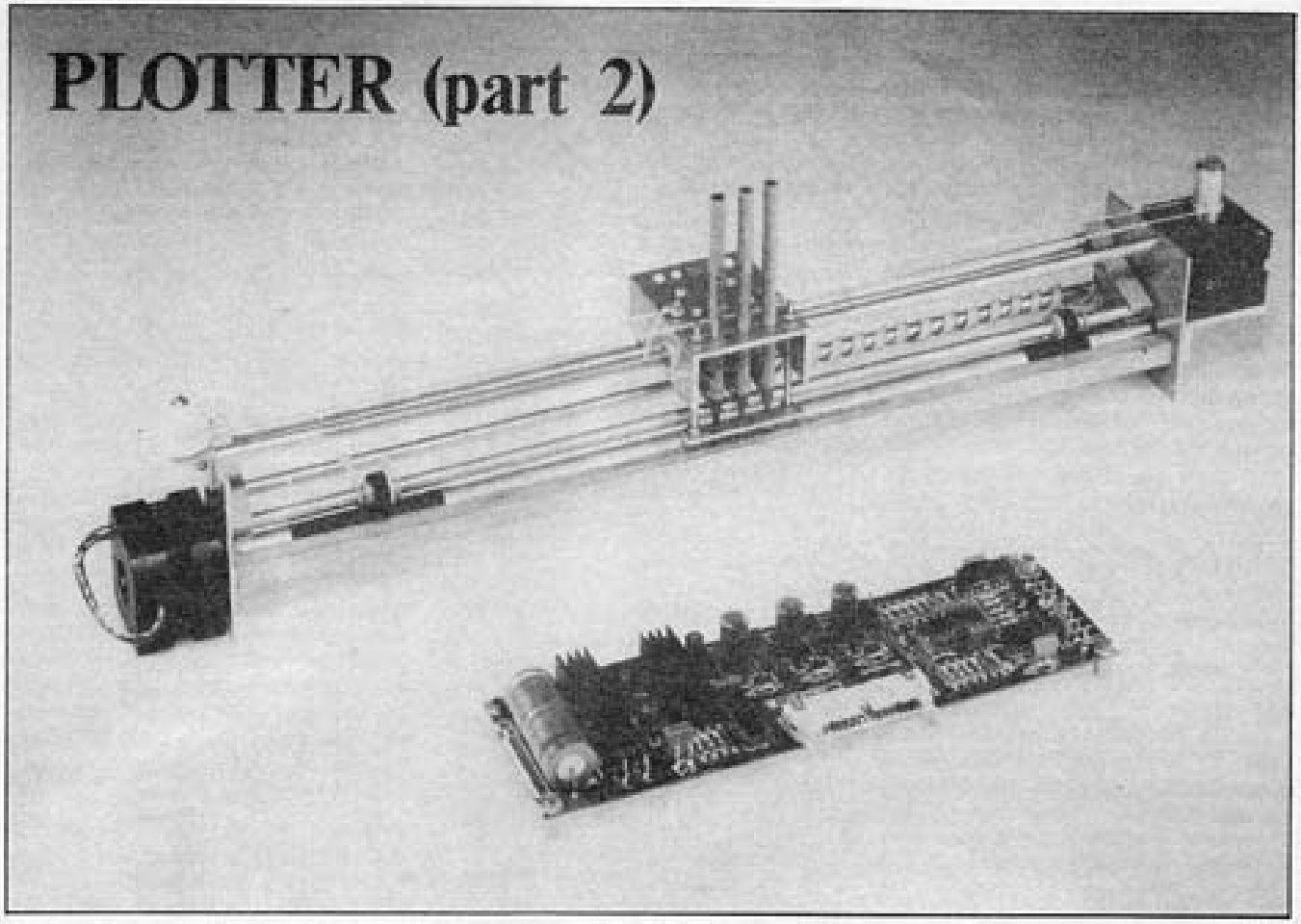
PLOTTER (part 2)



No hardware without software, and vice versa. In this month's final instalment we lend a hand to all constructors of the plotter who are eager to write control software, but need general algorithms, flow diagrams and elementary programming procedures as guidance for tailoring the communication patch between their computer, available graphics programs, and the plotter.

Before attempting to write a plotter interface program, it is necessary to acquire a basic understanding of computer control (*software/hardware*) in combination with graphics (or, more specifically, *drawing*). An algorithm needs to be devised for translating graphics information (on screen or in any form of memory) to actual pen positioning commands. The low-cost plotter described last month has no "on-board" intelligence, and must, therefore, be controlled at the bit level by the computer. In order to obtain reasonable drawing speed, it is necessary to write part of the control program in machine language, which accepts commands or command strings from a line editor, and translates these into pen movement commands by actuating the relevant control lines on the plotter interface board.

The bit assignment in the plotter control word is shown in Fig. 9. A stepper motor performs a full or half step, depending on the logic level of bit 2, on each positive transition of the clock signal (bit 0). The clock pulse must remain logic high for at least 10 μs. Straight lines can be drawn by actuating the X or Y motor alone. Lines under an angle of 45° are drawn when the motors are actuated simultaneously. When both motors are actuated, but one is operated in the full step mode, and the other in the half-step mode, the slant angles become 26°34' or 63°27', corresponding to the tangent of 0.5 and 2, respectively.

Elementary routine

A number of routines and algorithms are given below to provide a basis for developing one's own software. It should be noted that the information given is intended as guidance for those who have little or no experience in handling computer graphics. It is beyond doubt that there are other, perhaps more efficient, ways of controlling the plotter, but the methods outlined here have the advantage of being illustrative and relatively simple to put in practice on a particular computer system.

The suggested elementary routine does what its name implies: it provides control of the most fundamental capabilities of the plotter. Depending on the structure of the 8-bit control word sent along as a parameter, a single full or half step is performed in the X or Y direction, and/or a particular pen is selected. Bits 0 and 3 determine which motor, or which motors, is or are actuated. A step is performed by the relevant motor when the associated clock bit goes logic low. Direction of travel and full/half-step operation are controlled by the remaining four bits.

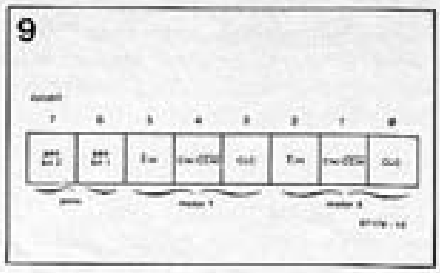


Fig. 9. Bit assignment in the plotter control word.

10

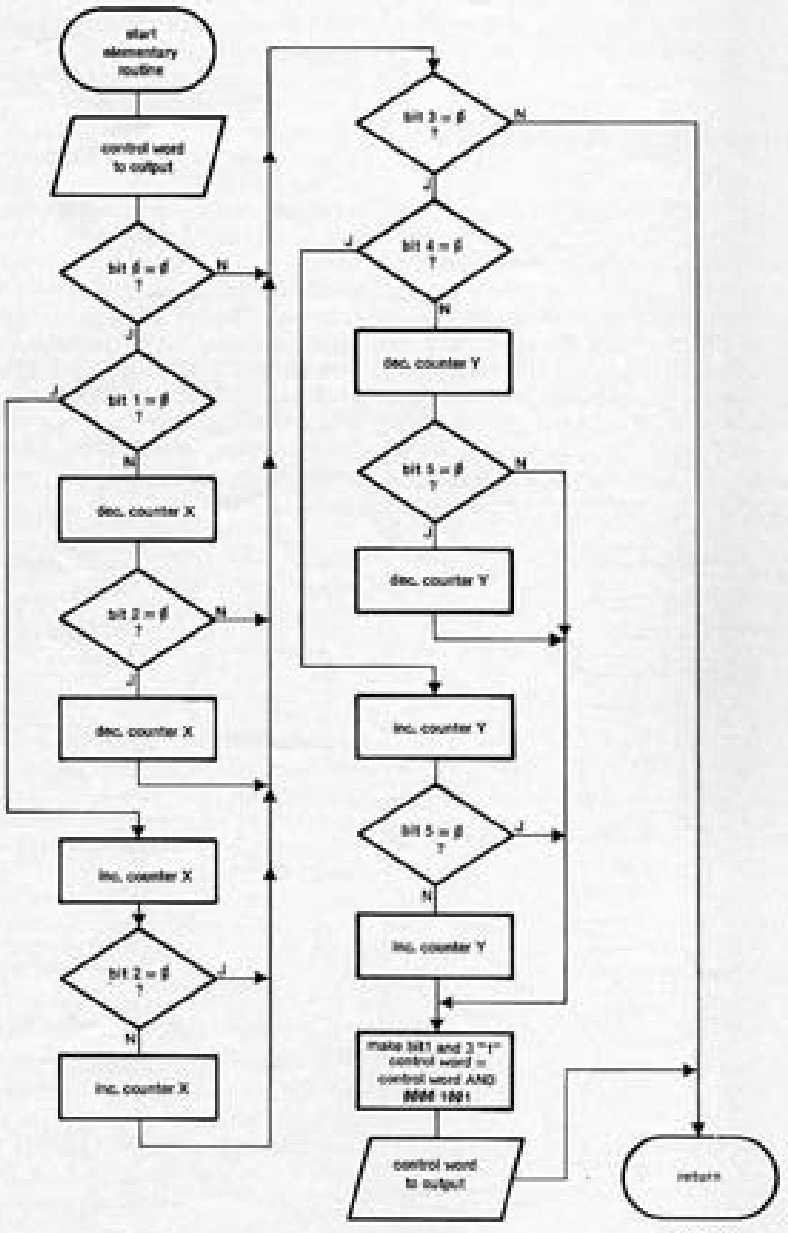


Fig. 10. Flow chart of the elementary routine. Instantaneous X and Y coordinates are stored in 16-bit counters.

11

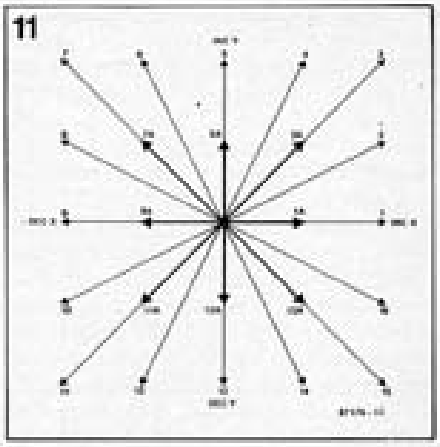


Fig. 11. Oblique lines of certain fixed angles are relatively simple to draw using the control words listed in Table 1.

Table 1.

vector (Fig. 11)	control word BIN. 7 6 5 4 3 2 1 0	HEX.	DEC.
1	x x x x 1 0 0 0	08	08
1A	x x x x 1 1 0 0	0C	12
2	x x 1 0 0 0 0 0	20	32
3	x x 0 0 0 0 0 0	00	00
3A	x x 1 0 0 1 0 0	24	36
4	x x 0 0 0 1 0 0	04	04
5	x x 0 0 0 x x 1	01	01
5A	x x 1 0 0 x x 1	21	33
6	x x 0 0 0 1 1 0	06	06
7	x x 0 0 0 0 1 0	02	02
7	x x 1 0 0 1 1 0	26	38
8	x x 1 0 0 0 1 0	22	34
9	x x x x 1 0 1 0	0A	10
9A	x x x x 1 1 1 0	0E	14
10	x x 1 1 0 0 1 0	32	50
11	x x 0 1 0 0 1 0	12	18
11A	x x 1 1 0 1 1 0	36	54
12	x x 0 1 0 1 1 0	16	22
13	x x 0 1 0 x x 1	11	17
13A	x x 1 1 0 x x 1	31	49
14	x x 0 1 0 1 0 0	14	20
15	x x 0 1 0 0 0 0	10	16
15A	x x 1 1 0 1 0 0	34	52
16	x x 1 1 0 0 0 0	30	48
pen selection			
pen 1	0 0 x x 1 x x 1	09	09
pen 2	0 1 x x 1 x x 1	49	73
pen 3	1 0 x x 1 x x 1	89	137
lift all pens	1 1 x x 1 x x 1	C9	201

x = don't care;
assumed logic 0 here

The flowchart of Fig. 10 shows that the control word is first sent to the output port that drives the plotter interface board. Full/half-step operation and direction of travel are set, and the clock input is programmed logic low. Next, bits 0 and 3 are examined to determine which motor is to perform a step. A 16-bit counter set up for the relevant motor is updated to keep track of the instantaneous coordinate. Direction of travel and full/half-step operation is taken into account as the counter is decremented or incremented. Bits 0 and 3 are made logic high, and the resultant control word is once again written to the output port. The selected motor(s) will thereupon perform one (full or half) step.

The two most significant bits control pen selection. In most cases, it will not be desired to perform a step while a pen is being selected, requiring bits 0 and 3 to be made logic high. Bit levels are frequently examined in the course of the elementary routine. The Type Z80 microprocessor offers special bit checking instructions. These can be simulated by other processors by, for example, logic ANDing of the control word with a mask byte in which the bit to be examined is logic high. The result of the check can then be read from the status of the zero-flag.

Straight lines and pen selection

A small addition to the elementary routine makes it possible to draw straight lines at certain fixed slant angles. To begin with, the command word is set up, taking the bits for pen selection into account. The desired line length can be

related to a specific X or Y coordinate. Each step is followed by a check for arrival at the end position. When this has not yet been reached, the next step is performed after a short delay. The delay time can be generated with the aid of a simple software loop, or a timer as available in, for instance, the 6522-VIA or Z80-CTC. A hardware timer has the advantage of making the final step rate, within limits, independent of the program routine that is to be executed between two steps. It is the task of the programmer to ensure that the motors operate smoothly in both the full and the half-step mode. Motor control can be enhanced by programming equal accceleration and deceleration rates for both motors when these are being stopped and started. This reduces the risk of one motor lagging because it misses out on a few steps, and in addition keeps longitudinal vibration of the pen carriage to a minimum (this effect is caused by inertia in combination with elasticity of the string).

The wind-rose shown in Fig. 11 is drawn by actuating one or both motors, in combination with direction of travel and full/half-step operation. The number of full or half steps is always constant. Reverse the polarity of one stator when a motor revolves in the wrong direction. Table 1 may be examined to see how the various control words for drawing the wind-rose were built from individual command bits.

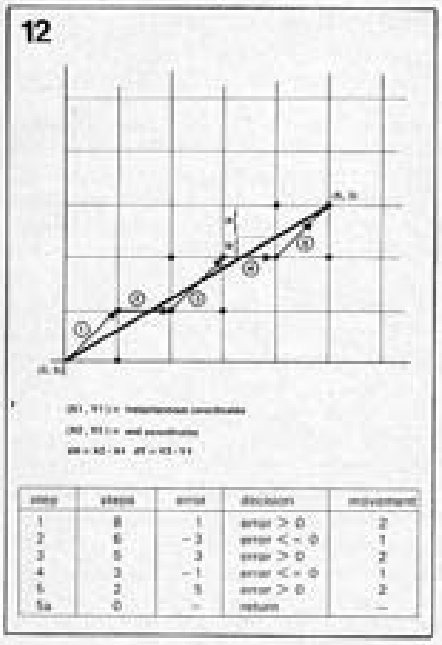
Bits 6 and 7 allow four logic combinations: three for putting the invidual pens on paper, and one (combination 11₂) for lifting all three pens simultaneously. Pen-down commands are preceded by a small, fixed, displacement in the X direction (offset, 58 or 116 steps) to compensate the distance between the pens in the carriage.

Random lines: Bresenham's algorithm

The drawing of oblique lines under slant angles other than the fixed ones discussed above is relatively complex. In most graphics applications, the working area is considered a system of coordinate axes. In this, a plotter should be able to draw a straight line between two random coordinates. In practice, however, the line drawn by the plotter will deviate from the desired, ideal, line owing to the limited number of discrete pen positions. Bresenham's line algorithm allows close approximation of the ideal line between random points in the coordinate system.

The drawing and Table in Fig. 12 illustrate the theory behind Bresenham's line algorithm. It is assumed that a line is to be drawn from starting point X1,Y1 — set at coordinates 0,0 for convenience's sake — and destination X2,Y2 at coordinate 5,3. Assuming the slant angle of the line to be smaller than 45° (Y2≦X2), the line can be drawn by actuating the X motor one step per increment, or the X and Y motor simultaneously. The choice between these options is determined by the difference between *a* and *b*. When *a* is greater than *b*, only the X motor is actuated, else the X and Y motor simultaneously. In essence, the procedure entails measuring the

12



step	steps	error	decision	movement
1	5	1	error > 0	2
2	5	−3	error <= 0	1
3	5	3	error > 0	2
4	3	−1	error <= 0	1
5	2	5	error > 0	2
5a	0	-	return	—

Fig. 12. Bresenham's line algorithm. The ideal line is shown in bold print, the dots in the raster form the discrete positions that can be reached by the pen. The choice between stepping in the X or Y direction, or stepping obliquely (X and Y simultaneously) is made after calculating the difference between *a* and *b*.

13

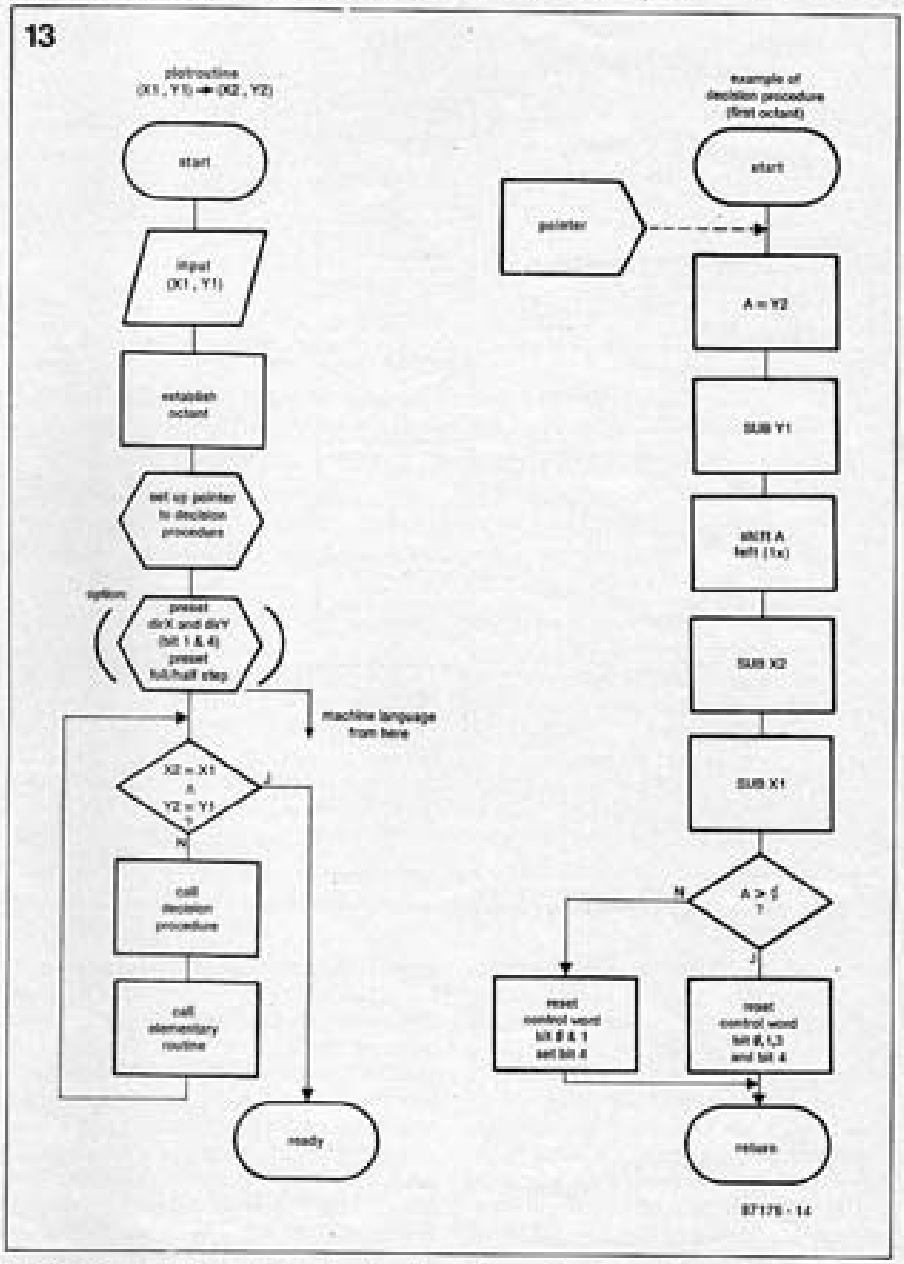


Fig. 13. Suggested flow chart for drawing lines to Bresenham's algorithm. The right-hand sequence is an example of one of the eight decision routines listed in Table 2.

angle of the line that can be drawn between the instantaneous and destination coordinates. When this angle is greater than 22°30' (2dY−dX>0), the next discrete position, X+1,Y+1, is stepped to at an angle of 45°. Otherwise, only the X motor performs a step.

The above algorithm is attractive because it allows simple calculations to be used for the decision procedure. Displacements dX and dY are deduced by subtraction, while multiplication by two is effected at machine code level by a single shift-left operation in the accumulator.

The same algorithm can be used for lines of angles between 45° and 90°, provided X and Y are exchanged. Lines in the remaining three quadrants are also fairly simple to draw to the above method. It is necessary, however, to determine beforehand in which octant (half quadrant) the destination coordinate will be with respect to the start-coordinate.

The flow diagram of Fig. 13 shows how lines between random coordinates can be drawn using Bresenham's algorithm. A routine is included to find out in which octant the destination coordinate is going to be with respect to the start-coordinate. Depending on the result, a pointer is preset to point to one of eight decision routines listed in Table 2a. In these, the control word is set up to define which motor (or motors) is to perform a step in a certain direction.

The actual stepping is done by calling the elementary routine. After each step, the instantaneous coordinates are compared to the destination coordinates (X2,Y2).

Algorithm for octant one

Bresenham's line algorithm derives step information from the distance to be covered in the X and Y direction (dX and dY respectively). The algorithm for the first octant (angle between 0 and 45°) is shown in Table 3. First, dX and dY are calculated to obtain the initial value of decision variable "error", which must be corrected (updated) after each step. Depending on the direction of travel, "error" is corrected with d-error1 (after movement 1) or d-error2 (after movement 2). Variable "steps" holds the number of steps to be performed in the X and Y direction, and is used for stopping the plot routine in time. The actual plotting is done in a WHILE-DO-loop. Depending on the value of "error", steps are straight (X or Y) or oblique (X and Y). Variable "steps" is decreased by one or two in accordance with the movement performed (remember that one oblique step is one step in the X direction and one in the Y direction, i.e. two steps in all).

The Table in Fig. 12 and the flowchart in Fig. 13 illustrate the operation of the algorithm with the aid of some variables. As already stated, the routine is only valid for the first octant. It is, however, fairly simple to modify for drawing lines in other octants. Depending on the octant in which the line is drawn, it will be necessary to:

- use the absolute value of dX and/or dY;
- swap dX and dY;
- adapt the two elementary movements.

Table 2a provides an overview of the above functions for each of the eight octants. The drawing of a line between two points in an arbitrary octant requires an extended version of the line routine. The flow diagram of this is shown in Fig. 14. The first part of the program (up to ATTENTION) is, in fact, a programmed version of Table 2a. This part of the routine ensures that the actual plot routine (the loop at the end) draws a line in the correct direction. The calculation of "error" is scattered over several branches, but is still in accordance with Table 2a when the decision routine is called.

The listing in Table 4 is a Pascal procedure written after the flow-chart of Fig. 14. It should be noted that the program is intended to draw lines on a computer screen, so that instantaneous coordinates X and Y are read and updated for use as end criterions. Variables

Table 2a

octant	Δa	Δb	movement 1		movement 2	
0 . . . 45°	+dX	+dY	inc.X	—	inc.X	inc.Y
45 . . . 90°	+dY	+dX	—	inc.Y	inc.X	inc.Y
90 . . . 135°	+dY	−dX	—	inc.Y	dec.X	inc.Y
135 . . . 180°	−dX	+dY	dec.X	—	dec.X	inc.Y
180 . . . 225°	−dX	−dY	dec.X	—	dec.X	dec.Y
225 . . . 270°	−dY	−dX	—	dec.Y	dec.X	dec.Y
270 . . . 315°	−dY	+dX	—	dec.Y	inc.X	dec.Y
315 . . . 360°	+dX	−dY	inc.X	—	inc.X	dec.Y

dX = X2 − X1 — X1, Y1 = start coordinates
dY = Y2 − Y1 — X2, Y2 = destination coordinates
ERROR = 2 · Δb − Δa — initial error
dERROR1 = 2 · Δb — error change after movement 1
dERROR2 = 2 · Δb − 2 · Δa — error change after movement 2

Table 2b.

	control word
inc. x	x x x x x x x 0 0
dec. x	x x x x x x x 1 0
inc. y	x x x 0 0 x x x
dec. y	x x x 1 0 x x x

Table 3.

```
dX = X2 - X1
dY = Y2 - Y1
error = 2dY - dX
derror1 = 2dY
derror2 = 2dY - 2dX
steps = dX + dY

WHILE steps > 0 DO
  IF error <= 0 THEN
                    step X
                    error = error + derror1
                    steps = steps - 1
                  ELSE
                    step XY
                    error = error + derror2
                    steps = steps - 2
                  ENDIF
ENDWHILE
```

14

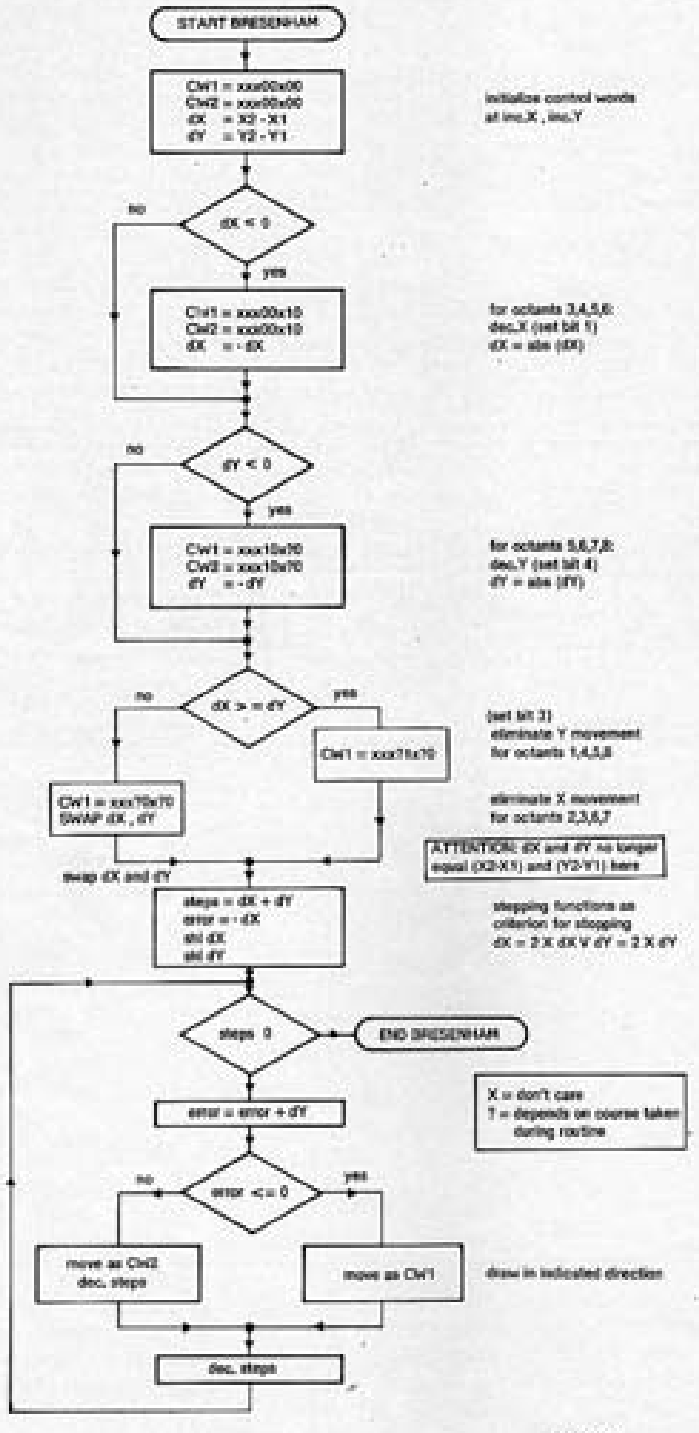


Fig. 14. Flow diagram of the extended line drawing routine.

STEP1 and STEP2 correspond to control word 1, and variables STEP3 and STEP4 to control word 2 in the flow diagram.

Circles and ellipsoids

The plotter will have to draw circles frequently. A set of coordinates of a circle can be computed with the aid of two tables: one holds data of one period of a sine function, the other data of one period of a cosine function. Table entries are rounded off to the nearest integer. The sine table then holds X coordinates, the cosine table Y coordinates. The amplitudes form the radius in the X and Y direction. Equal amplitudes result in a circle, unequal amplitudes in an ellipsoid whose major axis runs in parallel with the X or Y axis. Ellipsoids which are oblique with respect to the X or Y axis are obtained by mutual shifting of the tables. This effectively creates phase shift variation.

Calculation of coordinates lays a rather heavy claim on processor time, and is, therefore, done beforehand. The result, in the form of two tables, is stored in memory. The circle can then be drawn by having the plotter step from point to point using the Bresenham algorithm.

Extending the control program

The previously discussed elementary routine and general algorithms should enable programmers to develop a suitable control program for their computer. The bulk of the plotter control program may be written in a higher programming language, but there is no way to go round machine code for time critical routines. The final program should enable drawing

- lines between arbitrarily chosen coordinates (absolute function);
- lines between the current pen position and a coordinate defined with respect to that position (relative function);
- standard figures such as circles, squares, etc.;
- characters (letters, symbols and numbers).

Each character should have a corresponding set of relative coordinates, which can be multiplied by a fixed factor for enlarging or reducing character size.

TW

```
PROCEDURE BRESENHAM (X1,Y1,X2,Y2): WORD; ATTRIBUTE: CHAR; PAGE: BOOLEAN);

VAR X, Y, dX, dY, ERROR, STEP1, STEP2, STEP3, STEP4: WORD;

BEGIN
  dX := X2 - X1;
  dY := Y2 - Y1;
  STEP1 := 1; {initialize all steps at +1}
  STEP2 := 1;
  STEP3 := 1;
  STEP4 := 1;
  IF dX < 0 THEN BEGIN {initialize for octants 3, 4, 5, 6}
                     STEP1 := -1; {step backwards in X direction}
                     STEP3 := -1;
                     dX := -dX     {dX := ABS (dX)}
                 END
  IF dY < 0 THEN BEGIN {initialize for octants 5, 6, 7, 8}
                     STEP2 := -1; {step backwards in Y direction}
                     STEP4 := -1;
                     dY := -dY     {dY := ABS (dY)}
                 END
  IF dX >= dY THEN {eliminate Y direction in movement1 for octants
                      1, 4, 5, 8, and initialize decision variables.}
                     STEP2 := 0;
              ELSE {eliminate X direction in movement1 for octants
                      2, 3, 6, 7, and initialize decision variables.}
                     BEGIN
                       STEP1 := 0;
                       {dX and dY must be swapped.
                        ERROR serves as an auxiliary variable}
                       ERROR := dX; dX := dY; dY := ERROR;
                     END;
  {start plotting algorithm}
  X := X1; {make instantaneous and start coordinates equal}
  Y := Y1;
  ERROR := -dX;
  dX := 2 * dX; {these two lines prevent}
  dY := 2 * dY; {multiplications in the loop}
  HPLOT (X,Y,ATTRIBUTE,PAGE); {plot first pixel on screen}
  WHILE (X<>X2) OR (Y<>Y2) DO
  BEGIN
    ERROR := ERROR + dY;
    IF ERROR <= 0 THEN BEGIN {movement 1}
                         X := X + STEP1;
                         Y := Y + STEP2;
                       END
                  ELSE BEGIN {movement 2}
                         X := X + STEP3;
                         Y := Y + STEP4;
                         ERROR := ERROR - dX;
                       END;
    HPLOT (X,Y,ATTRIBUTE,PAGE) {plot pixel X,Y on screen}
  END
END;
```

Computer user groups and individual programmers are invited to write universally applicable software drivers for the plotter described. Publication in this magazine can be arranged in cooperation with the editor and a technical assessment committee. We are particularly interested in programs to the Hewlett Packard Graphics Language (HPGL) standard for the following computers: Acorn Archimedes, Acorn Electron, Amstrad CPC464, Atari ST, BBC B, Commodore Amiga, *Elektor Electronics* BASIC computer, IBM PC XT/AT and compatibles (Amstrad 1512/1640), MSX-2, Sinclair Spectrum and Quantum Leap.

Writers who have submitted programmes found suitable for publication are offered a publication fee and a year's subscription to this magazine. The closing date for sending in suggested programs is 1 December 1988.

Editorial Office
1 Harlegin Avenue Harlegin
BRENTFORD TW8 9EW
England.

Rain Synthesiser

This simple circuit has proved very reliable and effective as a background sound effect generator for use by organists etc.
Other simple devices of this type often use several stages of amplification or make use of special noise diodes which are comparatively expensive. The advantage of this design is that it employs an ordinary OA 91 or similar diode. The internal noise produced by the diode is amplified by the single stage preamplifier, consisting of T1 and its associated components, which is designed for high gain and low cost. T1 can be almost any silicon NPN transistor, a BC107 being used in the prototype.
The output at X may be taken straight to an amplifier if only a white noise output is required. However, the addition of the passive filter, comprising C3 and P1 enables a variety of effects ranging from light rain to a heavy storm to be obtained.