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 1 and 2 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.

These screens will obviously have to be modified for use on other machines.
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 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 10 POINT LOOP;

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

: 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 straightforward,
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 \(\text{DELTA}\) 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 \(\text{DELTA}\) value is negative. If \(\text{DELTA}\) is negative, a positive value of DY is added to form the new \(\text{DELTA}\) value. If \(\text{DELTA}\) is positive, a negative value based on both DX and DY is used to form a new \(\text{DELTA}\) 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:

\[
\begin{align*}
\text{SET- CGA-MODE} & \quad ( -- ) \\
\text{SET- TEXT-MODE} & \quad ( -- ) \\
\text{PLOT- POINT} & \quad ( \text{X Y color} \ -- ) \\
\end{align*}
\]

Places the display in graphics mode. This word may be redefined or renamed as appropriate for your computer.

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

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 ;

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 ;

SCREEN #12
0 \ LINE FOR SLOPE = 0 ( Horizontal )
1 DECIMAL \ Assume DX and DY are already set up
2 : LINEO ( 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 ;

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 @ DX @ DU P ABS DX @ OVER YNOW @ DX @ DU P ABS DX @
5 XUR 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<K<0 ELSE LINE-2<K<-1 THEN
9 ELSE ( Positive slope )
10 DX @ DY @ IF LINE0<K<1 ELSE LINE1<K<2 THEN
11 THEN)
12 ELSE ( Vertical ) DROP LINEZ THEN
13 ELSE ( Horizontal ) DROP LINEO THEN
14 MOVE-CURSOR ;

BRYTE FORTH
for the
INTEL 8031
MICROCONTROLLER

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

Volume VIII, No. 6 15
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


References

Bresenham
Line-Drawing

7776 Limericks
DOS File Disk I/O
Inverse Video and TI-FORTH
State of the Standard
Checksum More