i := 1 to length do
      begin
        PutPixel(trunc(x), trunc(y), c);
        x := x + xincrement;
        y := y + yincrement;
      end;
    PutPixel(x2, y2, c)
  end;
```

It is clear from the above listing that there are two cases. If the absolute value of the slope of the line is less than 1, the procedure will increment the x value by 1 each time through the loop, while the y value will be incremented by the slope value. If the absolute value of the slope exceeds 1, the y value is incremented by 1 each time through the loop and the x value is incremented by the reciprocal of the slope value.

The selection of the pixel nearest the line in each step is done simply by adding 0.5 to each of the x and y values and truncating them to integral values, which in essence, is rounding them to the closest integers.

THE BRESENHAM LINE ALGORITHM

In 1965 Bresenham [Bresenham 1965] introduced a simple and efficient algorithm for drawing lines that has widely been implemented by both software and hardware [Glassner 1990]. This algorithm uses only integer values and avoids any multiplications. It has a tight and efficient innermost loop that generates the pixels. [Hill 1990] states that it is an important example of an incremental algorithm that computes the location of each pixel on the line based on information about the previous pixel.

DERIVATION OF THE BRESENHAM LINE ALGORITHM

The derivation starts with the slope intercept form of the equation of a line, namely, $y = m\,x + b$, where m is the slope and b is the y-intercept. It is convenient to divide the derivation into four cases:

case 1a. Slope of line is between 0 and 1 ($0 < m < 1$). We assume that the pixel location (x_i, y_i) has been plotted for the line and now we need to decide which is the next pixel to plot. The two choices for the next pixel position are at coordinates (x_i+1, y_i) and (x_i+1, y_i+1). The actual position y on the line can be calculated by

$$y = m\,(x_i+1) + b\ .$$

Then the distances from the actual y to the center of the two choice ordinates y_i and y_i+1 can be calculated by

$$d_1 = y - y_i = m\,(x_i+1) + b - y_i$$

$$d_2 = (y_i+1) - y = y_i + 1 - m\,(x_i+1) - b$$

and the difference of these two distances is

$$d_1 - d_2 = 2\,m\,(x_i+1) - 2\,y_i + 2\,b - 1\ .$$

It is now convenient to define a parameter p_i that provides a measure of the relative distances of two pixels from the actual position on a given line.

$$p_i = dx\,(d_1 - d_2) \qquad (1)$$

Substituting

$$m = dy\,/\,dx\ ,$$

equation (1) can be rewritten so that it involves only integer arithmetic:

$$p_i = dx\,(d_1 - d_2)$$

$$= 2\,dy\,x_i - 2\,dx\,y_i + c \qquad (2)$$

where c has the value $2\,dy + dx\,(2\,b - 1)$ and could be calculated once for all points but (2) can be revised to eliminate this constant. The parameter p_i has a negative value if the pixel at y_i is closer to the line than the upper pixel y_i+1. In that case, we select the lower pixel; otherwise the upper pixel is chosen. Equation (2) is simplified by relating parameters for successive x intervals. Then the value for each succeeding parameter is obtained from the previously calculated parameter. Equation (2) can be rewritten as

$$p_{i+1} = 2\,dy\,x_{i+1} - 2\,dx\,y_{i+1} + c.$$

Subtracting (2) from this expression, we have

$$p_{i+1} - p_i = 2\,dy\,(x_{i+1} - x_i) - 2\,dx\,(y_{i+1} - y_i).$$

But $x_{i+1} = x_i + 1$, so that

$$p_{i+1} = p_i + 2\,dy - 2\,dx\,(y_{i+1} - y_i). \tag{3}$$

This equation provides a way to calculate the value of each successive parameter from the previous one. The first parameter, p_1, is obtained by evaluating (2) with(x_1, y_1) as the starting endpoint and $m = dy / dx$:

$$p_1 = 2\,dy - dx \tag{4}$$

and the successive pixel selection is made as (x_i+1, y_i) if $p_i < 0$, and $p_{i+1} = p_i + 2\,dy$.

Otherwise, the successive pixel location is $(x_i+1, y_i+1)$and

$$p_{i+1} = p_i + 2\,(dy - dx). \tag{5}$$

case 1b. Slope of line is between -1 and 0 ($-1 < m < 0$). We assume that pixel location (x_i, y_i) has been plotted for the line and now we need to decide which is the next pixel to plot. The two choices for the next pixel position are at coordinates (x_i+1, y_i) and (x_i+1, y_i-1).The actual position on the line can be calculated by

$$y = m\,(x_i+1) + b.$$

Then the distance from the actual y to the center of the two choice pixels y_i and y_i-1 can be calculated by:

$$d_1 = y_i - y = y_i - m\,(x_i+1) - b$$

$$d_2 = y - (y_i-1) = m\,(x_i+1) + b - y_i + 1$$

and the difference of these two distances is

$$d_1 - d_2 = 2\,y_i - 2\,m\,x_i - 2\,b - 2\,m - 1. \tag{6}$$

It is now convenient to define a parameter that provides a measure of the relative distances of two pixels from the actual position on a given line. Substituting $m = dy / dx$, equation (6) can be rewritten so that it involves only integer arithmetic:

$$p_i = dx\,(d_1 - d_2) \tag{7}$$

$$= 2\,dx\,y_i - 2\,dy\,x_i + c \tag{8}$$

where c has the value $-2\,dy - dx\,(2\,b + 1)$.

Rewriting (8) at i+1, we obtain

$$p_{i+1} = 2\,dx\,y_{i+1} - 2\,dy\,x_{i+1} + c. \tag{9}$$

Subtracting (8) from (9) we obtain

$$p_{i+1} - p_i = 2\,dx\,(y_{i+1} - y_i) - 2\,dy\,(x_{i+1}-x_i). \tag{10}$$

But $x_{i+1} = x_i + 1$ so that (10) becomes

$$p_{i+1} - p_i = 2\,dx\,(y_{i+1} - y_i) - 2\,dy. \tag{11}$$

Now consider equation (7) with (x_1, y_1) and $m = dy / dx$. We obtain

$$p_1 = -2\,dy - dx. \tag{12}$$

It is seen from (7) that if p_i is < 0 then d_1 is smaller than d_2 and the point corresponding to the distance d_1, namely (x_i+1, y_i) should be selected and from (10) it is seen that

$$p_{i+1} = p_i - 2\,dy. \tag{13}$$

Otherwise, if $p_i >= 0$ then d_2 is smaller so that (x_i+1, y_i-1) is chosen and

$$p_{i+1} = p_i - 2 (dx + dy). \qquad (14)$$

From (4), (5), (13), and (14) it is seen that cases 1a and 1b may be summarized as

```
if $m >= 0$, $s = 1$, otherwise $s = -1$
plot $(x_1, y_1)$
calculate $p_1 = 2 s dy - dx$
at any step i,
    $x_{i+1} = x_i + 1$,
    if $p_i < 0$ then $y_{i+1} = y_i$,
                      $p_{i+1} = p_i + 2 s dy$,
              else $y_{i+1} = y_i + s$,
                      $p_{i+1} = p_i + 2 (s dy - dx)$.
```

case 2a. Slope of line is greater than 1 $(m > 1)$. We need to take unit steps in the y direction, that is, $y_{i+1} = y_i+1$. Otherwise, unit steps of 1 in the x-direction as in cases 1a and 1b might result in steps in the y-direction by more than 1 thus causing holes to appear in the line. We will consider the case where $y_1 < y_2$. Assume (x_1, y_1) has been plotted. Then $y_{i+1} = y_i + 1$. We need to determine at each step which of the pixels (x_i, y_i+1) or $(x_i+1, y_i+1)$is to be plotted next. We need to use an alternate form of the slope-intercept equation of a line, namely,

$$x = y/m - b/m = y\ dx/dy - b\ dx/dy.$$

We define the distances from x_i or x_{i+1} from the true x value on the line by

$$d_1 = x - x_i = (y_i+1)\ dx/dy - b\ dx/dy - x_i$$

$$d_2 = x_i + 1 - x = x_i + 1 - (y_i + 1)\ dx/dy + b\ dx/dy.$$

Subtracting gives

$$d_1 - d_2 = 2\ y_i\ dx/dy - 2\ x_i + 2\ dx/dy - 2\ b\ dx/dy - 1.$$

Now define the parameter

$$p_i = dy\ (d_1 - d_2) \qquad (15)$$

$$p_i = 2\ y_i\ dx - 2\ x_i\ dy + c \qquad (16)$$

where $c = 2\ dx\ (1 - b) - dy$. Then

$$p_{i+1} = 2\ y_{i+1}\ dx - 2\ x_{i+1}\ dy + c$$

and

$$p_{i+1} - p_i = 2\ dx\ (y_{i+1} - y_i) - 2\ dy\ (x_{i+1} - x_i).$$

Since $y_{i+1} = y_i + 1$, this latter equation becomes

$$p_{i+1} - p_i = 2\ dx - 2\ dy\ (x_{i+1} - x_i). \qquad (17)$$

We now substitute (x_1, y_1) into (16) and use the value of c plus $b = y - x\ dy/dx$ to obtain

$$p_1 = 2\ dx - dy.$$

Therefore we can summarize this case by if $p_i < 0$ at any step then $d_1 < d_2$ so we select the pixel (x_i, y_i+1) and from (17) $p_{i+1} = p_i + 2\ dx$, otherwise, select $(x_i+1, y_i+1)$and

$$p_{i+1} = p_i + 2\ (dx - dy).$$

case 2b. Slope of the line is less than -1 ($m < -1$). We need to take unit steps of -1 in the y-direction (see discussion at the beginning of case 2a). Assume that $y_1 > y_2$ and that the pixel (x_1, y_1) has been plotted. Then we need to determine at each step which of the pixels (x_i, y_i-1) or(x_i+1, y_i-1) needs to be selected. We will use the following form of an equation for a line

$$x = y\ dx/dy - b\ dx\ dy.$$

We define the distances from the true x-value on the line from the x_i or x_i-1 by

$$d_1 = x - x_i = (y_i-1)\ dx/dy - b\ dx/dy - x_i$$

$$d_2 = x_i+1 - x = x_i+1 - (y_i-1)\ dx/dy + b\ dx/dy.$$

Subtraction gives

$$d_1 - d_2 = 2\ y_i\ dx/dy - 2\ x_i - 1 - 2\ (b+1)\ dx/dy.$$

Now we define the parameter

$$\begin{aligned} p_i &= dy\ (d_1 - d_2) \\ &= 2\ y_i\ dx - 2\ x_i\ dy + c \end{aligned} \qquad (18)$$

where $c = -dy - 2\ dx\ (b + 1)$. Then

$$p_{i+1} = 2\ y_{i+1}\ dx - 2\ x_{i+1}\ dy + c$$

and

$$p_{i+1} - p_i = 2\ dx\ (y_{i+1} - y_i) - 2\ dy\ (x_{i+1} - x_i). \qquad (19)$$

Since $y_{i+1} = y_i - 1$, this latter equation becomes

$$p_{i+1} - p_i = -2\ dx - 2\ dy\ (x_{i+1} - x_i).$$

We now substitute (x_1, y_1) into (18) and use the value of c plus $b = y - x\ dy/dx$ to obtain

$$p_1 = -dy - 2\ dx.$$

Therefore, we can summarize this case as follows:

if $p_i < 0$ at any step then $d_2 < d_1$ we select the pixel (x_i+1, y_i-1) and from (19) $p_{i+1} = p_i - 2\ dx - 2\ dy$, otherwise, select (x_i, y_i-1) and $p_{i+1} = p_i - 2\ dx$.

We can summarize cases 2a and 2b as

```
if $m >= 0$, $s = 1$, otherwise $s = -1$
plot $(x_1, y_1)$
calculate $p_1 = 2\ s\ dx - dy$
at any step i,
        $y_{i+1} = y_i + s$,
if $dy >= 0$ and if $p_i <= 0$ then
                $p_{i+1} = p_i + 2\ s\ dx$
            else
                $x_{i+1} = x_i + 1$
                $p_{i+1} = p_i + 2\ (s\ dx - dy)$
        else if $p_i <= 0$ then
                $x_{i+1} = x_i + 1$
                $p_{i+1} = p_i + 2\ (s\ dx - dy)$
            else $p_{i+1} = p_i + 2\ s\ dx$.
```

Cases 1a, 1b, 2a, and 2b are summarized in the pseudo-code given in the next section.

THE PSEUDOCODE FOR THE BRESENHAM LINE ALGORITHM

The derivation of the Bresenham Line Algorithm was presented in the previous section. The pseudo-code for this algorithm is as follows:

```
1. Input the endpoints $(x_1, y_1)$ and $(x_2, y_2)$ for the line segment.
2. Arrange the coordinates so that $x_1 < x_2$, that is, $(x_1, y_1)$ is the left endpoint.
3. Calculate $dx = x_2 - x_1$, and $dy = y_2 - y_1$ . Note that $dx > 0$ from step 2.
4. If $dy >= 0$ then set $s = 1$ else set $s = -1$ (since $dx > 0$, s is the sign of the slope).
5. Set $x = x_1$, $y = y_1$.
6. If abs(dy) <= abs(dx)
   then begin
     calculate $p = 2\,s\,dy - dx$,
     while $x <= x_2$ do
       begin
         if $p < 0$
           then
             $p = p + 2\,s\,dy$
           else
             $y = y + s$,
             $p_{i+1} = p_i + 2\,(s\,dy - dx)$
             set_pixel (x, y)
         $x = x + 1$
         end
       end
   else begin
     calculate $p = 2\,s\,dx - dy$
     while $y <> y_2$ do
       begin
         if $dy >= 0$ then
           if $p <= 0$
             then $p = p + 2\,s\,dx$
           else
               $x = x + 1$
             $p = p + 2\,(s\,dx - dy)$
         else if $p <= 0$ then
             $x = x + 1$
             $p = p + 2\,(s\,dx - dy)$
           else $p = p + 2\,s\,dx$
         set_pixel (x, y)
         $y = y + s$
       end
     set_pixel $(x_2, y_2)$
     end.
```

THE BRESENHAM LINE PROCEDURE

The Turbo Pascal 6.0 procedure for the Bresenham line algorithm is given below:

```
procedure BrLine(x1, y1, x2, y2 : integer; c : integer);
  var
    dx, dy, dxabs, dyabs, sdx, sdy, x, y, s, p, c1, c2 : integer;
  begin
    dx := x2 - x1;
    dy := y2 - y1;
    if dy < 0 then s := -1 else s := 1;
    dxabs := abs(dx);
    dyabs := abs(dy);
    if dyabs <= dxabs then
      begin
        p := 2 * s * dy - dx;
        y := y1;
        c1 := 2 * s * dy;
        c2 := 2 * (s * dy - dx);
        for x := x1 to x2 do
          begin
            PutPixel(x, y, c);
            if p < 0 then p := p + c1
              else
                begin
                  y := y + s;
                  p := p + c2
                end;
          end
      end
    else
      begin
        p := 2 * s * dx - dy;
        x := x1;
        c1 := 2 * s * dx;
        c2 := 2 * (s * dx - dy);
        y := y1;
        while y <> y2 do
          begin
            if dy >= 0 then if p <= 0 then p := p + c1
              else begin
                p := p + c2;
                x := x + 1
                end
              else begin
                if p <= 0 then
                  begin
                    p := p + c2;
                    x := x + 1
                  end
                  else p := p + c1
                end;
                y := y + s
          end
      end
  end;
```

The Turbo Pascal 6.0 main program to test the DDA and the Bresenham Line procedures is given below:

```
program test;
uses
  graph;
var
  GraphDriver : integer;
  GraphMode   : integer;
  ErrorCode   : integer;

procedure DDA (x1,y1,x2,y2:integer;c:integer);
        .
        .
        .
  end;
```

```
procedure BrLine(x1,y1,x2,y2:integer;c:integer);
         .
         .
         .
  end;
begin
  GraphDriver := Detect;
  InitGraph(GraphDriver, GraphMode, '\TP\BGI');
  SetGraphMode(VGA);
  DDA(5, 3, 10, 6, 15);
  BrLine(5, 3, 10, 6, 15);
  BrLine(5, 6, 10, 3, 15);
  BrLine(5, 3, 10, 13, 15);
  BrLine(5, 13, 10, 3, 15);
  Readln;
  CloseGraph;
end.
```

Figures 2 through 5 show the output produced by a modified version of this program which produced output on a plotter instead of a graphics monitor to better illustrate why the pixels were selected. They are based on the cases discussed in the derivation section.

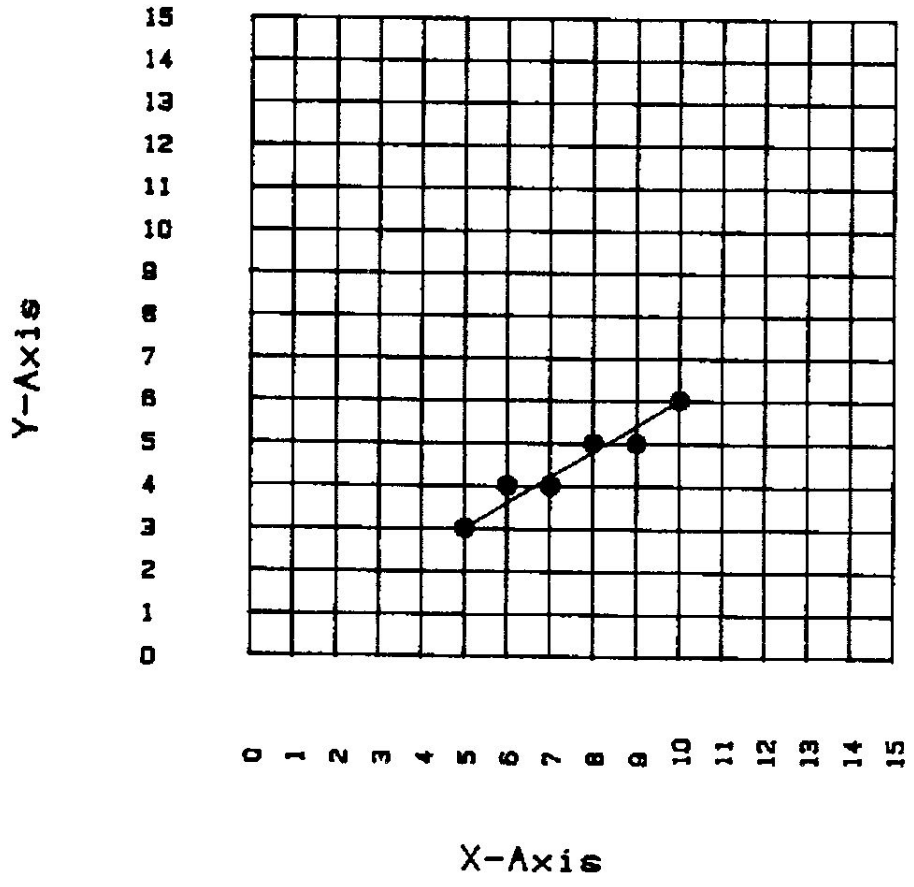

Figure 2 Case 1a of the Bresenham Line Procedure for a line from (5,3) to (10,6)

CONCLUSION

The Bresenham algorithm is representative of a number of curve generation methods and we have found it useful to assign it as a stand-alone programming exercise as well as part of the development of a 2-D graphics package. The discussion and the algorithm for the fundamental line output primitive are incomplete in the references that we have examined.

This paper provides the complete development of this algorithm and, hopefully, will prove to be a useful reference for teachers and students for a first course in computer graphics.

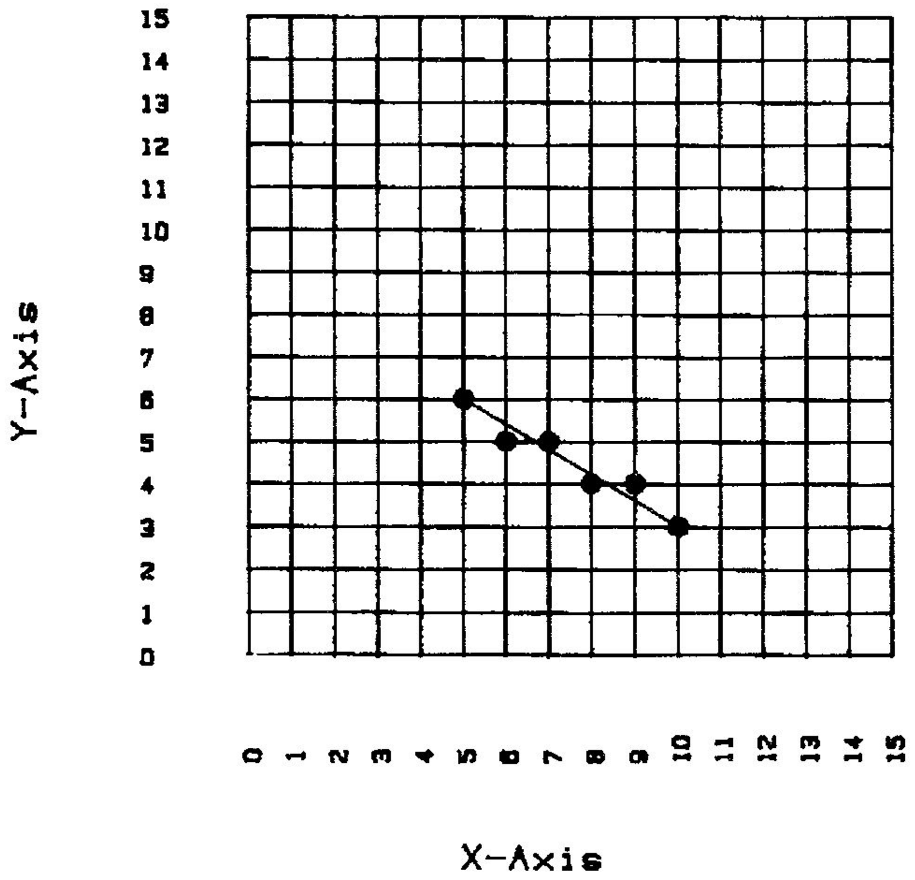

Figure 3 Case 1b of the Bresenham Line Procedure for a line from (5,6) to (10,3)

REFERENCES

[Artwick 1984] Bruce A. Artwick, Applied Concepts in Microcomputer Graphics, Prentice-Hall, 1984.

[Bresenham 1965] J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter", IBM Systems Journal, January 1965, pp.25-30.

[Foley 1990] James D. Foley, Andries van Dam, Steven Feiner and John Hughes, Comput er Graphics: Principles and Practice, 2nd edition, Addison Wesley, 1990.

[Giloi 1978] Wolfgang K. Giloi, Interactive Computer Graphics, Prentice-Hall, 1978.

[Glassner 1990] Andrew Glassner, Graphics Gems, Academic Press, 1990.

[Harrington 1983] Steven Harrington, Computer Graphics: A Programming Approach, McGraw-Hill, 1983.

[Hearn 1986] Donald Hearn and M. Pauline Baker, Computer Graphics, Prentice-Hall, 1986.

[Hill 1990] Francis S. Hill, Jr., Computer Graphics, Macmillan, 1990.

[Mielke 1991] Bruce Mielke, Integrated Computer Graphics, West, 1991.

[Pokorny 1989] Cornel K. Pokorny and Curtis F. Gerald, Computer Graphics: The Principles behind the Art and Science, Franklin, Beedle & Associates, 1989.

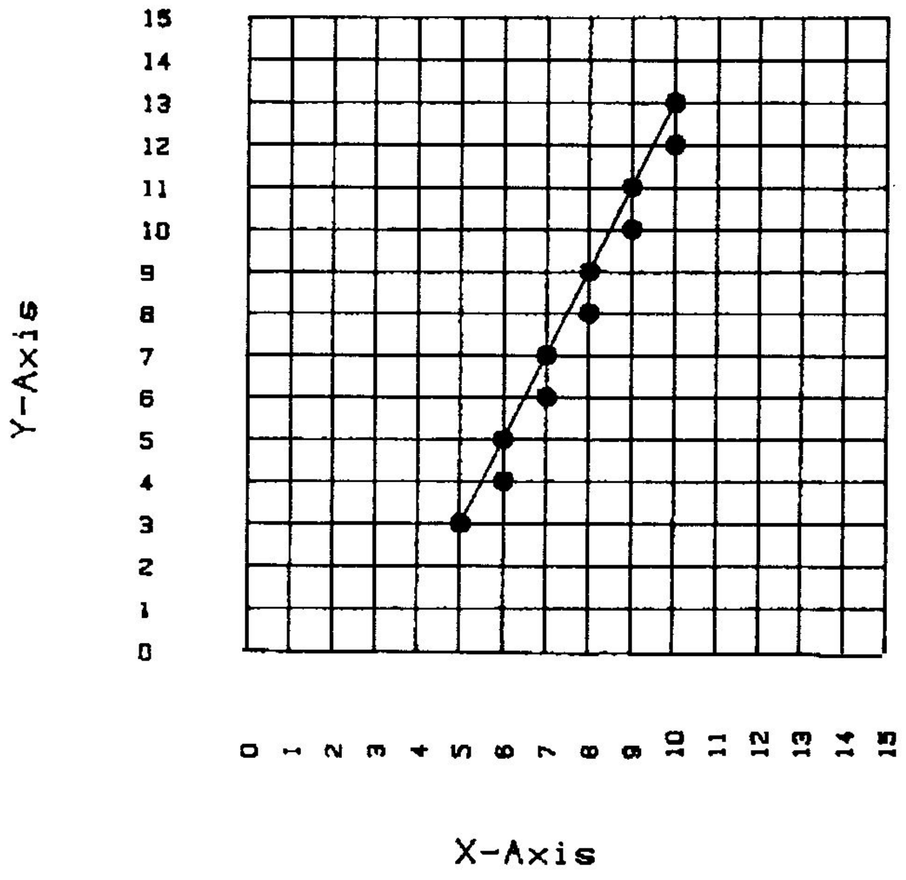

Figure 4 Case 2a of the Bresenham Line Procedure for a line from (5,3) to (10,13)

[Salmon 1987] Rod Salmon and Mel Slater, Computer Graphics: Systems & Concepts, Addison-Wesley, 1987.

[Stevens 1989] Roger T. Stevens, Graphics Programming in C, M&T Books, 1989.

[Watt 1984] Alan Watt, Three Dimensional Computer Graphics, Addison-Wesley, 1989.

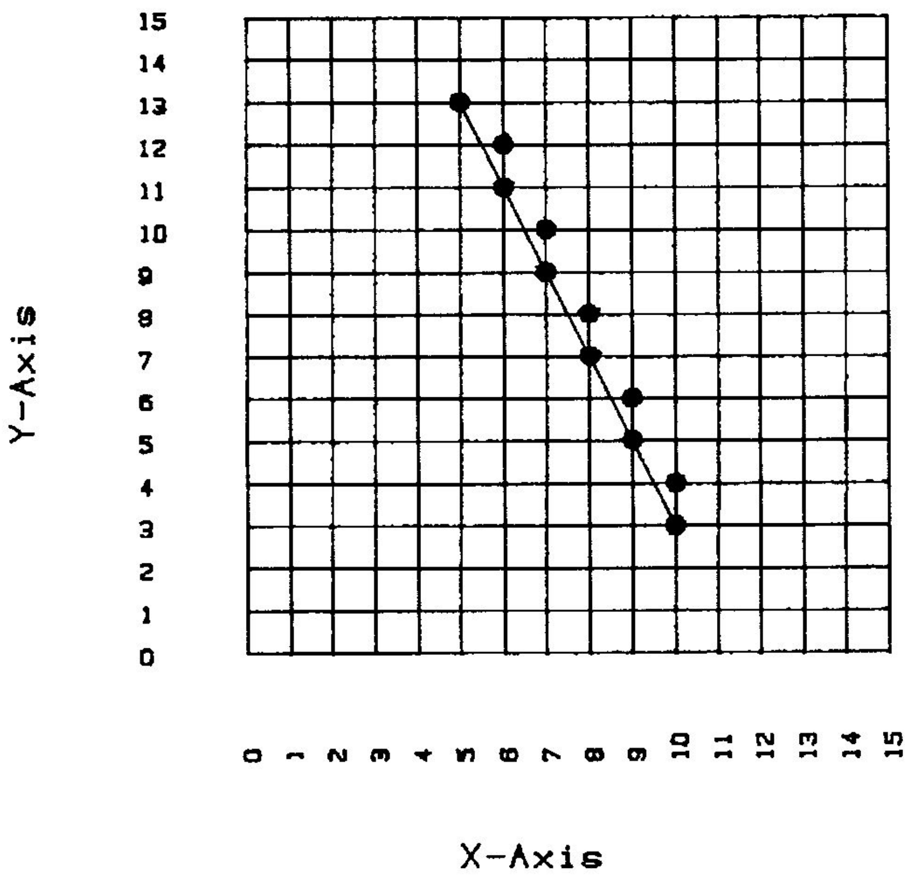

Figure 5 Case 2b of the Bresenham Line Procedure for a line from (5,13) to (10,3)