

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 cited¹. These screens will obviously have to be modified for use on other machines.

```
SCREEN #3
0 \ "PC" COMPATIBLE EGA, CGA, AND TEXT MODES
1 HEX \ Machine specific -- change for your machine!!
2 CODE SET-CGA-MODE ( -> ) \ Set mode and clear screen
3   AX , # 0004 MOV 10 INT \ 320 x 200 in 3 colors
4   NEXT JMP END-CODE
5 CODE SET-CGA-HIRES-MODE ( -> ) \ Set mode and clear screen
6   AX , # 0006 MOV 10 INT \ 640 x 200 in 2 colors
7   NEXT JMP END-CODE
8 CODE SET-EGA-MODE ( -> ) \ Set mode and clear screen
9   AX , # 0010 MOV 10 INT \ 640 x 350 in 16 colors
10  NEXT JMP END-CODE
11
12 CODE SET-TEXT-MODE ( -> ) \ 80 column text
13   AX , # 0003 MOV 10 INT
14   NEXT JMP END-CODE
15 DECIMAL
```

```
SCREEN #4
0 \ "PC" COMPATIBLE POINT PLOT FOR EGA AND CGA
1 HEX \ Machine specific -- change for your machine!!
2 \ Note that fancier direct screen access assembly language
3 \ programming can *SIGNIFICANTLY* speed up point plotting
4 \ at the cost of loss of generality.
5
6 CODE PLOT-POINT ( X Y COLOR -> ) \ Plot a single point
7   AX POP DX POP CX POP BX , BX XOR ( page 0 for EGA )
8   AH , # 0C MOV 10 INT
9   NEXT JMP END-CODE
10
11 DECIMAL
12 \ XMAX,YMAX delimit screen boundaries
13 319 CONSTANT XMAX \ Change to 639 for EGA or CGA/HIRES
14 199 CONSTANT YMAX \ Change to 349 for EGA
15 4 CONSTANT #COLORS \ Change to 16 for EGA , 2 for CGA/HIRES
```

```
SCREEN #5
0 \ VARIABLE DECLARATIONS, MOVE-CURSOR, SPECIAL BRESENHAM POINT
1 DECIMAL
2 VARIABLE XNOW \ (XNOW,YNOW) is current cursor location
3 VARIABLE YNOW \ (0,0) is top left corner of CRT
4 VARIABLE COLOR \ current line draw color
5 1 COLOR !
6 \ Variables per Foley & Van Dam, Fund. of ICAD, 1st ed. p 435.
7 VARIABLE INCR1 VARIABLE INCR2
8 VARIABLE DX VARIABLE DY
9
10: MOVE-CURSOR ( X Y -> ) \ Move cursor location before a draw
11 YNOW ! XNOW ! ;
12: POINT ( X Y -> ) \ Point plot using COLOR variable
13 COLOR @ PLOT-POINT ;
14: B-POINT ( X Y DELTA -> ) \ For Bresenham line drawing use
15 >R DDUP POINT R> ;
```

```

SCREEN #6
0 \ BRESENHAM LINE DRAW PRIMITIVES +X +Y -X -Y
1 DECIMAL
2 : +X ( X1 Y1 DELTA -> X2 Y2 DELTA )
3   ROT 1+   ROT ROT ;
4
5 : -X ( X1 Y1 DELTA -> X2 Y2 DELTA )
6   ROT 1-   ROT ROT ;
7
8 : +Y ( X1 Y1 DELTA -> X2 Y2 DELTA )
9   SWAP 1+   SWAP ;
10
11 : -Y ( X1 Y1 DELTA -> X2 Y2 DELTA )
12  SWAP 1-   SWAP ;
13
14
15

SCREEN #7
0 \ BRESENHAM LINE FOR 0 < SLOPE < 1
1 DECIMAL \ Assume DX and DY are already set up
2 : LINE0<M<1 ( NEWX NEWY -> )
3   DY @ 2* INCR1 !   DY @ DX @ - 2* INCR2 !
4   ( Pick min x ) OVER XNOW @ >
5   IF ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
6   DDUP POINT
7   ( Compute D ) INCR1 @ DX @ - \ Stack: ( X Y DELTA ---)
8   DX @ 0 DO DUP 0<
9     IF ( D < 0 ) +X B-POINT INCR1 @ +
10    ELSE ( D >= 0 ) +X +Y B-POINT INCR2 @ + THEN
11  LOOP
12  DROP DDROP ;
13
14
15

SCREEN #8
0 \ BRESENHAM LINE FOR 1 <= SLOPE < INFINITY
1 DECIMAL \ Assume DX and DY are already set up
2 : LINE1<M<Z ( NEWX NEWY -> )
3   DX @ 2* INCR1 !   DX @ DY @ - 2* INCR2 !
4   ( Pick min y ) DUP YNOW @ >
5   IF ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
6   DDUP POINT
7   ( Compute D ) INCR1 @ DY @ - \ Stack: ( X Y DELTA ---)
8   DY @ 0 DO DUP 0<
9     IF ( D < 0 ) +Y B-POINT INCR1 @ +
10    ELSE ( D >= 0 ) +X +Y B-POINT INCR2 @ + THEN
11  LOOP
12  DROP DDROP ;
13
14
15

SCREEN #9
0 \ BRESENHAM LINE FOR -1 < SLOPE < 0
1 DECIMAL \ Assume DX and DY are already set up
2 : LINE-1<M<0 ( NEWX NEWY -> )
3   DY @ 2* INCR1 !   DY @ DX @ - 2* INCR2 !
4   ( Pick min x ) OVER XNOW @ >
5   IF ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
6   DDUP POINT
7   ( Compute D ) INCR1 @ DX @ - \ Stack: ( X Y DELTA ---)
8   DX @ 0 DO DUP 0<
9     IF ( D < 0 ) +X B-POINT INCR1 @ +
10    ELSE ( D >= 0 ) +X -Y B-POINT INCR2 @ + THEN
11  LOOP
12  DROP DDROP ;
13
14
15

```

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 1 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
1 1 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 1 POINT 1+
DUP 1 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:

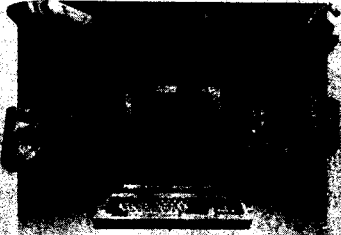
```

: 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 straightforward,

ASK FORTH ENGINEERING ABOUT REAL TIME THAT'S ON TIME.



Find Out How To Implement
Real-Time Systems in:

- Digital Signal Processing
- Manufacturing Process Control
- Machine Vision
- Robotics

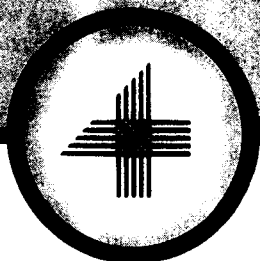
...on time and under budget.

For The Answers To Your
Questions Call
FORTH, Inc. at the
Number Below

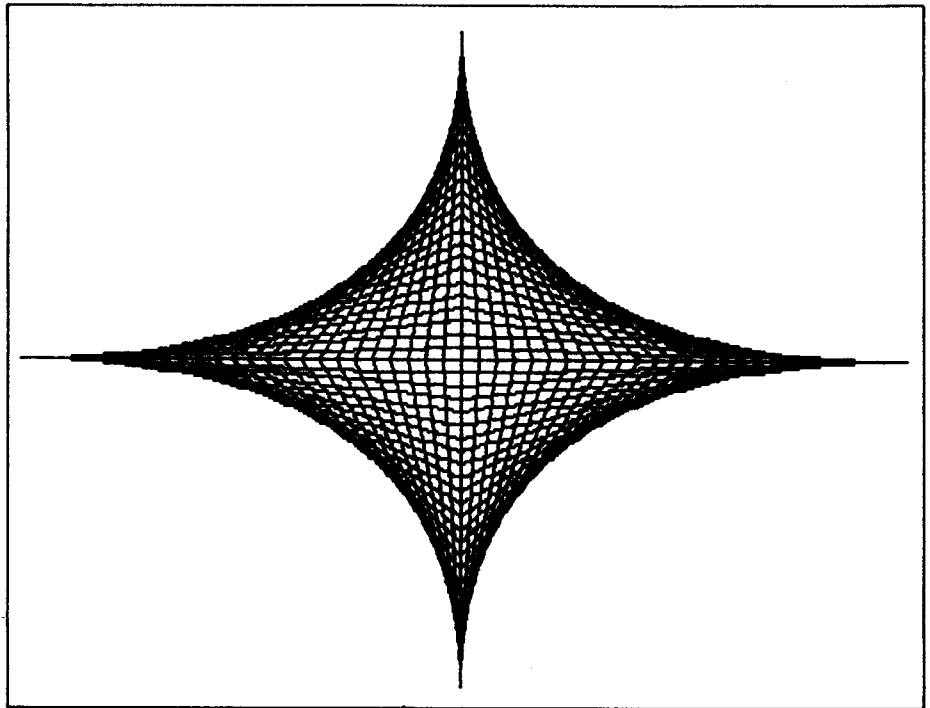
(415) 322-4900
EXT. 200

FORTH, Inc., 111 N. Sepulveda
Bldg., Manhattan Beach, CA
90266.

ON TIME. UNDER BUDGET.



FORTH, Inc.



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 algorithm² 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
0 \ BRESENHAM LINE FOR  -INFINITY < SLOPE < -1
1 DECIMAL  \ Assume DX and DY are already set up
2 : LINE-Z<M<-1 ( NEWX NEWY -> )
3   DX @ 2* INCR1 !      DX @ DY @ - 2* INCR2 !
4   ( Pick min y ) DUP  YNOW @ >
5   IF ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
6   DDUP POINT
7   ( Compute D ) INCR1 @ DY @ -      \ Stack: ( X Y DELTA ----)
8   DY @ 0 DO  DUP 0<
9     IF ( D < 0 )      +Y      B-POINT  INCR1 @ +
10    ELSE ( D >= 0 )   -X +Y   B-POINT  INCR2 @ + THEN
11  LOOP
12  DROP DDROP ;
13
14
15

```

```

SCREEN #11
0 \ LINE FOR  SLOPE = INFINITY ( Vertical )
1 DECIMAL  \ Assume DX and DY are already set up
2 : LINEZ ( NEWX NEWY -> )
3   ( Pick min y )  DUP  YNOW @ >
4   IF ( current cursor at min y ) DDROP XNOW @ YNOW @ THEN
5   DDUP POINT 0 ( dummy DELTA value )
6   DY @ 0 DO  +Y  B-POINT  LOOP
7   DROP DDROP ;
8
9
10
11
12
13
14
15

```

```

SCREEN #12
0 \ LINE FOR  SLOPE = 0 ( Horizontal )
1 DECIMAL  \ Assume DX and DY are already set up
2 : LINE0 ( NEWX NEWY -> )
3   ( Pick min x ) OVER XNOW @ >
4   IF ( current cursor at min x ) DDROP XNOW @ YNOW @ THEN
5   DDUP POINT 0 ( dummy DELTA value )
6   DX @ 0 DO  +X  B-POINT  LOOP
7   DROP DDROP ;
8
9
10
11
12
13
14
15

```

```

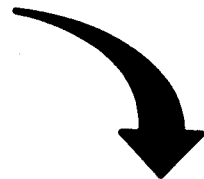
SCREEN #13
0 \ BRESENHAM PROLOGUE & CALLING ROUTINE
1 DECIMAL
2 : LINE ( XNEW YNEW -> )
3   DDUP ( Extra copy used for final MOVE-CURSOR )
4   OVER XNOW @ - DUP ABS DX ! OVER YNOW @ - DUP ABS DY !
5   XOR 0< ( Determine if signs are different )
6   DY @ IF DX @ IF ( Not horizontal or vertical )
7     IF ( Negative slope )
8       DX @ DY @ > IF LINE-1<M<0 ELSE LINE-Z<M<-1 THEN
9     ELSE ( Positive slope )
10      DX @ DY @ > IF LINE0<M<1 ELSE LINE1<M<Z THEN
11    THEN
12    ELSE ( Vertical ) DROP LINEZ THEN
13    ELSE ( Horizontal ) DROP LINE0 THEN
14    MOVE-CURSOR ;
15

```

BRYTE FORTH

for the

INTEL 8031 MICRO- CONTROLLER



FEATURES

- FORTH-79 Standard Sub-Set
- Access to 8031 features
- Supports FORTH and machine code interrupt handlers
- System timekeeping maintains time and date with leap year correction
- Supports ROM-based self-starting applications

COST

130 page manual —\$ 30.00
8K EPROM with manual—\$100.00

Postage paid in North America.
Inquire for license or quantity pricing.

Bryte Computers, Inc.
P.O. Box 46, Augusta, ME 04330
(207) 547-3218

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.

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.

```
SCREEN #14
0 \ BRESENHAM LINE DRAWING TEST PICTURE -- GODSEYE
1 DECIMAL
2 : GODSEYE
3   SET-CGA-MODE           \ Change to SET-EGA-MODE for the EGA, etc.
4   4 0 DO 3 I - COLOR !  ( Use this line for CGA )
5 \ 1 0 DO 1 COLOR !      ( Use this line for CGA/HIRES )
6 \ 16 0 DO 15 I - COLOR ! ( Use this line for EGA )
7   76 0 DO 75 I -
8     150 OVER 2* - 100 MOVE-CURSOR
9     150 OVER 25 + LINE
10    150 OVER 2* + 100 LINE
11    150 I 100 + LINE
12    150 OVER 2* - 100 LINE
13  DROP 3 +LOOP
14  ?TERMINAL ABORT" BREAK IN GODSEYE"
15  LOOP SET-TEXT-MODE ;
```

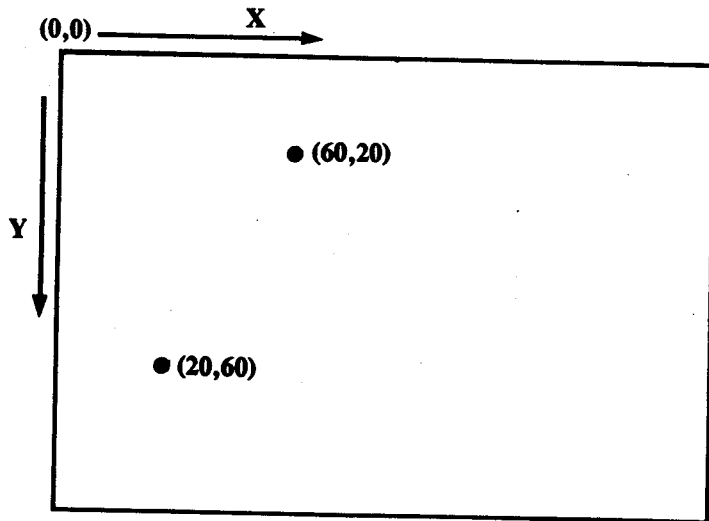


Figure One. Pixel layout on a graphics screen with example points.