Bresenham Line-Drawing Algorithm

Phil Koopman, Jr.
North Kingstown, Rhode Island

The task of drawing a straight line on a graphics screen is a fundamental building block for most computer graphics applications. Unfortunately, this capability is not included in many Forth implementations and, for that matter, is not included in the ROM support programs for many personal computers. This article will show you how to draw lines on almost any graphics display, and gives complete listings in MVP-FORTH.

The CRT Display Layout

First, let's establish some conventions. I will assume that the graphics display on your computer is addressed using (X,Y) Cartesian coordinate pairs, where X and Y are both non-negative integers (see Figure One). The point (0,0) — also called the origin — is the upper-left corner of the computer screen. Each addressable point on the screen is called a pixel (short for "picture element"). The X coordinates represent columns of pixels (horizontal distance from the origin), and the Y coordinates represent rows of pixels (vertical distance from the origin).

The exact number of pixels on your computer's display screen is hardware-dependent. However, some representative values are: 320 x 200 pixels (320 horizontal and 200 vertical pixels) for a PC-style, four-color color graphics adapter (CGA) display; 640 x 200 pixels for a PC-style, two-color CGA display; and 640 x 350 pixels for a PC-style sixteen-color enhanced graphics adapter (EGA) display.

The mechanics of setting the graphics display mode desired and plotting a single point on the display are hardware-dependent, and will be left to the user to determine. Screens 3 and 4 of the accompanying listing contain all the machine-specific primitives for PCs and clones with compatible BIOS ROM chips. They are formatted to use the public-domain 8088 assembler cited[1]. These screens will obviously have to be modified for use on other machines.

```
SCREEN #3
\ "PC" COMPATIBLE EGA, CGA, AND TEXT MODES
HEX    \ Machine specific -- change for your machine!!
CODE SET-CGA-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0004 MOV    10 INT     \ 320 x 200 in 3 colors
   NEXT JMP   END-CODE
CODE SET-CGA-HIRES-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0006 MOV    10 INT     \ 640 x 200 in 2 colors
   NEXT JMP   END-CODE
CODE SET-EGA-MODE   ( -> )     \ Set mode and clear screen
   AX , # 0010 MOV    10 INT     \ 640 x 350 in 16 colors
   NEXT JMP   END-CODE

CODE SET-TEXT-MODE   ( -> )          \ 80 column text
   AX , # 0003 MOV   10 INT
   NEXT JMP   END-CODE
DECIMAL
```

```
SCREEN #4
\ "PC" COMPATIBLE POINT PLOT FOR EGA AND CGA
HEX    \ Machine specific -- change for your machine!!
\ Note that fancier direct screen access assembly language
\    programming can *SIGNIFICANTLY* speed up point plotting
\    at the cost of loss of generality.

CODE PLOT-POINT    ( X Y COLOR -> )   \ Plot a single point
   AX POP    DX POP    CX POP     BX , BX XOR   ( page 0 for EGA )
   AH , # 0C MOV    10 INT
   NEXT JMP   END-CODE

DECIMAL
\ XMAX,YMAX delimit screen boundaries
319 CONSTANT XMAX      \ Change to 639 for EGA or CGA/HIRES
199 CONSTANT YMAX      \ Change to 349 for EGA
  4 CONSTANT #COLORS   \ Change to 16 for EGA , 2 for CGA/HIRES
```

```
SCREEN #5
\ VARIABLE DECLARATIONS, MOVE-CURSOR, SPECIAL BRESENHAM POINT
DECIMAL
VARIABLE XNOW    \ (XNOW,YNOW) is current cursor location
VARIABLE YNOW    \              (0,0) is top left corner of CRT
VARIABLE COLOR   \ current line draw color
  1 COLOR !
\ Variables per Foley & Van Dam, Fund. of ICAD, 1st ed. p 435.
VARIABLE INCR1                VARIABLE INCR2
VARIABLE DX                   VARIABLE DY

: MOVE-CURSOR   ( X Y -> )    \ Move cursor location before a draw
   YNOW !  XNOW ! ;
: POINT    ( X Y -> )         \ Point plot using COLOR variable
   COLOR @  PLOT-POINT ;
: B-POINT    ( X Y DELTA -> )   \ For Bresenham line drawing use
   >R  DDUP  POINT  R> ;
```

```
SCREEN #6
\ BRESENHAM LINE DRAW  PRIMITIVES  +X +Y -X -Y
DECIMAL
: +X  ( X1 Y1 DELTA -> X2 Y2 DELTA )
   ROT 1+     ROT ROT ;

: -X  ( X1 Y1 DELTA -> X2 Y2 DELTA )
   ROT 1-     ROT ROT ;

: +Y  ( X1 Y1 DELTA -> X2 Y2 DELTA )
   SWAP 1+    SWAP ;

: -Y  ( X1 Y1 DELTA -> X2 Y2 DELTA )
   SWAP 1-    SWAP ;
```

```
SCREEN #7
\ BRESENHAM LINE FOR   0 < SLOPE < 1
DECIMAL   \ Assume DX and DY are already set up
: LINE0<M<1    ( NEWX NEWY -> )
   DY @  2*  INCR1 !          DY @  DX @ -  2*  INCR2 !
   ( Pick min x )  OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DX @ -     \ Stack:  ( X Y DELTA ---)
   DX @   0 DO     DUP  0<
      IF     ( D < 0 )       +X      B-POINT     INCR1 @ +
      ELSE   ( D >= 0 )      +X +Y   B-POINT     INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

```
SCREEN #8
\ BRESENHAM LINE FOR   1 <= SLOPE < INFINITY
DECIMAL   \ Assume DX and DY are already set up
: LINE1<M<Z     ( NEWX NEWY -> )
   DX @  2*  INCR1 !          DX @  DY @ -  2*  INCR2 !
   ( Pick min y )  DUP   YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DY @ -     \ Stack:  ( X Y DELTA ---)
   DY @   0 DO    DUP  0<
      IF     ( D < 0 )       +Y      B-POINT     INCR1 @ +
      ELSE   ( D >= 0 )      +X +Y   B-POINT     INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

```
SCREEN #9
\ BRESENHAM LINE FOR   -1 < SLOPE < 0
DECIMAL   \ Assume DX and DY are already set up
: LINE-1<M<0     ( NEWX NEWY -> )
   DY @  2*  INCR1 !          DY @  DX @ -  2*  INCR2 !
   ( Pick min x )  OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DX @ -     \ Stack:  ( X Y DELTA ---)
   DX @   0 DO     DUP  0<
      IF     ( D < 0 )       +X      B-POINT     INCR1 @ +
      ELSE   ( D >= 0 )      +X -Y   B-POINT     INCR2 @ +  THEN
   LOOP
   DROP DDROP ;
```

Straightforward Line-Drawing Algorithms

Now that we can assume the availability of a point-plotting word, how can we draw lines? Horizontal and vertical lines are relatively straightforward. For example:

```
: HORIZONTAL-TEST  ( -- )
100 0 DO   I 10 POINT LOOP ;
```

shows that horizontal lines are drawn by merely incrementing an X value for a constant Y value. Similarly, forty-five-degree lines may be drawn by using a word that simultaneously increments both X and Y values, such as:

```
: DIAGONAL-TEST  ( -- )
  100 0 DO
  I I POINT   LOOP ;
```

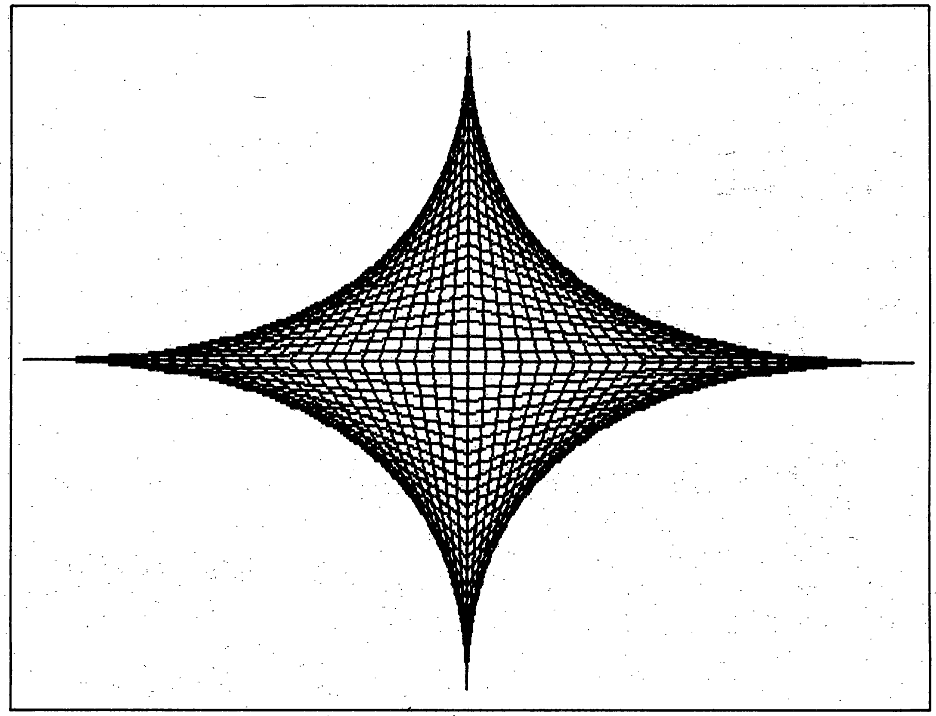
But what about lines that are in-between? A line which spans twice as many X points as Y points would be drawn by:

```
: X=2*Y   ( -- )
  0    100 0 DO
    DUP   I POINT 1 +
    DUP   I POINT 1 +    LOOP
  DROP ;
```

For a generalized line-drawing word with a slope between zero and one (meaning that the X distance of the line is greater than the Y distance, and that both distances are drawn from smaller to larger numbers), we would have:

```
: GENERAL-LINE  ( X1 Y1 X2 Y2 -- )
  SWAP 4 PICK - SWAP
  3 PICK -    >R >R   100 *    R> R>
  100 3 PICK */    SWAP 1 +   0
  DO    3 PICK   3 PICK   100 /   POINT
    SWAP OVER +
    ROT 1 +   SWAP ROT   LOOP
  DROP    100 /   POINT ;
```

The above word takes two (X,Y) coordinate pairs as an input, and scales all Y values by 100 to allow for non-integer increments of Y. While this line-drawing algorithm is conceptually straightfor-



Sample GODSEYE output.

ward, it does require a lot of arithmetic. Even if clever scaling factors were chosen to replace most multiplies and divides with shifts and byte-moves, the initial division of the difference between X1 and X2 (sometimes called "delta X" or just plain "DX") by the difference between Y1 and Y2 ("DY") is unavoidable. Another problem is that sixteen-bit scaled integers are not big enough for use on high-resolution screens. In this example, lines that span more than 100 pixels horizontally are improperly drawn.

The Bresenham Algorithm

The Bresenham line-drawing algorithm[2] requires only sixteen-bit integers with addition, subtraction and multiplication by two (shift left) to draw lines. Instead of a scaled, non-integer Y value, the algorithm shown on screen 7 uses the error accumulation term **DELTA** and integer X and Y values. For lines with a slope between zero and one, the algorithm increments the X value for each point, and increments the Y value only if the **DELTA** value is negative. If **DELTA** is negative, a positive value of DY is added to form the new **DELTA** value. If **DELTA** is positive, a negative value based on both DX and DY is used to form a new **DELTA** value.

Of course, slight variations of this algorithm are needed to account for lines with slopes that are not between zero and one. Screens 5 through 13 contain a complete Bresenham line-drawing vocabulary for all line slopes. Horizontal and vertical lines are treated as special cases for greater speed and simplicity.

The vocabulary for using this drawing package is:

SET-CGA-MODE (--)
Places the display in graphics mode. This word may be redefined or renamed as appropriate for your computer.

SET-TEXT-MODE (--)
Returns the display to an eighty-column text mode. This word may be redefined or renamed as appropriate for your computer.

PLOT-POINT (X Y color --)
Plots a single point on the graphics screen. This word may be redefined as appropriate for your computer.

```
SCREEN #10
\ BRESENHAM LINE FOR   -INFINITY < SLOPE < -1
DECIMAL    \ Assume DX and DY are already set up
: LINE-Z<M<-1    ( NEWX NEWY -> )
   DX @  2*  INCR1 !        DX @  DY @ -  2*  INCR2 !
   ( Pick min y )  DUP   YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT
   ( Compute D )  INCR1 @  DY @ -     \ Stack:  ( X Y DELTA ---)
   DY @  0 DO    DUP  0<
      IF    ( D < 0 )       +Y      B-POINT     INCR1 @ +
      ELSE  ( D >= 0 )      -X +Y   B-POINT     INCR2 @ +  THEN
   LOOP
   DROP DDROP ;

SCREEN #11
\ LINE FOR   SLOPE = INFINITY  ( Vertical )
DECIMAL    \ Assume DX and DY are already set up
: LINEZ    ( NEWX NEWY -> )
   ( Pick min y )   DUP  YNOW @  >
      IF  ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT    0  ( dummy DELTA value )
   DY @  0 DO    +Y   B-POINT   LOOP
   DROP DDROP ;

SCREEN #12
\ LINE FOR   SLOPE = 0    ( Horizontal )
DECIMAL    \ Assume DX and DY are already set up
: LINE0    ( NEWX NEWY -> )
   ( Pick min x )  OVER  XNOW @  >
      IF  ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
   DDUP POINT    0  ( dummy DELTA value )
   DX @  0 DO    +X   B-POINT   LOOP
   DROP DDROP ;

SCREEN #13
\ BRESENHAM PROLOGUE & CALLING ROUTINE
DECIMAL
: LINE   ( XNEW YNEW -> )
   DDUP    ( Extra copy used for final MOVE-CURSOR )
   OVER  XNOW @ -  DUP ABS  DX !   OVER  YNOW @ -  DUP ABS  DY !
   XOR  0<  ( Determine if signs are different )
   DY @  IF      DX @  IF    ( Not horizontal or vertical )
      IF      ( Negative slope )
         DX @  DY @  >  IF  LINE-1<M<0  ELSE  LINE-Z<M<-1 THEN
      ELSE    ( Positive slope )
         DX @  DY @  >  IF  LINE0<M<1  ELSE  LINE1<M<Z  THEN
      THEN
      ELSE  ( Vertical ) DROP  LINEZ  THEN
   ELSE  ( Horizontal )  DROP  LINE0  THEN
   MOVE-CURSOR ;
```

POINT (X Y --)

Same as **POINT**, but without a color value for consistency with **LINE**.

MOVE-CURSOR (X Y --)

Move the current drawing cursor location to the point (X,Y). This word is not called **MOVE** because of possible naming conflicts in some Forth dialects.

LINE (X Y --)

Draw a line from the last cursor position (set by either a **MOVE-CURSOR** or a **LINE** word) to the point (X,Y). The color of the line is determined by the value of the variable **COLOR**.

The demonstration program **GODSEYE** not only draws a pretty picture, but is a good test for the line-drawing algorithm, since it uses lines from each of the different slope-range cases of the line-drawing program.

Conclusion

The Bresenham line-drawing algorithm is an efficient way to draw straight lines. The lines can be drawn even faster than with the example programs by using techniques such as direct screen-memory access instead of BIOS ROM function calls, and by writing optimized assembly language programs that keep variables in registers instead of in memory. For more information on computer graphics (including mathematical derivations of the Bresenham algorithm), please see the recommended reading list.

In the next issue of *Forth Dimensions*, I will show you how to use these line-drawing words to draw fractal-based landscapes.

Recommended Reading

Fundamentals of Interactive Computer Graphics, J.D. Foley and A. Van Dam, Addison-Wesley, Reading MA, 1982.

Principles of Interactive Computer Graphics, W.M. Newman and R.F. Sproull, McGraw-Hill, New York, 1979.

```
SCREEN #14
 \ BRESENHAM  LINE DRAWING TEST PICTURE  -- GODSEYE
 DECIMAL
 : GODSEYE
   SET-CGA-MODE        \ Change to SET-EGA-MODE for the EGA, etc.
   4  0 DO   3 I -  COLOR !   ( Use this line for CGA )
 \ 1  0 DO        1  COLOR !   ( Use this line for CGA/HIRES )
 \ 16 0 DO  15 I -  COLOR !   ( Use this line for EGA )
     76 0 DO       75 I -
        150  OVER  2* -    100  MOVE-CURSOR
        150  OVER  25 +   LINE
        150  OVER  2* +  100  LINE
        150  I  100 +  LINE
        150  OVER  2* -    100  LINE
   DROP   3 +LOOP
   ?TERMINAL  ABORT" BREAK IN GODSEYE"
   LOOP    SET-TEXT-MODE  ;
```

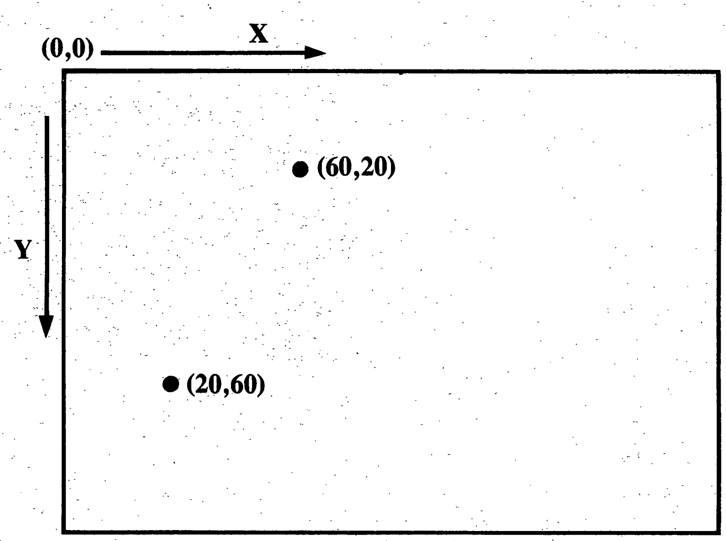


Figure One. Pixel layout on a graphics screen with example points.

References

1. *MVP-FORTH Integer and Floating-Point Math*, P. Koopman, Mountain View Press, 1985.
2. "Algorithm for Computer Control of a Digital Plotter," J.E. Bresenham, *IBM Systems Journal*, Vol. 4, No. 1, pp. 25-30, 1965.

Volume 8, Number 6

March/April 1987
$5.00

FORTH Dimensions

Bresenham Line-Drawing

7776 Limericks

DOS File Disk I/O

Inverse Video and TI-FORTH

State of the Standard

Checksum More