

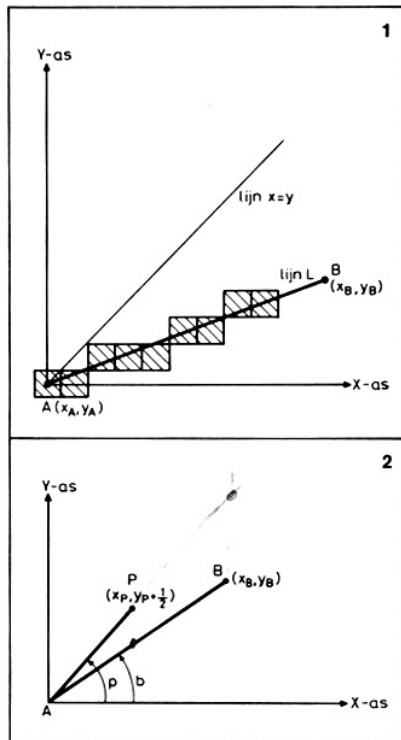


LINES

On the graphical display

R. Koekoek

A number of articles discuss software for the graphic display published in Radio Bulletin. This article describes a program that draws straight lines on the graphic display. Only the start and end points of the line need to be entered; the program then draws the line.



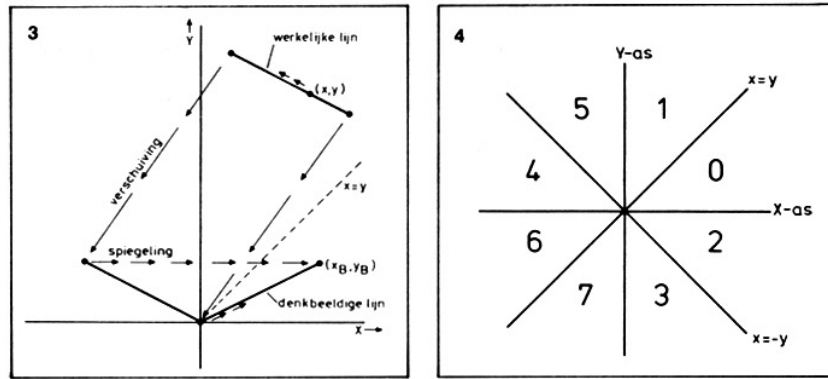
Background

The graphic display shows figures by switching discrete dots on or off. To draw a straight line at an arbitrary angle we must therefore settle for the best possible approximation using individual dots, as shown in fig. 1.

The aim of the program is to have the computer draw lines. To start simply, we assume a line that begins at the origin (point A) with coordinates x_A, y_A and is drawn to a point B with coordinates x_B, y_B . We further assume that the angle with the positive x-axis lies between 0 and 45 degrees. This means that $x_B \geq 0, y_B > 0$ and $x_B \geq y_B$.

A line with these restrictions can be shown on the display by moving to the right from the origin, turning dots white, and stepping up one dot now and then to follow the line. The program must now decide when to go right and when to go up.

The following method gives a good result, as can be seen in fig. 1. We want to draw the ideal line L between A and B, and begin by switching on the dot at A. We then move one dot to the right. If this dot is more than half a unit of length (the distance between two dots on the display) away from the ideal line, we move one dot up. We repeat this process until we finally arrive at B.



Formula

We need a formula, easy for the computer to calculate, that tells us whether or not a dot lies half a unit of length above the ideal line. Looking at fig. 2, we see that a point P with coordinates X_p, Y_p lies above the line between A and B if the angles p and b satisfy:

$$p > b$$

or, using the familiar definition:

$$\tan(p) = X_p / Y_p : [\text{sic; should be } Y_p / X_p]$$

$$\tan(p) > \tan(b)$$

Whether a dot lies half a unit of length above the ideal line now follows from:

$$(Y_p + 1/2) / X_p > Y_b / X_b$$

This equation contains divisions, which the computer does not like much. Multiplying both sides by $X_p \cdot X_b$ gives:

$$(Y_p + 1/2) \cdot X_b > Y_b \cdot X_p$$

Moving everything to the left and multiplying by 2 gives:

$$2Y_p \cdot X_b + X_b - 2Y_b \cdot X_p > 0$$

This equation contains a number of multiplications, which are time-consuming, so it is still not very suitable. We now call the left-hand side T:

$$T = 2Y_p \cdot X_b + X_b - 2Y_b \cdot X_p$$

For the first point on the line $X_p = Y_p = 0$, so:

$$T = X_b \text{ (first point)}$$

For the next point there are two possibilities: we move one dot to the right or one dot up.
In other words:

1. X_p is incremented by one, so the T for this new point (T_{new}) can be calculated from the old T (T_{old}) with:

$$T_{\text{new}} = T_{\text{old}} - 2Y_b$$

2. Y_p is incremented by one, so T_{new} follows from:

$$T_{\text{new}} = T_{\text{old}} + 2X_b$$

By remembering the T value of the previous point, we can calculate the T value of the new point by adding and subtracting the constants $2X_b$ and $2Y_b$, without involving the coordinates X_p , Y_p .

If T is positive, the point with coordinates X_p , $Y_p + 1/2$ lies above the ideal line and moving right is sufficient. Otherwise we must also move one dot up. The line-drawing algorithm is based on this formula.

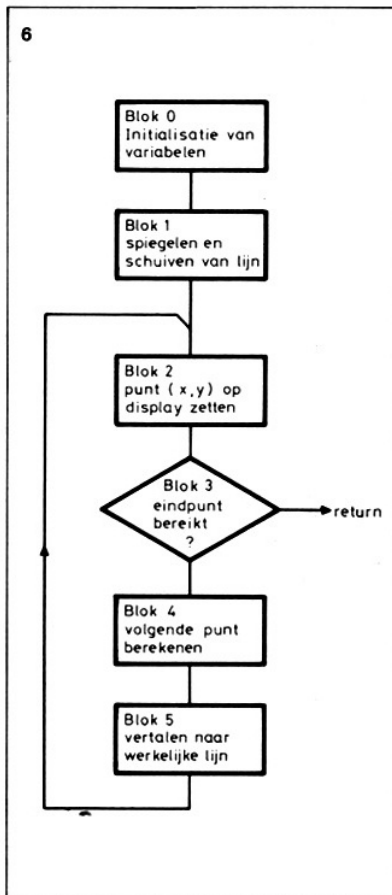
Drawing a line then goes as follows. We start at the origin by switching that dot on, and move one dot to the right. Using the formula derived here, we decide whether or not we must also move one dot up. The T value and the coordinates of each drawn dot are then updated and the dot is drawn. We continue until we have reached the end point.

Drawing arbitrary lines

The algorithm we have developed so far only works for lines that start at the origin and make an angle of between 0 and 45 degrees with the positive x-axis.

Even so, we can tackle an arbitrary line with this simple algorithm. We do the calculations in a coordinate system in which the line does meet the conditions. Relative to the display coordinate system, this calculation coordinate system is mirrored and shifted. By determining the next point in the calculation coordinate system and then translating it to the display coordinate system, we have an algorithm for arbitrary straight lines.

Fig. 3 shows an example of such a shift and mirroring to make the line meet the conditions. In this example, after shifting the start point to the origin, one mirroring was sufficient.



Program

Listing 1 contains the assembler listing of the program "lijn" (line); fig. 6 shows how the program is built up from blocks.

The program is written as a subroutine.

The program uses the following variables:

Xbegin, Ybegin = coordinates of the start point of the line.

Xeind, Yeind = coordinates of the end point of the line.

XpuntL, XpuntH = the number $2 \times Xb$.

YpuntL, YpuntH = the number $2 \times Yb$.

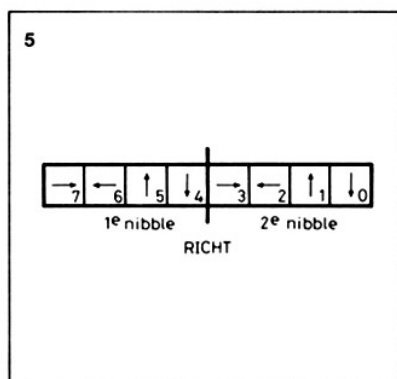
TL, TH = Tsum as described above.

RICHT = variable in which the mirroring is stored.

X, Y = coordinates of the actual dot (registers).

For the X and Y coordinates of the start and end points, 8-bit numbers are sufficient, since the graphic display is 256 by 256 dots. Tsum, Xpunt and Ypunt take values between 0 and 65535, so 16-bit numbers are used for these.

The blocks do the following:



Block 0. Lines 380 – 420.

Variables are initialised here, for example by clearing the most significant bytes of Xpunt, Ypunt and T. The Y register is also cleared.

Block 1. Lines 430 – 770.

The purpose of this block is to translate the line into a line that starts at the origin and makes an angle of between 0 and 45 degrees with the positive x-axis.

It starts with shifting: Xpunt (Xb, Yb of the line in the calculation coordinate system) is set equal to End – Begin.

If the x coordinate of Xpunt is negative, we must mirror about the Y-axis and record this in bit 2 of the Y register. If the Y coordinate is negative, we mirror about the x-axis and record this in bit 1 of the Y register.

The angle now lies between 0 and 90 degrees and we are nearly there.

If the Y coordinate is greater than the x coordinate, we must mirror about the line $x=y$ to get an angle between 0 and 45 degrees. We record this mirroring in bit 0 of the Y register.

The Y register now contains a number that indicates the direction of the actual line. Fig. 4 shows how this number relates to the angle with the positive x-axis. Using the Y register as an index, a number is fetched from TABEL (table); its meaning will become clear shortly. This number is stored in the variable RICHT.

Because the values $2 \cdot X_b$ and $2 \cdot Y_b$ are needed to calculate T, they are calculated once by shifting Xpunt and Ypunt to the right [sic; the code shifts left (ASL/ROL), i.e. multiplies by 2].

The X and Y registers are loaded with the start point of the line. For the rest of the routine, the X and Y registers always hold the coordinates of the actual dot currently being processed.

Block 2. Lines 810 – 890.

Here the dot is switched on on the display. The X and Y registers contain the coordinates of the current dot on the line, and that dot is drawn using the control codes of the graphic display. This assumes, of course, that the PIAs to which the graphic display is connected have already been initialised.

Block 3. Lines 900 – 940.

This block decides whether the whole line has been drawn. If the X or Y register equals the x or y coordinate of the end point, the end of the line has been reached and we return from the subroutine. [Note: the code actually requires both X and Y to match the end point.]

Block 4. Lines 980 – 1130.

The next point on the line in the calculation coordinate system is now calculated. First we move one dot to the right and update Tsum by subtracting $2 \cdot Y_b$. If T becomes negative as a result, we must also move one dot up, and account for this by adding $2 \cdot X_b$ to T. The sign bit records whether or not we moved up.

Block 5. Lines 1150 – 1330.

The movement in the calculation coordinate system from block 4 must still be translated into a movement along the line on the display.

For this, the value of the variable RICHT is used. RICHT was obtained by recording the mirrorings and using them as an index to fetch a constant from a table. If we have only moved right along the line in the calculation coordinate system, we use the first nibble of RICHT, otherwise the second.

What such a nibble represents is shown in fig. 6 [sic; fig. 5]. A 1 in the position of an arrow means that the dot is moved in the direction of that arrow. The table is arranged so that each nibble corresponds to the calculated mirrorings.

The X and Y registers, which hold the coordinates of the dot on the display, are updated using the nibble, translating the movement along the calculation line into a movement along the actual line.

We now jump to block 2 to show the calculated dot and, if necessary, calculate the next one.

Example

Listing 2 contains an example of how to use the subroutine "lijn". It draws lines from a point to the origin (the centre of the graphic display). We start in the top right corner and move one dot at a time to the left until the left edge is reached. The subroutine "lijn" draws a line by inverting, which produces an attractive interference pattern where lines nearly overlap.

In the example, the PIA registers are first initialised and the screen is cleared (lines 360 – 450).

Then the start and end coordinates are filled in and the lines are drawn, with the end point being moved to the left.

The routine ends by jumping to the monitor. This example was implemented on a KIM-1, but only the PIA locations depend on that.

Captions:

Fig. 1. Approximation of a line by discrete dots on the graphic display.

Fig. 2. Point above the ideal line, with the angles p and b between the x-axis and the lines from the origin to $(X_p, Y_p + 1/2)$ and to the end point (X_b, Y_b) respectively.

Fig. 3. Mirroring and shifting to transform an arbitrary line into one that is easy to calculate with.

Fig. 4. Value in the Y register indicating the mirrorings for lines between the boundaries shown.

Fig. 5. Meaning of the bits in the variable RICHT.

Fig. 6. Structure of the program in blocks.

Listing 1. Listing of the subroutine "lijn".

Listing 2. Example of the use of "lijn".

Lijst 1

LIJN HUCO 81 6502 ASSEMBLER V2.0 PAGE 01

```

0010: *****
0020: *
0030: * LIJNEN OP HET GRAFISCH DISPLAY *
0040: *
0050: * R. KOEKDEK *
0060: *
0070: * COMPUTER BULLETIN *
0080: *
0090: *****
0100: ;
0110: 0400
0120: ;
0130: ; ZERO PAGE VARIABLEN :
0140: ;
0150: 0400
0160: 0400
0170: 0400
0180: 0400
0190: 0400
0200: 0400
0210: 0400
0220: 0400
0230: 0400
0240: 0400
0250: 0400
0260: 0400
0270: ;
0280: ; PIA ADRESSEN
0290: ;
0300: 0400

```

```

0700: 0439 85 9A STA RICHT ;
0710: 0438 86 96 ASL YPUNT ; YPUNT := YPUNT * 2
0720: 043D 26 97 ROL YPUNT ;
0730: 043F 86 98 STX TL ; T := XPUNT
0740: 0441 86 94 ASL XPUNT ; XPUNT := XPUNT * 2
0750: 0443 26 95 ROL XPUNT ;
0760: 0445 86 90 LDX YBEGIN ; BEGINPUNT OP DISPLAY ZETTEN
0770: 0447 84 91 LDY YBEGIN ;
0780: ;
0790: ; PUNT X.Y OP DISPLAY ZETTEN
0800: ;
0810: 0449 8D 02 17 WAIT LDA PBD ; WACHT OP BLANKING
0820: 044C 10 FB BPL WAIT ; HOOG WOORD
0830: 044E 8E 00 17 STX PBD ; X-COORDINAAT
0840: 0451 EE 02 17 INC PBD ; 01 OP STUURUITGANG
0850: 0454 EE 02 17 INC PBD ; 02 " "
0860: 0457 8C 00 17 STY PBD ; Y-COORDINAAT
0870: 045A 89 08 LDAM #08 ;
0880: 045C EE 02 17 INC PBD ; 03 OP STUURUITGANG
0890: 045F 8D 02 17 STA PBD ; INVERTEER COMMANDO
0900: 0462 E4 92 CPX YEND ; EINDPUNT BEREIKT ?
0910: 0464 D0 05 BNE VERDER ;
0920: 0466 C4 93 CPY YEND ;
0930: 0468 D0 01 BNE VERDER ;
0940: 046A 60 RTS ; EINDE LIJN IS BEREIKT
0950: ;
0960: ; VOLGENDE PUNT BEREKENEN
0970: ;
0980: 046B 38 VERDER SEC ;
0990: 046C 85 98 LDA TL ; T := T - YPUNT
1000: 046E E5 96 SBC YPUNT ; EEN PUNT NAAR RECHTS OVER LIJN
1010: 0470 85 98 STA TL ;
1020: 0472 85 97 LDA TH ;
1030: 0474 E5 97 SBC YPUNT ;
1040: 0476 85 99 STA TH ;
1050: 0478 08 PHP ; BEWAAR SIGN BIT
1060: 0479 10 00 BPL SKIP ; IS T NEGATIEF ?
1070: 047B 18 CLC ;
1080: 047C 85 98 LDA TL ; T := T + XPUNT

```

Lijst 2

VOORB HUCO 81 6502 ASSEMBLER V2.0 PAGE 01

```

0010: *****
0020: *
0030: * VOORBEELD VAN HET GEBRUIK VAN LIJN *
0040: *
0050: *
0060: *****
0070: ;
0080: 04B0
0090: ;
0100: ; ZERO PAGE GEBRUIK
0110: ;
0120: 04B0
0130: 04B0
0140: 04B0
0150: 04B0
0160: 04B0
0170: 04B0
0180: ;
0190: ; MONITOR ADRESSEN KIN
0200: ;
0210: 04B0
0220: 04B0
0230: ;
0240: ; PIA ADRESSEN
0250: ;
0260: 04B0
0270: 04B0

```

```

0310: 0400
0320: 0400
0330: 0400
0340: 0400
0350: ;
0360: ; ROUTINE LIJN
0370: ;
0380: 0400 D8 LIJN CLD ;
0390: 0401 80 00 LDVIM #00 ;
0400: 0403 84 95 STY YPUNT ; X.Y PUNT H := 0
0410: 0405 84 97 STY YPUNT ;
0420: 0407 84 99 STY TH ; T H := 0
0430: 0409 38 SEC ;
0440: 040A 85 92 LDA YEND ; XPUNT := YEND - YBEGIN
0450: 040C E5 90 SBC YPUNT ;
0460: 040E 85 94 STA YPUNT ;
0470: 0410 80 08 BCS YVERG ; XPUNT NEGATIEF ?
0480: 0412 89 01 LDAM #01 ; SPIEGELEN OM X-AS
0490: 0414 E5 94 SBC YPUNT ;
0500: 0416 85 94 STA YPUNT ;
0510: 0418 80 04 LDVIM #04 ; SPIEGELING ONTHOUDEN
0520: 041A 8A TRX ; XPUNT BEWAREN
0530: 041C 38 SEC ;
0540: 041E 85 93 LDA YEND ; YPUNT := YEND - YBEGIN
0550: 041F E5 91 SBC YPUNT ;
0560: 0420 85 96 STA YPUNT ;
0570: 0422 80 08 BCS LZ ; YPUNT NEGATIEF ?
0580: 0424 89 01 LDAM #01 ;
0590: 0426 E5 96 SBC YPUNT ; SPIEGELEN OM X-AS
0600: 0428 85 96 STA YPUNT ;
0610: 042A C8 INY ; SPIEGELING ONTHOUDEN
0620: 042B C8 INY ;
0630: 042C C5 94 CMP YPUNT ; XPUNT > YPUNT ?
0640: 042E 90 06 BCC LZ ;
0650: 0430 86 96 STX YPUNT ; XPUNT EN YPUNT VERWISSELEN
0660: 0432 85 94 STA YPUNT ;
0670: 0434 8A TRX ;
0680: 0435 C8 INY ; VERWISSELING ONTHOUDEN
0690: 0436 B9 A3 04 LZZ LDARY TABEL ; HAAR RICHTING OP

```

```

1090: 047E 65 94 ADC YPUNT ;
1100: 0480 85 98 STA TL ;
1110: 0482 85 99 LDA TH ;
1120: 0484 65 95 ADC YPUNT ;
1130: 0486 85 99 STA TH ;
1140: ;
1150: 0488 A5 9A SKIP LDA RICHT ; RICHTING
1160: 048A 28 PLP ; SIGN BIT TERUG
1170: 048B 10 04 BPL RIGHT ; ALS NIET POSITIEF
1180: 048D 0A ASLA ;
1190: 048E 0A ASLA ; TWEEDE NIBBLE GEBRUIKEN
1200: 048F 0A ASLA ;
1210: 0490 0A ASLA ;
1220: 0491 0A RIGHT ASLA ;
1230: 0492 90 01 BCC LEFT ;
1240: 0493 E8 TAX ; PLATS NAAR RECHTS
1250: 0495 0A LEFT ASLA ;
1260: 0496 90 01 BCC DOWN ;
1270: 0498 0A DEX ; PLATS NAAR LINKS
1280: 0499 0A DOWN ASLA ;
1290: 049A 10 01 BPL UP ;
1300: 049C 88 DEY ; PLATS ONLAAG
1310: 049D 90 0A UP BCC WAIT ;
1320: 049F C8 INY ; PLATS OMHOOG
1330: 04A0 4C 49 04 JMP WAIT ; ZET PUNT NAH
1340: ;
1350: ; TABEL MET RICHTING VAN LIJN
1360: ;
1370: 04A3 0A TABEL = #0A ;
1380: 04A4 2A = #2A ;
1390: 04A5 89 = #89 ;
1400: 04A6 19 = #19 ;
1410: 04A7 46 = #46 ;
1420: 04A8 26 = #26 ;
1430: 04A9 45 = #45 ;
1440: 04AA 15 = #15 ;
1450: ;
1460: ; EINDE LIJN

```

```

0280: 04B0
0290: 04B0
0300: 04B0
0310: 04B0
0320: 04B0
0330: ;
0340: ; VOORBEELD
0350: ;
0360: 04B0 A2 FF EXAMP LDVIM #FF ; INITIALISEER I/O VAN DISPLAY
0370: 04B2 8E 01 17 STX PBD ;
0380: 04B5 A2 1F LDVIM #1F ;
0390: 04B7 8E 03 17 STX PBD ;
0400: 04BA A9 1C LDAM #1C ; MAAK SCHERM SCHOON
0410: 04BC 85 00 STA TEMP ;
0420: 04BE 8D 02 17 LDA PBD ;
0430: 04C1 20 1F 1F CLS JSR SCANDS ; WACHT 2 MS
0440: 04C4 C6 00 DEC TEMP ;
0450: 04C6 D0 F9 BNE CLS ;
0460: ;
0470: ; TREK LIJN
0480: ;
0490: 04C8 A2 00 LDVIM #00 ; BEGIN PUNT #00.00
0500: 04CA 8E 02 17 STX PBD ;
0510: 04CD A9 00 LDAM #00 ;
0520: 04CF 86 92 STX YEND ;
0530: 04D1 86 93 STX YEND ; EIND PUNT #00.00
0540: 04D3 C8 DEX ;
0550: 04D4 86 91 STX YBEGIN ;
0560: 04D6 85 90 STA YBEGIN ;
0570: 04D8 20 00 04 LOOP JSR LIJN ; TREK LIJN
0580: 04DB E6 92 INC YEND ; YEND := YEND + 1
0590: 04DD D0 F9 BNE LOOP ;
0600: 04DF 4C 22 1C JMP MONIT ; GA DOOR TOT YEND = 0
0610: ;
0620: ; EIND VOORBEELD

```