5 Graphics Primer,

Finding your way around the colour and graphics of the *CPC464*

Subjects covered in this chapter:

* Screen modes a
* The colours
* INK, PAPE
* Drawing line
* WINDOWS

5.1 **Machine-specific features**

The description and applications of AMSTRAD BASIC have so far largely been based on an industry-standard specification. Most of the purely arithmetical operations will run with only slight adjustments from one 'dialect' of BASIC to another however, the BASIC commands that control the graphics (and the text cursor locations) are more specifically dedicated to the way in which the CPC464 hardware controls the screen display and must be carefully understood to get the most from your micro.

As usual BASIC K E Y W 0 R D S are shown displayed in the special computer display style typeface, further examples and a concise description of the keyword and its use can be found by reference to the appropriate entry in chapter 8.

5.1.1 Colour Selection

'Black' ie no colour or illumination is considered as a colour for the purposes of all following descriptions that describe the attributes of the colours and the various commands associated with them.

The B 0 R D E R can be set to a ANY pair of colours regardless of the screen mode, and is not reset when a MO D E command is issued. It may be set to flash, or to a single steady colour.

The number of available I N K s that may be displayed simultaneously depends on the screen M 0 D E selected. Each I N K can be set to a pair of colours ie flashing; or a single colour ie steady. The number of usable inks at any time depends on the screen mode as previously defined. The text PA P E R, text P E N, and graphics P E N can then be set to an available ink.

5.1.2 Transparent option and the relationship of P E N, I N K and PAPER.

Except for the conditions where the flashing alternate colours are specified, two I N Ks are used when writing to the screen; one I **NK** determines the colour of the P**EN**, while the other determines the colour of the P A P E **R**.
NB The number associated with ***PA P E R*** command is the I N K declared for that number - and NOT the colour number as listed in Appendix VI. Similarly the number associated with the **PEN** command is the I N K declared for that PEN number, and NOT the colour number as listed in Appendix VI.

The **PA P E R** number defaults to 0 if not specified, while the P E N number defaults to 1 if not specified. To set the I N K for PA P E R number 0 to green, which is colour number 9, you would type in:

```
INK 0,9
```

Similarly to set the I N K for ***P EN*** number 1 to black, which is colour number 0, you would type in:

```
INK 1,0
```

Setting the ***PAPER*** to the same **INK** as the **PEN** ie: ***INK 0,0*** will black out the entire display.

The text 'writing' can be set to be transparent or opaque using one of a series of control characters that provide useful extensions to the major BASIC graphics commands. With the transparent option, you can either ignore the paper colour and overwrite the graphics, or completely overwrite the background. This brief program illustrates the effect available:

[CTRL][SHIFT][ESC]

```
10 MODE 1
20 INK 2,19
30 DRAW 200,200,2
40 LOCATE 1,21
50 PRINT "1 NORMAL"
60 PRINT CHR$(22)+CHR$(1)
70 ORIGIN 0,0
80 DRAW 500,200,2
90 LOCATE 12,18
100 PRINT"2 TRANSPARENT"
110 PRINT CHR$(22)+CHR$(0)
120 LOCATE 22,15
130 PRINT"3 NORMAL AGAIN"
```

The first D RAW command in line 30 was performed before transparent mode was set, but the second was drawn after the C H R **$(2 2)+** C H **R$(1)** command to set transparent mode was issued in line 60. Note how the overlap points have changed colour, and how (in transparent mode) the I **NK** filling the character cell has been completely overwritten.

Swap the location of the transparency-on (line 60) and transparency-off (line 110) commands in the program and see how this affects the displayed results. A full list of these additional commands is given in Appendix VI.

5.2 Screen modes

There are three modes in which the screen (the text and graphics operation) functions:

a) Normal

Mode 1: 40 columns x 25 lines, 4 I N K text
320x200 pixels, individually addressable in 4 colours

b) Multicolour mode

Mode 0: 20 columns x 25 lines, 16 I N K text
160x200 pixels, individually addressable in 16 colours

c) High Resolution mode

Mode 2: 80 columns x 25 lines, 2 I N K text
640x200 pixels, individually addressable in 2 colours

As you can see, the difference is in the number of individual horizontal 'elements' of the display. Not to be confused with the small stripes on the face of the TV tube, which are a separate feature of the TV monitor hardware.

Each of the three different displays is referred to as the screen or display M 0 D E, and only one mode may be displayed at any given time using BASIC. Changing mode causes the screen to clear completely - including all text and graphics windows (the same effect as a C L S and C LG command), but does not affect the contents of the program memory.

Mode changes may be invoked from BASIC programs, or they may be entered using the direct commands.

5.2.1 M 0 D E Ø is the multicolour graphics display.

16 of the 27 colours available can be displayed simultaneously. The display consists of 160 pixels in each horizontal row, and 200 in each vertical column. A plan of this grid appears in Appendix VI.

In M 0 D E 0, there are 20 characters on each of 25 lines.

5.2.2 M 0 D E *1* is the 'standard' or default mode.

M 0 D E 1 is pre-set when the CPC464 is turned on. 4 of the 27 colours may be displayed simultaneously - although you can switch rapidly through all 27 if you want. The display is 320 pixels wide, by 200 high. A plan of this grid appears in Appendix VI.

In M 0 D E 1, there are 40 characters on each of 25 lines.

5.2.3 Mo*d e 2* is the high resolution mode.

MO ***D E*** 2 allows two colours to be used simlutaneously, and is used primarily for its ability to produce 80 text characters per line - which makes a program much easier to write, since you can see so much more of the program at a glance.

M 0 D E 2 provides 640 pixels per horizontal row, again with 200 in each vertical column.

5.2.4 Try this....

With the CPC464 fully reset using **[CTRL][SHIFT][ESC]**, type in this program:

```
5 REM GRAPHICS EXAMPLE DEMONSTRATION
10 MODE 1
15 INK 2,0
16 INK 3,6: REM SETS THE COLOUR USED IN LINE 90
17 BORDER 1: REM DARK BLUE
20 CLG: REM CLEAN UP THE DISPLAY
30 b%=RND*5+1 :REM SET UP PSEUDO
 RANDOM INTEGER VARIABLES
40 c%=RND*5+1
50 ORIGIN 320,200 :REM FIX THE GRAPHICS ORIGIN
60 FOR a = 0 TO 1000 STEP PI/30
70 x%=100*COS(a)
80 MOVE x%,x% :REM MOVE THE GRAPHICS CURSOR
90 DRAW 200*COS(a/b%),200*SIN(a/c%),3
 :REM DRAW THE LINE
91 IF INKEY$<>"" THEN 20
100 NEXT :REM BACK TO 60 UNLESS INTERRUPTED AT 91
110 GOTO 20
```

Now RUN the program. Hit 'any key' on the keyboard, to get another pattern. This demonstrates several important features of the CPC464's hardware and software: the CPC464 'writes the screen' very smoothly without judder or 'tearing', and the software includes commands that permit very sophisticated effects to be achieved with the minimum of effort. The ***REM*** statements (R EMarks) are simply there for your convenience, you don't need to include them for the program to work, it just helps you (and particularly people who did not write the program in the first place) to understand what's going on.

Note that several of the line numbers indicate 'afterthought' entries - and whilst we could tidy the listing by simply issuing a ***R ENUM*** command, it will help you to follow the way in which programs evolve and develop from their initial structures if we leave the original numbering.

S A V E this program on the cassette - eg:

```
SAVE "GRAPHICS 5.5.84"
```

This graphics demonstration plots a different coloured interference pattern:

```
new
10 a$=INKEY$: REM PRESS ANY KEY TO
 INITIATE A NEW PATTERN SEQUENCE
20 IF a$="" THEN 10
30 CLS
40 m=INT(RND*3):REM SELECT A RANDOM
 NUMBER BETWEEN 0 AND 3
50 IF m>2 THEN 40:REM TRY AGAIN
 IF THE VALUE EXCEEDS 2
60 MODE m
70 i1=RND*26:REM SELECT RANDOM INK VALUES
80 i2=RND*26
90 IF ABS(i1-i2)<5 THEN 70
100 INK 0,i1:INK 1,i2
110 s=RND*5+3
120 ORIGIN 320,-100
130 FOR x= -1000 TO 0 STEP s
140 MOVE 0,0
150 DRAW x,300:DRAW 0,600
160 MOVE 0,0
170 DRAW -x,300: DRAW 0,600
180 a$=INKEY$
190 IF a$<>"" THEN 30:REM INTERRUPT
 THE LOOP BY PRESSING ANY KEY
200 NEXT' x
210 GOTO 10
```

This and the preceding program illustrate simple mathematical concepts in a colourful and very visual way. Both are basically doing some sums on randomly generated 'seed' numbers to ensure that each pattern is different in some way, and displaying the results as random lines.

Your CPC464 is excellent electronic graph paper, and one of the most classic geometrical patterns is a sine wave:

```
10 REM DRAW SINE WAVE
20 MODE 2
30 INK 1,21
40 INK 0,0
50 CLS
60 DEG
70 ORIGIN 0,200
80 FOR n=0 TO 720
90 y=SIN(n)
100 PLOT n*640/720,198*y,1
110 NEXT
```

The P LOT statement in line 100 is the part of the program that draws the line. It produces one dot (pixel) on the screen for each calculation it makes in the F0R N E X T loop (lines 80-l 10) - and the result is displayed on your screen.

The CPC464 has many simple and powerful commands - you can add to the effect of the above progam by simply adding:

```
15 BORDER 6,9
```

R U N again. The border is now alternating between the colour numbers 6 and 9. The flashing rate is set by the 'default' values. To make the program loop continuously until you press **[ESC]ape** (twice to break out of the program, once to suspend operation), add:

```
120 GOT0 50
```

See that the flashing border did not stop when the program did - this is because the border is controlled indepedently of the rest of the program. To stop the flashing and set the border to bright blue, press **[ESC]** twice, then change line 15 to

```
15 BORDER 2
```

R U N the program, and the flashing stops.

To change the colour of the curve and the background, you must change the colour of the I N K in lines **30** and 40. When you L I S T the program, it should then look like:

```
10 REM DRAW SINE WAVE
15 BORDER 2
20 MODE 2
30 INK 1,2
40 INK 0,20
50 CLS
60 DEG
70 ORIGIN 0,200
80 FOR n=0 TO 720
90 y=SIN(n)
100 PLOT n*640/720,198*y,1
110 NEXT
120 GOT0 50
```

The number 1 at the end of the P L 0 T statement in line 100 tells the computer to plot the curve in the colour specified by the I N K 1 command in line 30. Check the definition of the P L 0 T statement in the keyword listing of Chapter 8 and you will see exactly how the various parts of this statement work.

If you look closely at the curve being plotted on the screen, you will see that it is not a continuous line, but is broken in many fine segments. The smallest individual segment is an example of a 'pixel' described earlier.

5.2.5 The graphics cursor and drawing lines

You have now tried some of the ways you can translate programs into graphic displays - and several of the program commands and concepts have been given a chance to perform. When drawing lines at the screen, there are some important considerations to watch out for to avoid confusion.

The first point to watch for is the current state of the program memory. The computer remembers the current colour settings even after a NEW instruction to clear the program memory. To reset everything to the starting point, you should use the simultaneous **[CTRL][SHIFT][ESCAPE]** sequence to get back to the switch-on conditon. (S ***A V E*** anything you need to before doing this!)

You can prove this by simply typing:

NEW:CLS

. . ..after you have broken out of the previous program. Now type:.

DRAW 100,100

The ***DRAW*** instruction draws a straight line from the last location of the GRAPHICS cursor to the **x,y** coordinate point specified (100,100). The GRAPHICS CURSOR is an invisible concept that indicates the point at which the next graphics operation will occur.

To find out where it is, you must use the functions X P 0 S and Y P 0 S. Type in:

PRINT XPOS

The answer is

100

(which is the same for Y P 0 S at this point)

Note that if the text goes down to the bottom of the screen and causes the display to be moved up ('scrolled' up), the graphics display will move up as well but the graphics cursor position remains the same as before. Try it - hold down the cursor down key [↓] until the screen clears away at the top, then ask for X P 0 S and Y P 0 S again. The graphics cursor value is still there in the memory.

To specify a colour for a line drawn, add the instruction at the end of the ***D*** RAW command (see the description of the P LOT command after the program on the previous page - it works the same way).

You must first have specified the I N K - and remember that you can only use the number of I N Ks and colours that are permissible in the screen mode you are using. To see this, type in:

```
10 MODE 1
20 INK 0,10
30 ORIGIN 0,0
40 INK 1,26
50 INK 2,0
60 DRAW 320,400,1
70 DRAW 640,0,2
```

Here's an example of a program that uses all the items mentioned so far, and introduces a couple more useful concepts. See how the first line (10) sets up the colour and ink conditions to make sure that whatever was in the memory of the CPC464 at the time is reset to produce the expected results:

```
10 INK 0,0:INK 1,26:INK 2,6:INK 3,18: BORDER 0
20 REM this programs draws patterns
30 mode 1:DEG
40 PRINT "3,4 or 6 sided pattern ? ";
50 LINE INPUT p$
60 IF p$="3" THEN sa=120:GOTO 100
70 IF p$="4" THEN sa=135:GOTO 100
80 IF p$="6" THEN sa=150:GOTO 100
90 GOTO 50
100 PRINT:PRINT"Calculating";
105 IF p$="3" THEN ORIGIN 0,-50,0,640,0,400
 ELSE ORIGIN 0,0,0,640,0,400
110 DIM cx(5),cy(5),r(5),lc(5)
120 DIM np(5)
130 DIM px%(5,81),py%(5,81)
140 st=1
150 cx(1)=320:cy(1)=200:r(1)=80
160 FOR st=1 TO 4
170 r(st+1)=r(st)/2
180 NEXT st
190 FOR st=1 TO 5
200  lc(st)=0:np(st)=0
210  np(st)=np(st)+1
220  px%(st,np(st))=r(st)*SIN(lc(st))
230  py%(st,np(st))=r(st)*COS(lc(st))
240  lc(st)=lc(st)+360/r(st)
245 IF (lc(st) MOD 60)=0 THEN PRINT ".";
250  IF lc(st)< 360 THEN 210
252  px%(st,np(st)+1)=px%(st,1)
254  py%(st,np(st)+1)=py%(st,1)
260 NEXT st
265 CLS:ink 1,2
270 st=1
280 GOSUB 340
290 LOCATE 1,1
300 EVERY 25,1 GOSUB 510
310 EVERY 15,2 GOSUB 550
320 EVERY 5,3GOSUB 590
530 GOTO 330
340 REM draw circle plus 3,4 or 6 others
 around it
350 cx%=cx(st):cy%=cy(st):lc(st)=0
360 FOR x%=1 TO np(st)
370  MOVE cx%,cy%
```

/more.. .....

```
380 DRAW  cx%+px%(st,x%),cy%+py%(st,x%),
1+(st MOD 3)
390  DRAW  cx%+px%(st,x%+1),cy%+py%(st,
x%+1),1+(st MOD 3)
400  NEXT x%
410  IF st=5 THEN RETURN
420  lc(st)=0
430  cx(st+1)=cx(st)+1.5*r(st)*SIN(sa+lc(st))
440  cy(st+1)=cy(st)+1.5*r(st)*COS(sa+lc(st))
450  st=st+1
460  GOSUB 340
470   st=st-1
480 lc(st)=lc(st)+2*sa
490 IF (lc(st) MOD 360)<>0  THEN  430
500  RETURN
510  ik(1)=1+RND*25
520  IF ik(1)=ik(2) OR ik(1)=ik(3) THEN 510
530  INK 1,ik(1)
540  RETURN
550  ik(2)=1+RND*25
560  IF ik(2)=ik(1) OR ik(2)=ik(3) THEN 550
570  INK 2,ik(2)
580  RETURN
590  ik(3)=1+RND*25
600  IF ik(3)=ik(1) OR ik(3)=ik(2) THEN 590
610  INK 3,ik(3)
620  RETURN
```

When you R U N this program, it will ask you a question (line ***40)*** - answer 3 for the speediest results. The program will then display the message Calcul***a t i n g***, and display a dot . every few seconds (line ***245)*** to indicate that it is still 'thinking' to itself and confirm that the program is still running.

The subroutines called by lines 300-320 flash the different **coloured** inks at the rates determined by the E V E R Y command. If you want to slow down the flashing, E **DI** T lines 300-320 to read:

```
300 EVERY  250,1 GOSUB 510
310  EVERY 150,2 GOSUB 550
320  EVERY 50,3 GOSUB 590
```

To see what you have done, look up the E V E R Y command in Chapter 8, it's one of the most useful features in AMSTRAD BASIC. One interesting effect of the E V E R Y command is the way it stacks up requests to do something if the program is interrupted by pressing the **[ESC]** key - only ONCE.

Pause the operation of the program for a few seconds by doing this, then restart by pressing 'any key'. The display will flash frantically as the 'queued' timing instructions rush through to catch up. There's only a finite amount of space in the queue, so after a while, the new ***EV E R Y*** command gets discarded until space is made by allowing those in the queue to work their way through.

5.3 Windows

The user can select up to eight text windows into which characters are written, and also a graphics window into which plotting may be performed. Windows are reset to defaults when the screen mode is set. See the keyword description in Chapter 8.

NB: If the text window is equivalent to the entire screen (default), then rapid rolling is achieved by hardware. If the text window is less than the available screen, then rolling is achieved by software, which is correspondingly slower.

The WINDOW command specifies the left/right/top/bottom character cells of the specified screen stream - windows may overlap one another, and provide a rapid means of drawing filled boxes. Before starting to explore them, type:

```
KEY 139, "mode 2:paper 0:ink 1,0:ink 0,9:
          list"+chr$(13)
```

This sets the smaller **[ENTER]** key to clear and restore the text to visible colours should you get lost in some invisible combinations of P E N and PA P E R. The following program draws a series of windows across the screen, and illustrates two major points:

```
5 MODE 0
10 FOR n=0 TO 7
20 WINDOW #n,n+1,n+6,n+1,n+6
30 PAPER #n,n+4
40 CLS #n
50 FOR c=1 TO 200:NEXT
60 NEXT
```

The first point is that each new screen overwrites the one before, and the second is to emphasise that the messages appear in stream #0 at all times (unless redirected). Before doing anything else, type:

```
LIST
```

And the program will be squeezed through stream 0. Try:

```
LIST #5
```

Then:

```
CLS #6
```

..illustrating the point that the most recently addressed screen stream will overwrite all else - and that the system message **Ready** appears in stream 0, even when the listing was sent to stream 5.

Using the W I N D 0 W SW A P command, add in line 55:

```
55 IF n=3 THEN WINDOW SWAP 7,0
```

You may imagine that this will direct the **Ready** message at the end of the program execution to stream 7. Run it and see. By developing this simple program, you will get an appreciation of the way W I N D 0 W s operate and interact.