Chapter 6

GRAPHICS

6.1 INTRODUCTION

No book of this sort would be complete without a discussion of the superb graphics capabilities of the BBC microcomputer. This chapter presents the simpler aspects of using graphics. It is not possible to cover all the advanced features here, such as use of the more complicated PLOT commands, plotting over or under existing screen displays and special facilities for animation.

Nor shall we attempt to discuss graphics in Mode 7, which is entirely different from that in all the other modes, and mainly suitable for logos and the like.

6.1.1 The graphics modes

Setting aside Mode 7, the remaining modes can be subdivided first into graphics and non-graphics modes.

Mode 3 and Mode 6 give a two-colour, 25 line, text-only display of respectively 80 and 40 characters per line. They cannot be used for graphics. Graphics commands issued in these modes will have no effect, though they will not give rise to any error message that would not have arisen in Modes 0, 1, 2, 4 or 5. The reason why these modes cannot allow for graphics is that the displayed lines of text are separated by a small gap. This can clearly be seen if you use commands such as

```
MODE 6
VDU 19,0,4,0,0,0
```

Some users find this type of display more pleasing than the normal one, with a coloured rather than black background, and the lines separated by black gaps. Redefined characters can be used with these modes, however, and this is one reason why they use nearly as much memory as the graphics modes (the difference arises from the saving due to the smaller number of lines of pixels stored).

The remaining modes: 0, 1, 2, 4 and 5, all nominally give a 32 line text

display with no gaps between the lines. They vary both in the number of characters per line and the number of colours available. Modes 0 to 2 require 20 Kbytes of memory and are only available on Model B computers, while Modes 4 and 5 take up 10 Kbytes, and can thus be used on Model A computers also, although in this case they leave rather little memory available for the BASIC program.

Table 6.1 summarizes the properties of Modes 0 to 6.

Table 6.1 Properties of the display modes

Mode	*Characters per line*	*Colours available*	*Memory used*	*Graphics*	*Resolution*
0	80	2	20K	YES	640×256
1	40	4	20K	YES	320×256
2	20	16	20K	YES	160×256
3	80	2	16K	NO	
4	40	2	10K	YES	320×256
5	20	4	10K	YES	160×256
6	40	2	8K	NO	

Note that the colours available in a mode include black and white; thus a 2 colour mode is normally white on a black background, with no other colours available.

6.1.2 Pixels

The fundamental principle of graphics display is that the screen is divided up into a large number of 'picture elements', or *pixels*. Each pixel can be lit up in any of the available colours. Thus, like a newspaper picture, a graphics display is made up of small dots. The size of these dots determines the *resolution* of the picture. Tiny dots give a high resolution picture containing fine detail, whereas large dots give a much coarser picture in which fine detail is lost. It is common to express the resolution by the horizontal and vertical size of the display, expressed in pixels. For example, on the BBC computer, the resolution of Mode 1 is 320×256.

Knowing the resolution of the computer we can work out the number of lines of text that are possible, in the following way. Text characters all occupy character 'cells' which are made up of 8 rows of 8 pixels, with the appropriate pixels being lit to generate the particular character being displayed (one row and column are always blank in order to leave a space around characters, except for lower case letters with descenders). The way in which each

character is formed can be seen in the Appendix of the *User Guide*. All graphics display modes are 256 pixels high which allows 32 lines of 8 pixel high text. The horizontal resolution varies from 160 to 640 pixels, depending on mode. The 40 character modes give a nearly square display, where the height and width of each pixel is nearly equal. In other modes, the non-square nature will result in horizontal and vertical plotted lines having different widths.

In order to plot a point on the screen the nearest pixel must be lit up or 'plotted'. Obviously, any pixel could in principle be specified by its horizontal and vertical position on the screen. However, the same position on the screen would then need to be specified differently in the different modes, because the size of the pixels is different. On the BBC computer, therefore, a pixel is specified by means of screen coordinates which are independent of the mode.

The display area is always 1280 units wide by 1024 units high, and positions on the screen are specified in terms of these units. The origin of the display is at the bottom left of the screen, just as with a normal graph, and this position has coordinates 0,0. The possible positions on the screen are thus 0 to 1279 horizontally, and 0 to 1023 vertically, as shown in Figure 6. 1.

It will be apparent that a pixel has dimensions of several screen units, and this will vary in the different modes. Figure 6.2 shows the size of a pixel in terms of screen units for the different modes. When a point or line is being plotted, a pixel is lit up if the point or line passes through any of the screen coordinates within it.

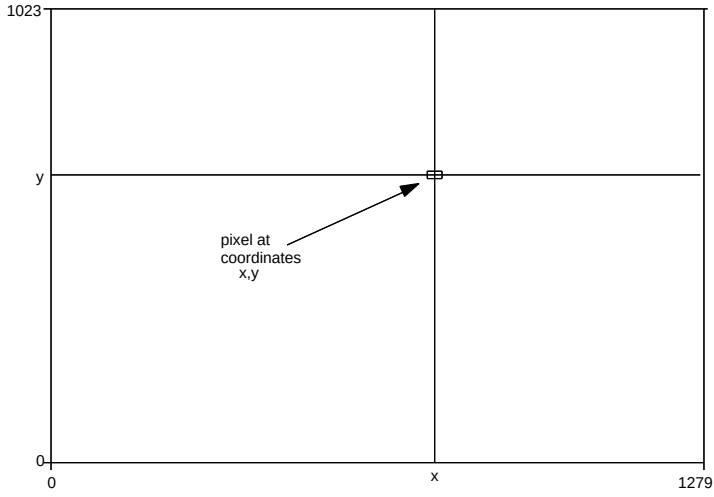

Figure 6.1 Coordinate system for the graphics modes.

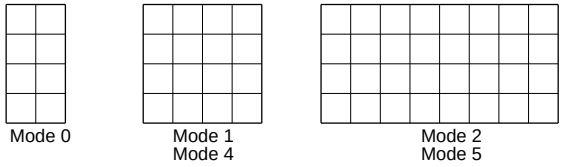

Figure 6.2 The size of a pixel in screen units for the different graphics modes.

The biggest problem with the limited resolution available on a microcomputer is not, as might be expected, the limitation on fine detail, although this is a consideration, but the unpleasant effect that can occur with straight lines. This particularly affects lines that are nearly horizontal or nearly vertical. With a slightly sloping line, the ideal line will pass successively through several pixels in the same row, after which it will cross the boundary between one row of pixels and the adjacent row, and abruptly the screen line will jump to that row for several more pixels. The result is that the sloping line becomes a jagged series of horizontal or vertical lines.

The effect is not confined to microcomputers, and you can observe the effect, for example, on television clocks. It is most easily seen on the second hand as it reaches one second to or one second past the minute.

Example 6.1

The following commands will demonstrate the resolution limitation very clearly.

```
MODE 1
MOVE 0,496: DRAW 1279,528
MOVE 0,0: DRAW 1279,1023
MOVE 628,1023: DRAW 656,0
MOVE 0,1023: DRAW 1279,0
```

Try these commands in other graphics modes as well, and notice the non-square effect showing up in the unequal line widths in Modes 0, 2 and 5.

6.1.3 Colour

The BBC computer provides an excellent range of facilities for colour displays and colour pictures of very high quality, particularly when the RGB output is used with an RGB colour monitor, rather than the UHF output and a television set. Text and graphics colours are set by separate commands, and both foreground and background colours may be controlled.

The number of colours available depends on the mode, and is shown in the table earlier. For text, the text colour can be changed by the command

```
COLOUR n
```

where n is the number for the desired colour. If n is greater than the range of colours available for the mode in use but less than 128, it is treated modulo that number. For example, in a four colour mode COLOUR 6 would actually select colour number 2, and in a two colour mode COLOUR 4 would select colour number 0.

For graphics the same rules apply, but the command is

```
GCOL 0,n
```

The first number specified can have a range of values in order to give rise to special effects, but in most cases it will be zero.

The colour numbers for the different modes are given in Table 6.2. In all cases the default colours for both text and graphics, set when the mode selection statement is executed, are white foreground and black background.

Table 6.2 Colours available in different modes.

Mode	Colour	Colour number	Background number
0,3,4,6	black	0	128
	white	1	129
1,5	black	0	128
	red	1	129
	yellow	2	130
	white	3	131
2	black	0	128
	red	1	129
	green	2	130
	yellow	3	131
	blue	4	132
	magenta	5	133
	cyan	6	134
	white	7	135
	flashing	8-15	136-143

The last eight ‘colours’ in Mode 2 are actually flashing combinations between pairs of complementary colours: colour n+8 flashes between n and 7-n. (The reason why only eight separate colours are provided is because these are the only eight colours that can be generated by various combinations of the three primary colours red, green and blue in equal intensity.)

Background colours may also be changed, and this is done by specifying the colour number plus 128 in the above commands. Thus to set a red text or graphics background in Mode 1 or 5, the command would be

```
COLOUR 129
```
or
```
GCOL 0,129
```

Note that this will not turn the whole screen red. It will simply provide a red background to any subsequent characters (including spaces) that are printed. The whole screen can be turned red by using CLS, which clears the screen to the current background colour. Similarly the graphics screen can be cleared to the current graphics background colour by CLG. Note that unless a text or graphics window has been set, these two ways of changing the background colour of the whole screen give the same result.

Exercise 6.1

Experiment with the COLOUR commands in direct mode, for different graphics modes, including changing the background. Note that if you make the foreground and background colours the same, subsequent text will become invisible.

6.2 PLOTTING LINES AND CURVES

6.2.1 The MOVE and DRAW commands

A great deal can be achieved in the way of graphics displays just by use of the two simple commands MOVE and DRAW, along with the GCOL command to set the graphics colour. The two commands can be envisaged as moving a pen across the screen to the position specified, starting from the position where it last came to rest. After a graphics mode is selected the pen starts from screen position 0,0. (Note that in OS 1.2 a CLG command does not restore the pen to 0,0, contrary to what is stated in the BBC User Guide.) The difference between the commands MOVE and DRAW is that with DRAW the pen leaves a trail of 'ink' in the foreground colour as it moves, whereas with MOVE it is 'lifted' from the screen and moves invisibly. The commands are followed by the horizontal (x) and vertical (y) screen coordinates to be moved to. Thus

```
MOVE 0,512
DRAW 1279,512
```

will draw a horizontal line through the middle of the screen.

Although the size of the screen is 1280 units by 1024 units, it is possible to plot outside this area, though obviously nothing will be seen. The plotting does exist, however, at least in the sense that a line drawn from somewhere off the screen to a position on the screen will appear to come from that hypothetical off-screen position. The overall limits on the x and y parameters to MOVE and DRAW are -32768 to 32767. (Larger values will not cause an error. It is a general feature of graphics commands, as well as other VDU commands, that they do not give rise to execution errors — they simply produce incorrect effects. For instance, numbers between 32768 and 65535 will he. treated as neeative numbers.

Exercise 6.2

Experiment with these commands in direct mode. (Note that the text and graphics may superimpose on the screen, and if the text screen scrolls, it will carry the graphics display with it.) Try commands such as

```
MODE 1
GCOL 0,1
MOVE 0,512: DRAW 1279,512
MOVE 640,0: DRAW 640,1023
GCOL 0,2
MOVE 0,0: DRAW 1279,1023
MOVE 0,1023: DRAW 1279,0
```

Example 6.2

On some computers, there are commands to draw a line between a pair of specified points with a single instruction. Graphics is an ideal area for the application of procedures, and a procedure to draw a line from Xl, Yl to X2, Y2 in colour COL could take the form

```
1000 DEF PR0C_line(X1,Y1,X2,Y2,COL)
1010 GCOL 0,COL
1020 MOVE X1,Y1
1030 DRAW X2,Y2
1040 ENDPROC
```

6.2.2 Plotting with straight fines

In view of the great variety of purposes for which simple graphics involving the drawing of straight lines and curves can be used, it is difficult to generalize how to tackle it, since different approaches will be needed for different problems. The most obvious division between uses is whether straight lines or curves are being plotted.

The most straightforward use of graphics is for plotting straight lines, since the DRAW command is directly appropriate for drawing any straight line; i.e. the software is already implemented. As an example, consider how to plot a monthly sales graph.

Example 6.3

```
 5 DIM SALES(12)
10 REPEAT
20 MODE 1
30 PRINT "Input the monthly sales figures"
40 PRINT "one per line, for January"
50 PRINT "to December."
60 MAX=0
70 FOR J=1 TO 12
```

```
 80 INPUT SALES(J)
 90 IF SALES(J)>MAX THEN MAX=SALES(J)
100 REM MAX IS THE MAXIMUM SALES FIGURE
110 NEXT J
120 CLS
130 MULT=1: POW10=0
140 IF MAX<50 THEN GOTO 200
150 REPEAT
160 MAX=MAX/10
170 MULT=MULT*10: POW10=POW10+1
180 REM MULT IS MULTIPLE OF 10 SCALE FACTOR, I.E. 10^POW10
190 UNTIL MAX<50
200 SCALE=1-(MAX>10)-3*(MAX>20)
210 REM SCALE*MULT IS SCALE FACTOR PER DIVISION
220 GCOL 0,3
230 MOVE 128,148
240 FOR J=148 TO 948 STEP 80
250 MOVE 128,J: DRAW 1184,J: REM PLOT GRID
260 NEXT J
270 MOVE 128,144: DRAW 1184,144: REM PLOT ORIGIN DOUBLE
    THICKNESS
280 MOVE 124,144: DRAW 124,948
290 MOVE 128,148
300 FOR J=128 TO 1184 STEP 96
310 MOVE J,148: DRAW J,948
320 NEXT J
330 FOR J=0 TO 10 STEP 2
340 PRINT TAB(1,27-2.5*J);SPC(2-LEN(STR$(SCALE*J))); SCALE*J;
350 NEXT J
360 PRINT TAB(0,0);"x10^";POW10
370 PRINT TAB(3,28);"Jn Fb Mr Ap My Jn Jl Au Sp Oc Nv Dc"
380 PRINT
390 GCOL 0,1: PROC_pltpt(1,FN_ycoord(1,SCALE,MULT))
400 FOR J=2 TO 12
410 DRAW 32+J*96,FN_ycoord(J,SCALE,MULT)
420 PROC_pltpt(J,FN_ycoord(J,SCALE,MULT))
430 NEXT J
440 PRINT "Press SPACE for another graph"
450 PRINT "or any other key to end";
460 UNTIL GET$<>" "
470 END
2000 DEF PROC_pltpt(X,Y)
2010 LOCAL Y1
2020 FOR Y1=-8 TO 8
2030 MOVE 24+X*96,Y+Y1: DRAW 40+X*96,Y+Y1
2040 NEXT Y1
2050 MOVE 32+X*96,Y: REM MOVE TO CENTRE OF POINT READY TO DRAW
     LINE
```

```
2060 ENDPROC
2070 DEF FN_ycoord(J,SCALE,MULT)
2080 =148+80*SALES(J)/(SCALE*MULT)
```

This program plots a suitable sales graph for a set of monthly figures typed in by the user.

From the point of view of the graphics commands required, the only problem in the above program is in scaling the graph, since the screen is fixed at 1024 units high by 1280 units wide. If we are plotting the graph for one year, then a convenient horizontal scale would be 96 units per month (which corresponds to exactly three characters in Mode 1 or 4), leaving 128 units blank on the sides of the graph.

The choice of the vertical scale is not so simple. If the chart is to be split up into 10 divisions, it would be convenient to use 80 units per division, with 224 units (56 pixels or 7 lines of characters) shared between the spaces at the top and bottom.

After this choice has been made it will be necessary to find the largest monthly sales figure, and use this to determine the vertical scale for each of the ten divisions. If we want each division to represent a round quantity (normally 1, 2 or 5 times some power of 10) it is also necessary to make the scale larger still to accommodate the next higher round value.

Line 90 determines MAX, the maximum value to be plotted, which is used to calculate the vertical scaling. Lines 150 to 190 ascertain the power of ten needed in the scaling factor, the whole factor being stored in the variable MULT and the power in POW10. Line 200 determines whether each division will represent 1, 2 or 5 times MULT, using the fact that when evaluating a Boolean test, TRUE is -1 and FALSE is zero.

Lines 340, 360 and 370 print the scale and months on the graph, using the TAB command as described in Chapter 11.

Finally and most importantly, lines 250 and 310 respectively draw the horizontal and vertical grid lines, PROC_pltpt at line 2000 plots each sales ‘point’ and line 410 draws the lines joining the points, calling FN ycoord at line 3000 to calculate the screen y-coordinate from the sales values SALES(J).

6.2.3 Plotting curves

For the most accurate representation, curves should be built up by plotting pixel by pixel, but this is generally too slow to be practical. Curves are almost always acceptable if plotted as short straight lines, but trial and error may be necessary to determine how long the straight lines can be.

The only curves which can be drawn simply from within a program are those with some definite mathematical formula. The simplest way to draw a curve is for functions that have the form

y=f(x)

which allows the curve to be built up by stepping along the x-axis, calculating y from the function at every step.

A simple example of plotting a sine wave is shown in Example 6.4. Try different values of the order n of the curve (which is of the form sin(nx)) and the step length (step length here means the horizontal length of each line *in pixels*) to see what step length gives an acceptable compromise between resolution of the curve and speed of execution for different orders. Note that some interesting effects can be obtained with the program below if very large values are specified for n.

Example 6.4

```
 10 MODE 1
 15 REPEAT
 20 PRINT TAB(0,26); "What order of sine curve would you"
 30 INPUT "like? (1-10) "N
 40 PRINT "What step length would you like?"
 50 INPUT "(1-50) "STP
 60 GCOL 0,3
 70 X0=640: Y0=608
 80 MOVE 0,Y0: DRAW 1279,Y0
 90 MOVE X0,192: DRAW X0,1023
100 REM PLOT AXES
110 XMAX=INT(636/STP)*STP
120 YMAX=400
130 MOVE X0-XMAX,Y0+YMAX*SIN(-N*PI)
140 FOR J=-XMAX+4*STP TO XMAX STEP 4*STP
150 X=PI*J/XMAX: REM SCALE ANGLES
160 Y=YMAX*SIN(N*X)
170 DRAW J+X0,Y+Y0
180 NEXT J
190 PRINT "Press SPACE to plot another curve"
200 PRINT "or any other key to end ";
210 UNTIL GET$<>" "
```

The variable STP specifies the step length, and is multiplied by 4 (since the program is working in Mode 1) so that it directly represents the number of pixels. Remember in your own applications that this step length always needs to be an integral number of pixels, otherwise you may get uneven results in your curves.

For sine curves scaling is simple since sin(nx) has a maximum value of 1, so y values are simply multiplied by YMAX (400). Similarly, the maximum value of angle wanted is π so the x values can be scaled fairly simply. The X0 and Y0 added to the x and y values shift the origin to the centre of the top section of the screen (leaving room for a few lines of text at the bottom). Note that there is an alternative way of moving the origin to the centre of the screen, which is discussed in Section 6.3.2.

The next example plots a parabola, with a variable position for the focus, which means that the function must be scaled horizontally so that the curve

makes the maximum use of the screen area. This is done by the scale factor SCALE, calculated in line 140 and used to scale X*X in line 170.

Example 6.5

```
 10 MODE 1
 15 REPEAT
 20 PRINT TAB(0,26)"What focal length would you like?"
 30 INPUT "(12-400) "A
 40 PRINT "What step length would you like?"
 50 INPUT "(1-50) "STP
 60 GCOL 0,3
 70 X0=640: Y0=192
 80 MOVE X0-12,Y0+2*A: DRAW X0+12,Y0+2*A
 90 MOVE X0,Y0+2*A-12: DRAW X0,Y0+2*A+12
100 REM PLOT FOCUS
110 XMAX=INT(636/STP)*STP
120 MOVE X0-XMAX,Y0: DRAW X0+XMAX,Y0
130 REM PLOT BASE LINE
140 SCALE=(800-A)*4*A/(XMAX*XMAX)
150 YOLD=A
160 FOR X=4*STP TO XMAX STEP 4*STP
170 Y=A+SCALE*X*X/(4*A)
180 REM FORMULA FOR A PARABOLA IS X*X - 4A(Y-A)
190 MOVE X-4*STP+X0,YOLD+Y0: DRAW X+X0,Y+Y0
200 MOVE -X+4*STP+X0,YOLD+Y0: DRAW -X+X0,Y+Y0
210 YOLD=Y
220 NEXT X
230 PRINT "Press SPACE to plot another curve"
240 PRINT "or any other key to end ";
250 UNTIL GET$<>" "
```

Plotting a curve as a series of straight lines for equal steps of X is not always the best method. One occasion when this is not the optimum strategy is when the curve has a vertical section. Compare the next two programs, both of which plot a circle of variable radius. The first uses equal steps of X, whereas the second program plots around the circle in equal steps of angle, which is much more satisfactory in the case of a geometric form such as a circle. The circle drawn on the screen may in practice be slightly elliptical due to variation in the height and width gain settings on your TV or monitor.

Example 6.6

```
10 MODE 1
20 INPUT "Radius of circle? (4-400) "R
30 STP=INT(R/20): IF STP<1 THEN STP=1
40 X0=640: Y0=512
50 MOVE X0-R,Y0
```

```
 60 FOR X=-R TO R STEP STP
 70 Y=SQR(R*R-X*X)
 80 DRAW X+X0,Y+Y0
 90 NEXT X
100 REM FORMULA FOR A CIRCLE IS X*X+Y*Y=R*R
110 FOR X=R TO -R STEP -STP
120 Y=SQR(R*R-X*X)
130 DRAW X+X0,-Y+Y0
140 NEXT X
150 DRAW X0-R,Y0
```

Here the number of steps has automatically been set to about 40, which is a reasonable compromise between speed of drawing and smoothness of the circle drawn.

Example 6.7

```
10 MODE 1
20 INPUT "Radius of circle? (4-400) "R
30 X0=640: Y0=512
40 MOVE X0-R,Y0
50 FOR ANGLE=0 TO 360 STEP 5
60 Y=R*SIN(RAD(ANGLE))
70 X=-R*COS(RAD(ANGLE))
80 DRAW X+X0,Y+Y0
90 NEXT ANGLE
```

This time the step length is 5 degrees, giving 36 steps in each semi-circle.

Exercise 6.3

Alter the program from Example 6.7 to produce an ellipse by the following changes

```
15 INPUT "Semi-major axis of ellipse? (4-636)"A
20 INPUT "Semi-minor axis of ellipse? (4-400)"B
40 MOVE X0-A,Y0
55 R=1/SQR(SIN(RAD(ANGLE))^2/(B*B)+COS(RAD(ANGLE))^2/(A*A))
```

Exercise 6.4

Finally, the original circle program of Example 6.7 can also be modified to produce a spiral, as follows

```
20 R=0
50 FOR ANGLE=0 TO 3600 STEP 5
55 R=ANGLE/10
```

6.3 THE PLOT AND VDU GRAPHICS COMMANDS

6.3.1 The PLOT command

Although a great deal can be achieved with MOVE and DRAW, these are only the two simplest of a large range of plotting commands. The full range is provided by the command PLOT, which has the form

```
PLOT K,X,Y
```

X and Y are the screen coordinates just as with DRAW, but K can take a large number of possible values, each giving a different form of plotting action. The factor K itself can be subdivided into three components, L + M + N.

L chooses the pen colour, while M determines whether the pen movement is relative or absolute. The choice of colours is: L=0, none (i.e. pen-up movement); L=1, the graphics foreground colour; L=3, the graphics background colour (this is not exactly the same as pen up movement because it will effectively eradicate any line already present by merging it with the background); and L=2, the logical inverse colour. This last colour we shall not consider further.

M=0 gives relative plotting, while M=4 gives absolute plotting. The latter means that the X and Y values specify the actual x- and y-coordinates on the screen, whereas when X and Y are relative, the value of X and Y specify the distance to be moved from the present pen position. Care must be taken when using relative plotting, as each relative movement can contribute a cumulative error to the position of the pen.

The effect of different combinations of L and M are shown in Table 6.3.

Table 6.3 Plot codes for different plotting colours.

Plotting colour	*Relative movement (M=0)* L + M	*Absolute movement (M=4)* L + M
Move only	0	4
Current graphics foreground	1	5
Logical inverse	2	6
Current graphics background	3	7

It can be seen from this that the commands MOVE and DRAW are exactly equivalent to PLOT 4 and PLOT 5. They are given separate, alternative BASIC keywords which are easier to remember and use because of

their special importance, but the PLOT form can equally well be used if you prefer.

The last part of the K parameter to PLOT gives rise to further variants, each available in the 8 possible colourlmovement combinations set by L and M. The values of N and their effect are summarized in Table 6.4.

Table 6.4 Effects of different N values in PLOT.

N	*Action*
0	Simple drawing
8	Omitting the last point for inverting actions
16	Drawing a dotted line
24	Combination of 8 and 16
64	Plotting a single point only
80	Triangle filling

N=8 and 24 are beyond the scope of this chapter. N= 16 is quite useful, but note that the effect is to draw dots in specific screen columns, so the effect is slightly limited in usefulness, and will vary depending on the angle at which the line is drawn.

Using N=64 to plot a single point provides an alternative to the combination MOVE X,Y: DRAW X,Y. The simpler result is

```
PLOT 69,X,Y
```

The triangle filling option is very useful. It works on the point specified along with the previous two pen positions. This provides a way of very rapidly filling any desired area with solid colour.

Example 6.8

The most likely shape that you may want to fill is a rectangle. The following procedure will fill a rectangle in the current graphics foreground colour, between Xl and X2, and Yl and Y2.

```
1000 DEF PROC_rectangLe(X1,Y1,X2,Y2)
1010 MOVE X1,Y1
1020 MOVE X1,Y2
1030 PLOT 85,X2,Y1
1040 PLOT 85,X2,Y2
1050 ENDPROC
```

Example 6.9

By combining the triangle filling with the program to draw a circle, we can write a procedure to draw a solid circle of any radius and centre, in the current

graphics foreground colour. The procedure below will draw a solid circle of radius R, centred at X0,Y0.

```
1000 DEF PROC_circle(R,X0,Y0)
1010 LOCAL ANGLE
1020 MOVE X0+R,Y0
1030 FOR ANGLE=5 TO 360 STEP 5
1040 MOVE X0,Y0
1050 PLOT 85,X0+R*COS(RAD(ANGLE)),Y0+R*SIN(RAD(ANGLE))
1060 NEXT ANGLE
1070 ENDPROC
```

6.3.2 The VDU command

We have already met the VDU command as an alternative to CHR$, and indeed the command

```
VDU M,N,...
```

is exactly equivalent to

```
PRINT CHR$(M);CHR$(N);...
```

The most valuable use of the command is to control the screen display (or VDU display, hence the name of the command) by the issue of control characters. Indeed, in direct mode a third way of achieving the same result is to type CTRL characters. For instance, CTRL-V CTRL-A would select Mode 1, as would VDU 22,1 (or MODE 1). The reason for this redundancy is presumably simply to enable advanced users to achieve all their needs with the VDU command, while beginners have simpler, easy to remember commands for the functions that they will need. The VDU commands which have an alternativ form are listed in Table 6.5, and will not be considered further.

Table 6.5 VDU commands and their equivalents.

VDU number	*Control code*	*Equivalent*
9	CTRL-I	<TAB>
12	CTRL-L	CLS
13	CTRL-M	<RETURN>
16	CTRL-P	CLG
17	CTRL-Q	COLOUR
18	CTRL-R	GCOL
22	CTRL-V	MODE
25	CTRL-Y	PLOT
27	CTRL-[	<ESCAPE>
31	CTRL-_	PRINT TAB( );
127	N/A	<DELETE>

Non-graphics VDU commands

A number of the VDU commands are really nothing directly to do with graphics − they apply equally to text. Some of these are standard control codes, such as CTRL-G (VDU 7) to beep the speaker. These are contained in Table 6.6.

Table 6.6 Non-graphics VDU commands.

VDU number	*Control code*	*Effect*
0	CTRL-@	Does nothing
1	CTRL-A	Send next character to printer only
2	CTRL-B	Enable printer
3	CTRL-C	Disable printer
6	CTRL-F	Enable VDU drivers
7	CTRL-G	Beep the speaker (bell)
8	CTRL-H	Backspace cursor (but not delete)
10	CTRL-J	Move cursor down
11	CTRL-K	Move cursor up one line
14	CTRL-N	Switch on page mode
15	CTRL-O	Switch off page mode
21	CTRL-U	Disable VDU drivers or delete current line
30	CTRL-^	Home text cursor to top left

VDU 1, 2, 3, 6, 14 and 21 are dealth with in Appendix A. The remainder are self-explanatory.

The graphics-based VDU commands

The remaining VDU commands are used to control the graphics in various ways, and some have quite complex usage that needs to be described in detail. The commands are summarized in Table 6.7.

Table 6.7 Graphics-based VDU commands.

VDU number	*Extra bytes*	*Effect*
4	0	Write text at text cursor
5	0	Write text at graphics cursor
19	5	Define logical colour
20	0	Restore default logical colours
23	9	Define new character
24	8	Set graphics window
26	0	Restore default windows
28	4	Set text window
29	4	Set new graphics origin

To understand the effects of VDU 4 and VDU 5 we must first define the meaning of the text and graphics cursors. The text cursor is the position at

which text is normally printed on the screen, and it is governed by the TAB command in a PRINT statement, as well as being moved along every time a character is printed. Its position can be determined by POS and VPOS. The graphics cursor, on the other hand, is the position of the graphics pen as we have called it up to now. It is moved by the MOVE, DRAW and PLOT commands.

The two cursors are quite separate, but it is possible, by the command VDU 5, to switch text so that it is printed at the graphics cursor position instead of the text cursor (the text cursor remains at its old position, but no longer controls the text output). The text must now be positioned by the MOVE command. This has two main effects: firstly the position of text can be more precisely defined, to within one pixel in any direction instead of the nearest character cell. Secondly the screen will no longer scroll if output reaches the bottom of the screen; instead the output wraps round to the top of the screen. (If several screenfuls of text are printed, this causes overwriting and the text becomes unreadable.) Minor effects are that text colour and position are governed by the graphics colour setting and window (if any), the flashing cursor disappears, and characters do not obliterate the 8><8 square of pixels that they overwrite, so that they can be superimposed on each other, or on graphs. VDU 4 restores the effect of the text cursor.

Example 6.10

The following small program will print lines of text on the screen in one and a half line spacing (lines should be less than 40 characters long). Use ESCAPE to end the program. You can alter the line spacing by changing the factor 48 on line 60.

```
 10 ON ERROR GOTO 500
 20 MODE 1
 30 VDU 5
 40 J=0
 50 REPEAT
 60 MOVE 0,1023-48*J
 70 INPUT "" A$
 80 J=J+1
 90 IF J>21 THEN J=0
100 UNTIL FALSE
500 VDU 4
```

VDU 29 alters the position of the graphics origin. Very often, you will wish to centre your graphs, not on the default of the bottom left corner, but elsewhere such as at the centre of the screen. In our previous examples, this was achieved by defining X0 and Y0 and adding these into all MOVE and DRAW commands, but it can be more convenient to move the origin instead. VDU 29 is the first of these commands we have met that requires extra parameters. The 29 must be followed by two further numbers or variables, the x and y screen coordinates of the position where you want the new origin.

To move the graphics origin to the centre of the screen, for example, the command would be

```
VDU 29,640;512;
```

Successive parameters to the VDU command are normally separated by commas, but note that the x- and y-coordinates for VDU 29 must be followed by *semi-colons*. This tells the computer that they are two-byte numbers (since they may be greater than 255; you can alternatively break them up into low and high bytes, but this is very tedious). The semi-colon can be used at any time to indicate that the preceding parameter is a two-byte number, and this technique is sometimes used to concatenate redundant zeros, such as those at the end of VDU 19. Be careful, if you are doing this, that you do not omit the final semi-colon, as VDU commands do not give rise to error messages but can nevertheless result in peculiar effects. In this case, the next character input or issued would be seized as the last byte of the VDU command.

Exercise 6.5

Modify the programs in Examples 6.4 to 6.7 to use VDU 29 instead of X0 and Y0 for the graphics origin.

If VDU 29 is issued more than once, the newly specified coordinates are always relative to the initial origin at the bottom left-hand corner, not to the previously set origin.

6.4 SPECIAL GRAPHICS EFFECTS

6.4.1 Windows

It is possible to set windows for both the text and graphics displays. Initially either may be put anywhere on the screen, but if a window is set then the text or graphics, as appropriate, is restricted to the area of the screen inside the window − it is as if the defined area were a window through which you can look at the display behind. In the case of text, the text is restricted to the window area and the top left-hand corner of the text window becomes the new text origin, but graphics might well be plotted partly outside the area of the graphics window.

Text windows

Strictly speaking, text windows are unrelated to graphics, but it is convenient to treat them here along with the graphics window, especially as the two will often be used in conjunction.

A text window is defined by specifying the minimum and maximum character positions, both horizontally and vertically, in which text may appear. Figure 6.3 shows an example of this, in which only the top centre portion of the screen is used for a text window.

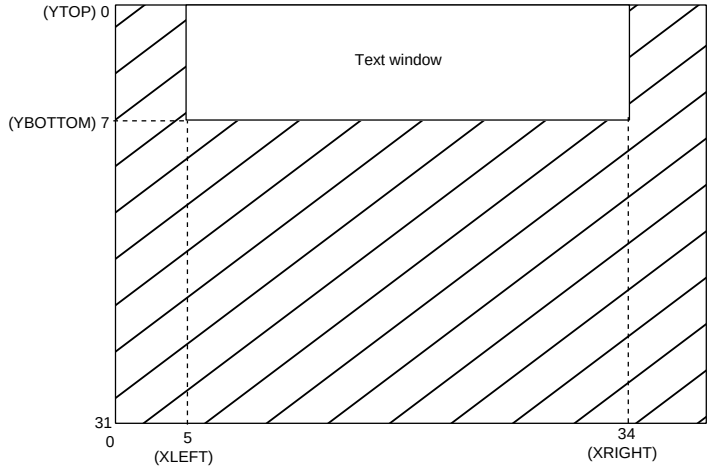

Figure 6.3 A text window as set up by the command VDU 28,5,7,34,0.

The command to set a text window is

```
VDU 28,XLEFT,YBOTTOM,XRIGHT,YTOP
```

You could think of this as specifying the bottom left and top right corners of the window. Note that for text Y is measured from the top, so YBOTTOM will be the *maximum* value of Y allowed. For the case shown in Figure 6.3, the command becomes

```
VDU 28,5,7,34,0
```

Note that the coordinates are the same as with the TAB command, and that the allowed ranges will vary with the mode in force. In this context you must remember that there are no error messages or error trapping for ‘faulty’ VDU statements − the command is simply ignored.

There are several reasons why you may wish to use a text window, quite apart from graphics windows. It would be the most convenient way of breaking program output into columns, particularly if the length of output may vary. A more important use is to keep a section of text permanently on the screen while other parts scroll. Once a text window is in force that does not encompass existing text, that text will not scroll, and so remains permanently on display. This technique is a favourite device in commercial programs such as word processors, where a top or bottom line showing status information remains permanently on display.

Any text window (or graphic window) may be cancelled with the command

```
VDU 26
```

The default full screen displays are also restored by a mode change (or, of course, a <BREAK>).

Example 6.11

The following program is a small demonstration which features the above points. It runs in Mode 6 to show that it is not restricted to graphics modes, but it could work equally well in any mode except Mode 2 or Mode 5, where the line length is too short.

```
10 MODE 6
20 PRINT "TEXT WINDOW DEMONSTRATION - THE ALPHABET"
30 VDU 28,5,9,34,2
40 FOR J=1 TO 26
50 FOR K=1 TO 240
60 PRINT CHR$(J+64);
70 NEXT K
80 PRINT
90 NEXT J
```

Exercise 6.6

Alter Example 6.11 so that it will work in Mode 2 or Mode 5.

Graphics windows

Graphics windows work in almost exactly the same way as text windows, except that, as with graphics displays on the full screen, the plotting range may run out of the window without ill effect, in which case only that part within the window is displayed.

The only difference is that the window will of course be in graphics screen units (with y measured from the bottom), which means that the x and y values will be greater than single bytes. They must therefore be two-byte parameters, and as before this is indicated by terminating them (even the last in the list) by semi-colons. The exact form of the command is therefore

```
VDU 24,XLEFT;YBOTTOM;XRIGHT;YTOP;
```

Note that this is the same convention as for the text window, specifying bottom left and top right corners of the window, but you must remember that this time YBOTTOM represents the smallest allowed value for Y.

The many conceivable uses include setting a window for a graph so that any lines that would otherwise spill outside the grid are suppressed. Another use could be to put a coloured border around a display.

A very useful application of windows is where text and graphics are to be used separately, but in the same display. Independent windows for the text and

graphics may be set so that they do not overlap, thus avoiding the possibility of one interfering with the other.

Example 6.12

The following program sets up a red graphics display area with a yellow border, and demonstrates the window effect further by attempting to draw a cross of full screen size.

```
10 MODE 1
20 GCOL 0,130
30 CLG
40 VDU 24,128;128;1151;895;
50 GCOL 0,129
60 CLG
70 MOVE 0,0: DRAW 1279,1023
80 MOVE 0,1023: DRAW 1279,0
```

Note that when no windows have been set, a background can be set to colour n either by

```
COLOUR 128+n: CLS
```

or

```
GCOL 0,128+n: CLG
```

but here CLG must be used because we wish to change the colour only within the graphics window. However, if text is subsequently printed onto the screen each character will have a black background unless COLOUR 128+n is also set, or VDU 5 is used to have text printed on a transparent background.

6.4.2 Changing colours

The limitation of most modes to 4 or 2 colours is not quite as restrictive as it may seem at first sight. Although the available colours in Mode 1, for instance, are by default black, red, yellow and white, it is possible to change these default colours. The command to do this is VDU 19. The change comes into effect immediately, changing the colour of pixels already displayed on the screen, and the command is frequently used to switch colours on and off to give an illusion of animation.

VDU 19 will change the colour that is associated with each of the permissible colour numbers in a particular mode. (In the *User Guide*, the colour number is called the *logical colour* and the colour associated with it the *actual colour*.)

The format of the command is

```
VDU 19,<colour number>,<actual colour>,0,0,0
```

The <actual colour> is specified by a number which is the same as the colour number sequence for Mode 2, as listed earlier. The trailing zeros are to allow for possible future expansion of the system.

Thus to change colour number 2 in Mode 1 from yellow to green the command would be

```
VDU 19,2,2,0,0,0
```

(or VDU 19,2,2;0; using the semi-colon to abbreviate the command).

The double occurrence of 2 in this particular example can be confusing, but a little thought will show that it is quite logical. If the colour number is outside the permitted range, then it is treated modulo the number of permissible colours, so in Mode 1 the same effect as above would be achieved by VDU 19,6,2,0,0,0. The command is not specifically a graphics command, and applies both to text and graphics colour.

Redefined colour numbers can all be set back to their default colours by

```
VDU 20
```

Exercise 6.7

Try changing the background colour of any of the earlier examples from black to blue, by using

```
VDU 19,0,4;0;
```

6.4.3 User-defined characters

One of the reasons why all the modes except Mode 7 take up so much memory is that each character is made up of an 8×8 pixel display. This means that it is possible to define new characters that will fit into an 8×8 display, or even redefine the existing character set. There is a command, VDU 23, that can be used to define characters.

Within the BBC microcomputer a character is specified by its ASCII code. The normal character set has ASCII codes from 32 to 127, with codes below 32 reserved for control codes (those that are normally issued by the VDU command). The default position for user-defined characters is that ASCII codes 224 to 255 can be defined by the user. The defined characters can then be printed either by PRINT CHR$(n) or VDU n. It is possible to change this default to a larger set, even to include redefining the standard characters, but extra memory has to be specially allocated for this purpose.

Defining a new character

A new character is defined by specifying which of the 64 pixels making up the 8×8 character cell are to be illuminated. Each row of the cell is defined by one number in the VDU string of parameters, made up in the following way: the right hand pixel counts as 1 if lit; the second as 2; the third as 4; and so on up

to 128 for the left hand pixel. In this way all possible combinations of lit and unlit pixels in the row can be represented by a single number between 0 and 255, which will just fit into a single byte of memory. (Readers familiar with bits and bytes will realize that each pixel is represented by one bit of the byte, the most significant bit representing the left-hand pixel)

The 8 rows of the cell give 8 numbers, which are issued with the VDU command as follows

```
VDU 23,<char no>,a,b,c,d,e,f,g,h
```

The <char no> is the ASCII code of the character being defined, and a to h are the 8 numbers defining the rows of pixels, a representing the top row, b the second, down to h for the bottom row.

Let us now see how to define a new character, the greek character pi. First it is necessary to plan out the character on a piece of squared paper. Since it is to be a normal character, there must be a line of unlit pixels along the bottom and left hand sides, following the same convention used for normal characters so that they are separated from one another. Figure 6.4 shows the shape to be defined.

The values for the rows are calculated as follows: the top row has just the right hand pixel lit, so the value is 1. The second row has pixels 2 to 6 lit, so the value is 62 (32+16+8+4+2). The next row has a value of 84 (64+16+4), the following two have the same value of 20 (16+4) and row f, 18 (16+2). The seventh row has a value of 33 (32+I) and we must not forget the blank row at the bottom, which simply has a value of 0. If this is to be the first special character available, 224, the required VDU command is

```
VDU 23,224,1,62,84,20,20,18,33,0
```

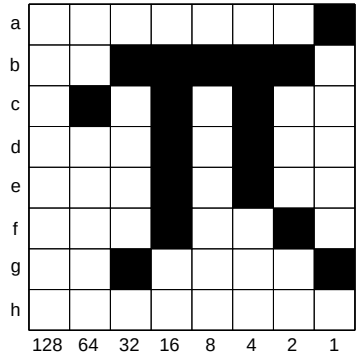

Figure 6.4 The shape needed for a user-defined character π

Exercise 6.8

Try out the above character by the direct mode commands

```
VDU 23,224,1,62,84,20,20,18,33,0
MODE 6
PRINT CHR$(224);" = ";PI
```

Note that the character, once defined, is preserved through a mode change [or even a BREAK).

Try to define another greek letter, such as phi.

Exercise 6.9

Try and define some symbols for the sales chart of Example 6.3, in place of PROC pltpt. Try shapes such as a square, a triangle and a circle, both solid and hollow.

User-defined characters are not restricted to single character cells. You can plan out a shape larger than 8x8 pixels, and spread it out over several character cells, which can be printed side by side or one above the other (using the TAB command). The sort of shapes that you might want are mathematical integral or summation signs, or small pictures. The latter will be plotted much faster in the form of user defined characters than by graphics commands. Each character can only be plotted in a single colour, but even this is not a major restriction, because the different colour components could be plotted as separate characters, superimposed with the aid of the VDU 5 command.

Making room for more user-defined characters

There is a *FX call, *FX 20,1, which allows the user to define as many characters as he wishes, or even redefine the normal character set. This uses up extra memory, at the start of the space normally used for programming. To allow for this, the normal value of PAGE for your system must be altered to accommodate the extra characters. (See Chapter 10 for a detailed discussion of PAGE, and general memory usage.) Provided that you have not already altered PAGE, the commands necessary to allow for extra characters are shown in Table 6.8 (note that the default character numbers are changed to 128-159 after *FX 20,1 − the call can also be used simply to make this change, if you wish).

Table 6.8 Making room for new characters.

ASCII character codes	*Command required*
128-159	`none`
160-191	`PAGE=PAGE+256`
192-223	`PAGE=PAGE+512`
224-255	`PAGE=PAGE+768`
32-63 (! to ?)	`PAGE=PAGE+1024`
64-95 (@ to _)	`PAGE=PAGE+1280`
96-126 (£ to ~)	`PAGE=PAGE+1536`

A full list of ASCII characters is given in Appendix H.

Switching the cursor off

There are two special purposes of the VDU 23 command. For the general user, the important one of these is to switch the flashing cursor off or on. Particularly when you have a graphics display, the flashing cursor can be an unwanted annoyance. It can be switched off with the command

```
VDU 23,1,0;0;0;0;
```

and switched on again by

```
VDU 23,1,1;0;0;0;
```

Note however that when text is linked to the graphics cursor by VDU 5, this also suppresses the flashing cursor, and if you are not wanting to print text at the text cursor position this is an easier way of removing it (VDU 4 brings it back, of course).

The second extra use of VDU 23, strictly for advanced users only, is to program the 6845 cathode ray tube controller chip (CRTC) which handles the screen display. We will mention only two possibilities here. First, the interlace which gives the screen display a slightly jittery effect, especially on some monitors, cannot be turned off in Mode 7 by the normal command *TV 0,1. It can be turned off, however, by

```
VDU 23,0,8,16,128;0;0;: VDU 23,0,9,9,0;0;0;
```

though you may think that the distorted character set that results is worse than the jitter.

The cursor can be altered in various ways by

```
VDU 23,0,10,n,0;0;0;
```

and

```
VDU 23,0,11,n,0;0;0;
```

In particular it can be turned off by

```
VDU 23,0,10,32,0;0;0;
```

and turned on again in its normal form by the VDU 23,1 commands. The above form will work even with the old 0.1 operating system.