1 Graphics Operations of the BBC Model B Microcomputer

Throughout the course of this book it will be assumed that the BASIC programming language of the BBC micro is reasonably familiar to the reader. In this first chapter, therefore, we shall be looking at some of the BASIC commands – those conncerned wholly or partly with graphics. The display capabilities of the micro will be explored by means of a series of example programs and simple exercises. In the chapters that follow we shall use this knowledge to develop a sound understanding, both practical and mathematical, of computer graphics.

Initially we shall consider the hardware and software facilities that are available for producing pictures. On the BBC computer there is a choice of eight different display MODEs numbered 0 to 7 (the last of which is the special TELETEXT mode which is discussed separately in chapter 13). All the modes produce television pictures by using *raster scan* technology; this is also true of most of the newer commercial mini and main-frame computers. An area of memory at least 1K(ilo)byte long (1 Kbyte = 2^8 bytes = 1K for short), known as the *screen memory*, is reserved out of the available RAM (Random Access Memory – the area aailable for programming use) to hold the display information for the screen. This memory is examined, bit by bit, as the electron beam sweeps across the raster screen. The display is composed of dots or *pixels* (from picture-cells) each of which, in the simplest case of modes 0, 3, 4 and 6, is represented by a single bit (a binary on/off switch) in the memory. Whenever a binary-on is detected during the raster scan, the beam is switched on for a short period, so producing a dot of light on the screen. in the other modes more than one bit corresponds to each pixel (see later). The screen can be considered in two ways; either as a grid of individual points that are addressed by *graphics* commands or as a grid of blocks in which characters can be placed by text commands.

The MODE Command

On the BBC micro there is a palette of sixteen different *actual* colours/effects (numbered 0 to 15) and the MODE command is used to decide how many different colours Hom this palette will be available and what type of display is used.

MODE N switches to display mode N and decides how much memory must be set aside for the screen memory. The number of pixels (known as the resolution) and TEXT characters available, as well as their physical size, alter with each MODE. The various modes are detailed in table 1.1.

Table 1.1

MODE	TEXT characters (column x row)	Graphics pixels (horizontal x vertical)	Number of colours	Memory used	Pixel
0	80 x 32	640 × 256	2	20K	2 × 4
1	40 x 32	320 × 256	4	20K	4 × 4
2	20 x 32	160 × 256	16	20K	8 × 4
3	80 x 25	–	2	16K	–
4	40 x 32	320 × 256	2	10K	2 × 4
5	20 x 32	160 × 256	4	10K	8 × 4
6	40 x 25	–	2	8K	–
7 TELETEXT	40 x 25	80 × 75	16	1K	–

In this chapter neither the TELETEXT mode (7), which is dealt with separately in chapter 13, nor the two text-only MODEs (3 and 6) will be considered. 1n the two-colour modes (0 to 4) each pixel is represented by one bit in the memory. This bit defines which *logical* colour is used for that pixel and initially it is set to display a white pixel for a logical 1 and a black pixel for a logical 0. It is possible to change these default assignments (see listing 1.9) so that any actual colour may be displayed for either logical colour. In the four-colour modes (1 and 5) two bits of memory are used to represent each pixel. These two bits represent, in binary notation, a number between 0 and 3 which is the numerical code of the logical colour displayed for that pixel. This allows us to distinguish between four types of pixel and to use a different colour for each type. In the sixteen-colour mode (2) four bits are used per pixel to make up a logical colour between 0 and 15. For all the standard graphics modes the number of pixels vertically is 256; however, since more memory is required to represent a sixteen-colour pixel compared to a four-colour pixel, the number of points available horizontally varies inversely with the number of colours used.

Screen Memory

This type of screen picture is referred to as a memory mapped display since it corresponds directly to the contents ofan area of memory. On the BBC micro the memory used for the display, called the screen memory, starts at location

HIMEM (which is reset by the MODE command) and ends at location 32767 (the end of RAM). A simple exploration ofhow the screen is affected by changing the contents of the memory can be made with a program such as listing 1.1 .

Listing 1.1

```
10 REPEAT
20 INPUT'" Which mode ",M
30 MODE M
40 ?HIMEM=137
50 UNTIL FALSE
```

This program uses the indirection operator '?' (see the user guide) to indicate the intention of placing a number (VALUE) in the memory location with address HIMEM. This is the first location of the display file and it holds the information for the top left-hand corner ofthe screen. Run the program and select mode 4 (or 0). Since each location, or *byte*, contains eight binary *bits*, the first eight pixels on the display are affected in such two-colour modes. These change to show a pattern of dots that is equivalent to the binary representation of the VALUE, in this case 10001001. Run the program again but now choose a four-colour mode (1 or 5). This time only four pixels will be affected since two bits are used per pixel. Try the same program with the sixteen-colour mode (2) and you will see that only two pixels are now affected since four bits per pixel are used. Table 1.2 shows how the eight bits in one byte are split up by different modes to represent the logical colours of the different number of pixels that use the specific value 10001001.

From the above we see that it is possible to construct a complete picture by storing various values in the locations of the display file. This is tedious for two colour pictures, and extremely complicated for pictures with a greater number of colours. Obviously we need a simpler method for altering the contents of the display file. BASIC provides the graphics commands that deal precisely with this problem. The first command to be considered is PLOT, a very complicated command that offers many options, as we shall see later. For the time being, however, we shall limit ourselves to using just three options, PLOT 69, PLOT 4 and PLOT 5. Two of these are considered so important that they are given alternative names: PLOT 4 is MOVE and PLOT 5 is DRAW.

Because the number of pixels and their relative positions are MODE-dependent, a new object is defined for the BBC micro, the *addressable point* (or *point* for short). All the graphics commands treat the display as a grid of 1280 addressable points horizontally by 1024 addressable points vertically (1 310 720 in total). Each point is uniquely defined by a pair of integers such that point (X, Y) is X addressable points to the left and Y points above the screen origin (point (0, 0) at the bottom left-hand corner of the screen). We have already seen that the number of available pixels is mode-dependent; in fact each pixel is composed

Table 1.2

Two-colour MODES

	1	0	0	0	1	0	0	1	
pixel 1 =	B_7								= 1 = logical colour 1
pixel 2 =		B_6							= 0 = logical colour 0
pixel 3 =			B_5						= 0 = logical colour 0
pixel 4 =				B_4					= 0 = 1ogical colour 0
pixel 5 =					B_3				= 1 = logical colour 1
pixel 6 =						B_2			= 0 = logical colour 0
pixel 7 =							B_1		= 0 = logical colour 0
pixel 8 =								B_0	= 1 = logical colour 1

Four-colour MODES

	1	0	0	0	1	0	0	1	
pixel 1 =	B_7				B_3				= 11 = logical colour 3
pixel 2 =		B_6				B_2			= 00 = logical colour 0
pixel 3 =			B_5				B_1		= 00 = logical colour 0
pixel 4 =				B_4				B_0	= 01 = logical colour 1

Sixteen-colour MODE

	1	0	0	0	1	0	0	1	
pixel 1 =	B_7		B_5		B_3		B_1		= 1010 = logical colour 10
pixel 2 =		B_6		B_4		B_3		B_0	= 0001 = logical colour 1

of a small block of addressable points (see the last column of table 1.1). This correspondence between points and pixels is all worked out by the computer and means that, since we are working with addressable points, we can switch between MODEs without having to change the programs. Any command that affects one point will actually affect the whole pixel that contains this point. The use of points is a great help when changing between MODEs since point (640, 512) always represents the middle of the screen, whereas if we counted in pixels then pixel (80, 128) is close to the left side of the screen in mode 0, one-quarter of the way across the screen in mode 1 and the middle of the screen for mode 2 only. The graphics commands help in constructing pictures by allowing us to control a *graphics pen*, which is initially positioned over point (0, 0).

PLOT 4, X, Y or MOVE X, Y moves the pen from its current position and places it above the pixel that contains the point (X, Y).

PLOT 69, X, Y moves the pen to the point (X, Y) and plots a pixel there.
PLDT 5, X, Y or DRAW X, Y draws a line from the pen's current position to the point (X, Y).

After the execution of these coinmands the pen remains over the last point that was visited, wMie awaiting the next command. Before examining the other more advanced graphics commands, a simple example and some exercises will serve to demonstrate what can be achieved with only the few commands that have been dealt with so far.

Example 1.1
PLOT 69 can be used to scatter pixel dots over the screen. The program in listing 1.2 illustrates the flexibility of addressing the pixels via the overlying grid of addressable points.

Listing 1.2

```
10 INPUT"Which mode",M : MODE M
20 REPEAT
30 PLOT 69,RND(1280),RND(1024)
40 UNTIL FALSE
```

Exercise 1.1
Alter listing 1.2 to DRAW lines, either between the random points as they are generated or from the middle (point (640, 512)) to each point.

Exercise 1.2
Write a program to calculate the position of lines that form a grid on the screen. DRAW them by using two FOR. . .NEXT loops (one for horizontal lines, the other for vertical lines).

Exercise 1.3
Write a prograin that accepts the INPUT of N pairs of addressable point coordinates from the keyboard, and then DRAWs an irregular polygon of N sides by joining the points in order. (Remember to join the last point to the first.)

PRINT, LIST and VDU

So far we have not discussed the most obvious method of changing the display, namely the text commands PRINT (PRINT TAB), LIST and VDU (consult the user manual for a full description of these commands). This is because these use character-size text blocks and are designed primarily for use with low-resolution graphics. This will be dealt with in chapter 5, but since the BBC micro allows

high-resolution and low-resolution graphics to be freely intermixed we shall give a minor exainple here. Suppose we alter the program from exercise 1.2 to draw a grid of the appropriate size for character blocks. If we run the program, select a MODE and press control E (mix text and graphics) followed by LIST, followed by control D (separate text and graphics), we get a display similar to figure 1.1, which shows the size and position of the character blocks.

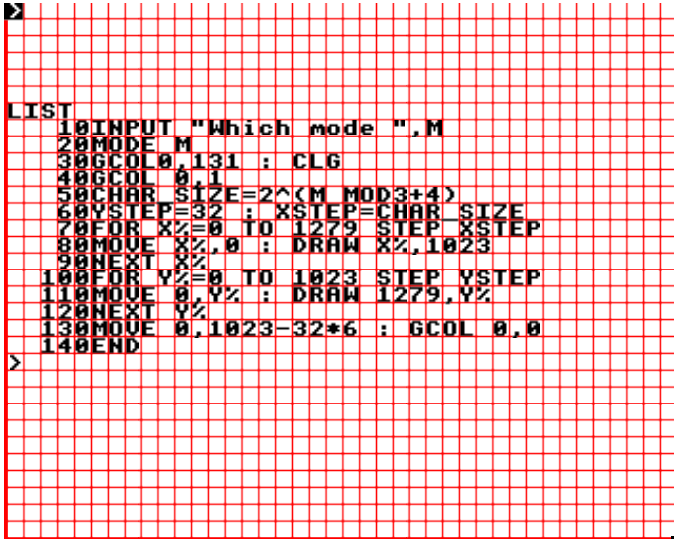


Figure 1.1

Example 1.2

We can use the PLOT command to demonstrate the high-resolution capabilities of the BBC micro by drawing *fractals* (see Mandelbrot, 1977).

To draw a simple fractal we proceed as follows. Imagine a square with sides of length 4^n. This may be divided into 16 smaller squares, each with sides of length 4^{n-1}; the smaller squares are numbered 1 to 16 in accordance with the pattern of figure 1.2. Four of these smaller squares, numbers 2, 8, 9 and 15, are rearranged to produce figure 1.3.

Each ofthe squares in the pattern is now split up into 16 even smaller squares in the same way and these are similarly rearranged. This process is repeated until we have squares with sides of length 1. The resulting fractal pattern consists entirely of unit squares which we can PLOT 69 as single pixels (since we know how many pixels are available in each mode we can work out the area covered

by one pixel in terms of addressable points). The program in listing 1.3 starts from a square with sides of length 6^4, which is 4^3; thus in the program there must be three FOR. . .NEXT 1oops nested inside each other. The final picture produced is shown in figure 1.4.

Listing 1.3

```
 10 MODE 1
 20 D1M X(16),Y(16)
 29 REM assign an X and Y coordinate to each square
 30 FOR 1%=1 TO 4
 40 FOR J%=1 TO 4
 50 K%=4'1%+J%-4
 60 X(K%)=J%-3 : Y(K%)=1%-3
 70 NEXT J% : NEXT1%
 79 REM move squares 2,8,9 and 15
 80 X(2)=0 : Y(2)=-3
 90 X(8)=2 : Y(8)=0
100 X(9)=-3 : Y(9)=-1
110 X(15)=-1 : Y(15)=2
119 REM plot each square inside a square inside a square as
      a single pixel
120 FOR 1%=1 TO 16
130 FOR J%=1 TO 16
140 FOR K%=1 TO 16
150 XX=16'X(1%)+4'X(J%)+X(K%)
160 YY=16'Y(1%)+4'Y(J%)+Y(K%)
170 PLOT 69,640+XX'4,512+YY'4
180 NEXT K% : NEXT J% : NEXT 1%
190 END
```

We shall now consider the options that affect the colour of the lines and the points that are placed on the screen. There are two commands that allow us to select a new logical colour (one affects text and the other affects graphics).

COLOUR COL is the command used to change text colours. COL is an integer between 0 and the number of colours available in the present mode, and represents the new foreground colour for text. If we use COLOUR 128 + COL we can change the background colour of the text. All subsequent printing of characters will be affected by this command.

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Figure 1.2

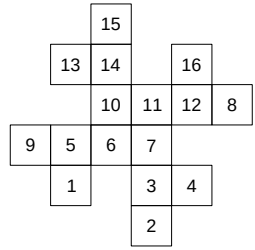


Figure 1.3

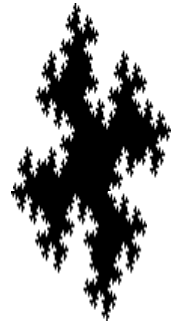


Figure 1.4

GCOL G, COL ($0 \leq G \leq 4$) is the command that affects the colour produced by further PLOT (and DRAW) commands. As with COLOUR above, COL can select a foreground colour and the colour for the lines and points plotted, or with GCOL G, 128 + COL a background colour. Graphics background colours will be seen only when a CLG (clear graphics) command is executed, so filling the whole of the graphics area with the background colour.

Before we go on to explain the other part of the GCOL command (the G option), try the program in listing 1.4 which uses the simplest form of GCOL (GCGL 0) to change the colour that is used for drawing lines. This sort of pattern, joining equivalent points on two curves, is known as a Lissajous figure.

The G-parameter in the GCOL command affects the way in which the colours are added to the screen. The effect produced by various values of G are explained opposite. (For further explanation of the operators AND, EOR and OR see the user guide.)

Listing 1.4

```
  9 REM Lissajous figures
 10 MODE 2
 20 MOVE 1140,512
 30 FOR I=0 TO 2*PI STEP PI/100
 39 REM calculate the 2 points on 2 curves and join with
       a line drawn in a random colour
 40 S=SIN(I) : C=COS(I)
 50 S2=SIN(2*I) : C2=COS(2*I)
 60 X1=640+C^3*500 : Y1=512+S^3*500
 70 R=(C2^3+S2^3)*500
 80 X2=640+R*C : Y2=512+R*S
 90 GCOL 0,RND(7)
100 MOVE X1,Y1 : DRAW X2,Y2
110 NEXT I
```

G Operator	*Effect*
0 REPLACE	All pixels affected are changed to the colour that is specified by COL
1 OR	Pixels affected become the colour that is given by their present (logical) colour OR COL. For example, in mode 1 a pixel that was colour 1 (binary 01) replotted in colour 2 (binary 10) would become colour 3 (binary 11 = 01 OR 10)
2 AND	Pixels affected become the colour that is given by their present (logical) colour AND COL. Thus, in mode 2 a pixel that was colour 5 (binary 0101) replotted in colour 6 (binary 0l10) would become colour 4 (binary 0100 = 0101 AND 0110)
3 EOR	Pixels affected become the colour that is given by their present (logical) colour EOR COL. For example, in mode 2 a pixel that was colour 5 (binary 0101) replotted in colour 7 (binary 0111) would become colour 2 (binary 0010 = 0101 EOR 0111). It is worth noting that two identical EOR operations have the effect of cancelling each other out. 1f we were to PLOT the same pixel again with colour 7 it would be returned to colour 5
4 INVERT	Pixels affected become the colour that is given by inverting all the binary digits in their present (logical) colour. Note that this is exactly the same as EOR with 15 (binary 1111). Thus, in mode 2 a pixel that was colour 5 (binary 0101) replotted in any colour would become colour 10 (binary 1010 = 0101 EOR 1111)

Exercise 1.4
Experiment with the program in listing 1.4 by using different GCOL options and perhaps by making the colours change in a non-random way. Try different equations for calculating the radius at each angle of the Lissajous fiares.

Example 1.3
Using these options for the GCOL command we can produce simple programs that generate seemingly complex patterns. Listing 1.5 gives a program that uses option 3 (EOR) for GCOL combined with a method of creating complicated patterns.

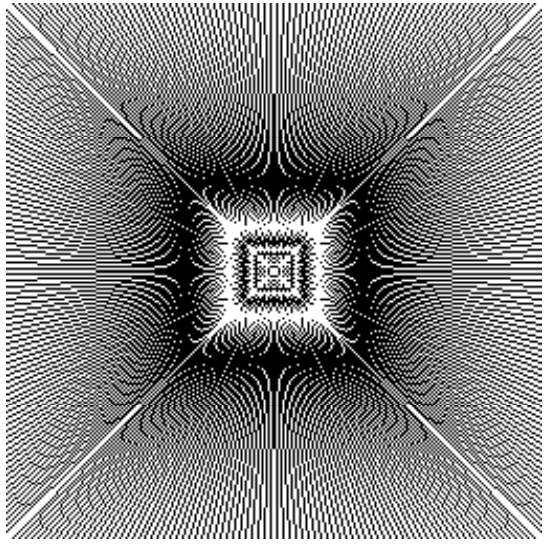
On a display that is composed of discrete points (pixels), angled lines will be drawn as a series of short, horizontal or vertical steps. Many of the steps on the lines will overlap when two such lines are drawn close together at slightly different angles. Consider figure 1.5, which is drawn by listing 1.5. The lines that form the central area overlap each other many times and so this area would have been solid white if EGR had not been used. However those pixels that lie on an even number oflines are not affected (as noted above) and only those that are PLOTted an odd number oftimes are lit up. This produces the striking pattern at the middle of the figure. On the other hand the outer area ofthe pattern is produced by holes (of pixels not lying on any line) that are left by the line steps.

Listing 1.5

```
 9 REM joins points around a square to form a Moire pattern
10 MODE1
20 GCOL3,3
30 FOR I%=0 TO 1023 STEP 12
40 MOVE I%+128,0
50 DRAW (1023-I%)+128,1023
60 MOVE 128,I%
70 DRAW 1023+128,1023-I%
80 NEXT I%
```

Exercise 1.5
Alter listing 1.5 to INPUT the value of the STEP-size used and also INPUT the variable that indicates which GCOL option is to be used.

Example 1.4: Simple Animation
We note from listing 1.5 that the EOR option ensured that the display changed with each new command, even if the previous command was repeated by DRAWing the same point, or line. This property may be used to display an object briefly by drawing it twice – the first to put it on the screen and the second to take it off. Listing 1.6 moves a dot around the screen by PLOTting it at its new position and immediately PLOTting its last position again to remove the old point.



Listing 1.6

```
 10 MODE 2 : VDU 19,0,6,0,0,0
 20 GCOL 3,1
 30 SPEED=8 : X=0 : Y=0
 40 XADD=SPEED : YADD=SPEED
 50 PLOT69,X,Y
 60 REPEAT : OLDX=X : OLDY=Y
 70 X=X+XADD : IF X > 1279-SPEED OR X < SPEED THEN XADD=-XADD :
    SOUND 1,-15,200,1
 80 Y=Y+YADD : IF Y > 1023-SPEED OR Y < SPEED THEN YADD=-YADD :
    SOUND 2,-15,100,1
 90 PLOT 69,X,Y
100 PLOT 69,OLDX,OLDY
110 UNTIL FALSE
```

Example 1.5

We may extend this program to allow keyboard control of the moving point (listing 1.7). The cursor control keys (either singly or in combination) enable the point to move in eight separate directions under our control. If a ‘p’ is typed then the point leaves a trailing line that indicates its past movements: if a ‘q’ is typed then the point ceases to leave this trail.

This type of animation is an important and commonly used technique. We shall use it extensively both in programs such as the game in chapter 15, and in programs such as the ‘cursor’ routine in chapter 6.

Listing 1.7

```
 10 MODE 1
 20 GCOL 3,1
 30 X=0 : Y=0
 40 PLOT69,X,Y
 50 REPEAT
 60 XADD=0 : YADD=0
 70 OLDX=X : OLDY=Y
 79 REM check cursor keys for direction of movement
 80 IF INKEY(-58) AND Y < 1019 THEN YADD=4
 90 IF INKEY(-42) AND Y > 0 THEN YADD=-4
100 IF INKEY(-26) AND X > 0 THEN XADD=-4
110 IF INKEY(-122) AND X < 1275 THEN XADD=4
119 REM check 'p' to  leave a trail
120 IF INKEY(-56) THEN GCOL 0,1
130 IF INKEY(-17) THEN GCOL 3,1
140 IF XADD=0 AND YADD=0 THEN 80
150 X=X+XADD : Y=Y+YADD
160 PLOT 69,X,Y
170 PLOT 69,OLDX,OLDY
180 UNTIL FALSE
```

Example 1.6: Relative Plotting

All the PLOT commands that hae been given so far have used the absolute coordinates of the points that we were drawing; however, there are equivalent relative commands for PLOTting. These commands move the pen to the position that is X points to the left of and Y points above the original point. Listing 1.8 uses the relative version of the DRAW option to produce an animated effect over a large area simply by changing the lines around the edge of the area.

Listing 1.8

```
 10 MODE 2 : UP=255 : ACROSS=159
 20 X=0 : Y=0 : DIF=1 : I%=1
 30 REPEAT
 39 REM draw the ever decreasing or ever increasing rectangles
 40 GCOL 0,I%
 50 MOVE X,Y
 60 PLOT 1,0,UP*4 : PLOT 1,ACROSS*8,0
 70 MOVE X,Y
 80 PLOT 1,ACROSS*8,0 : PLOT 1,0,UP*4
 90 X=X+DIF*8 : Y=Y+DIF*4
100 UP=UP-2*DIF : ACROSS=ACROSS-2*DIF
109 REM if rectangle is a trivial line then start drawing outward
          if it is the outer rectangle then draw inwards
110 IF ACROSS<0 OR ACROSS=159 THEN DIF=-DIF : I%=(I% MOD 7)+1
120 UNTIL FALSE
```

Logical and Actual Colours (VDU 19 and VDU 20)

On the BBC microcomputer there is a palette of sixteen different colours/effects known as actual colours, any of which can be assigned to an available logical

colour. We can redefine the relationship between logical and actual colours with the VDU 19 command. The format of this command is the characters VDU 19 followed by a list of five integers, these numbers being the code for the logical colour, the code for the actual colour and three zeros (to allow for future ‘wansion of the number of colours in the palette). The program in listing 1.9 allows us to see all the colours available in each mode, and to redefine them. VDU 20 will restore all logical colours to their original (default) settings.

We can make a simplified interpretation of the pixels on a colour television screen by imagining them as groups of three dots of light that are packed closely together at the vertices ofan equilateral triangle. For each pixel there is one red,

Listing 1.9

```
  9 REM N(I) is the number of colours in mode I.
    C(I,J) is the actual colour of logical colour J in mode I
 10 DIM N(6),C(6,15)
 20 FOR M=0 TO 6: READ N(M) : NEXT
 30 FOR M=0 TO 6 : FOR C=0 TO N(M)-1
 40 READ C(M,C)
 50 NEXT C : NEXT M
 60 REPEAT : CLS : INPUT "Which mode ",M
 70 MODE M
 80 PRINT TAB(0,1);"MODE ";M;" : ";N(M);" COLOURS"
 90 PRINT TAB(0,3);"LOGICAL     ACTUAL"
 99 REM for given mode draw a table of logical/actual relationships
100 FOR I=0 TO N(M)-1
110 COLOUR 128+0 : COLOUR (N(M)-1) MOD 8
120 PRINT TAB(2,I+4);I
130 PRINT TAB(7,I+4);"= ";
140 COLOUR 128+I : PRINT"   ";
150 COLOUR 128+0 : PRINT"  ";C(M,I)
160 NEXT I
170 COLOUR 128+0 : COLOUR (N(M)-1) MOD 8
179 REM"change array C for a new  logical/actual relation
180 PRINT TAB(0,21); :
    INPUT"Do you want to alter colour settings ",A$
190 IF A$<>"Y" THEN 260
200 PRINT"Alter LOGICAL colour    "; : VDU 8,8,8 : INPUT A
210 PRINT"to be ACTUAL  colour    "; : VDU 8,8,8 : INPUT B
220 A=A MOD N(M) : B=B MOD 16
229 REM"change the display table
230 VDU 19,A,B,0,0,0 : C(M,A)=B
240 PRINT TAB(14,A+4);C(M,A);" "
250 GOTO 180
260 UNTIL FALSE
270 MODE 7
280 END
289 REM"DATA about mode/colour
290 DATA 2,4,16,2,2,4,2
300 DATA 0,7
310 DATA 0,1,3,7
320 DATA 0,1,2,3,4,5,6,7,8,9,10,11,12,12,14,15
330 DATA 0,7
340 DATA 0,7
350 DATA 0,1,3,7
360 DATA 0,7
```

one green and one blue dot, and the binary value of the actual colour is used to control the illumination of the three different dots. A location in the screen memory holds the information that indicates the logical colour of a particular pixel, and the equivalent actual colour to be displayed is looked up by the computer. The lowest three bits (bits 0 to 2) of the actual colour are used to decide whether the red, green and blue dots of that pixel are on or off. Our eyes contain only three types of colour sensor (red, green and blue). Our brains take the signals from the three dots and combine them into a single pixel of composite colour. So if the last three bits of the actual colour are 111, equivalent to colour 7, we get red plus green plus blue, which corresponds to white light. The other colour codes, when written in binary form, can be translated in this way: see table 1.3.

Table 1.3

Colour	Number	Binary	Illuminated dots
Black	0	000	
Red	1	001	Red
Green	2	010	Green
Yellow	3	011	Green + Red
Blue	4	100	Blue
Purple	5	101	Blue + Red
Cyan	6	110	Blue + Green
White	7	111	Blue + Green + Red

If the actual colour has the fourth bit set (bit 3), then the colour will flash between the colour that corresponds to the lower three bits and the inverse of this colour, that is colour EOR 7 (binary 111). The speed of flashing depends on a counter in the computer, and the setting can be changed by the *FX9 and *FX10 calls to the computer. This counter is at location &251; it is loaded first with the value stored at &252 which is then decremented every 1/100th of a second until zero is reached. Then the second colour is displayed while the counter goes from the value at &253 to zero, and the process is repeated. Listing 1.10 shows this process in action in mode 2. Change the values at &252 and &253 and rerun the program to see their effect.

Exercise 1.6

Experiment with different colours by using the programs from this chapter. Certain colour combinations can be too complicated for an ordinary television screen to handle. Unless you are using an expensive monitor rather than a television screen, the combination of flashing colours for the program in listing

Listing 1.10

```
10 MODE 2
20 VDU 23,1,0;0;0;0;
30 COLOUR 9
40 *FX10,99
50 *FX9,99
60 PRINT "THIS SHOWS THE FLASH COUNTER"
70 COLOUR 13
80 REPEAT PRINT TAB(0,1) : PRINT ?(&251) DIV 10 : UNTIL FALSE
```

1.5 should produce a rather interesting effect of waves washing across the screen.

Example 1.7: Colour Animation

We can produce more animated effects by using flashing and non-flashing colours. Listing 1.11 shows some interesting techniques of colour animation. The first part of the program (which produces the boundary) is particularly useful because the display, once set up, needs no maintenance. The boundary is a sequence of blocks that is made up of alternate blocks of flashing blue/yellow and flashing yellow/blue foreground colours. On seeing this our brains are tricked into believing that the yellow and blue colours are moving around the boundary sequence. The second part of the program simply scrolls the words 'THIS IS YOUR BBC COMPUTER SPEAKING' in different colours up the screen.

Listing 1.11

```
  10 MODE 2
  19 REM print boundary
  20 FOR I%=0 TO 1
  30 COLOUR 128+11+I%
  40 FOR X%=0 TO 9
  50 PRINT TAB(X%*2+I%,0);" " : PRINT TAB(X%*2+1-I%,29);" "
  60 NEXT X%
  70 FOR Y%=0 TO 14
  80 PRINT TAB(0,2*Y%+I%);" " : PRINT TAB(19,2*Y%+1-I%);" "
  90 NEXT Y%
 100 NEXT I%
 109 REM read message
 110 DIM A$(7)
 120 FOR I%=1 TO 7 : READ A$(I%) : NEXT I%
 130 DATA " HELLO  "," THIS  "," IS  "," YOUR  ",
       " B.B.C. ","COMPUTER","SPEAKING"
 140 C%=1
 150 REPEAT
 159 REM scroll message
 160 FOR I%=1 TO 28
 170 COLOUR 128+C% : COLOUR (C% EOR 7)
 180 PRINT TAB(1,I%);SPC(5);A$(C%);SPC(5)
 190 C%=(C% MOD 7)+1
 200 NEXT I%
 210 C%=(C% MOD 7)+1
 220 UNTIL FALSE
```

Exercise 1.7
Write text and colour versions of the bouncing point program and the other animation programs. In your programs move characters instead of pixels around the screen.

Filling in Areas

The BBC micro has still more PLOT options, some of which deal with drawing dotted lines where only every other pixel in their path is illuminated. Try filling the MODE 0 screen with dotted white horizontal lines on a black background by using the dotted equivalent of DRAW, that is PLOT 21. It is possible that some areas appear pale purple while others are pale green. This is because the size of the points plotted is almost as small as the three-coloured dots that make up the colours and in some areas the points lie more towards the green side of the triangle while in others they are more to the red or the blue sides. Try redefining the background and foreground colours of the display to see what other effects can be produced.

Now look at the group of PLOT commands that fill in triangles. The most important of these is PLOT 85, the absolute foreground colour version (see the user guide). This command constructs a triangle between the point currently being plotted and the last two points visited, and the area enclosed is filled with the current graphics colour. As a simple example, try typing the following commands, which will produce a large red triangle.

```
MODE 2 : GCOL 0, 1 : MOVE 100, 100 : DRAW 500, 900 : PLGT 85,
1000, 100
```

Example 1.8
The program in listing 1.12 produces a pattern by PLOTting a series of equilateral triangles in varying orientations and in different colours. It then uses the VDU 19 command to redeHne each colour in turn, so producing the illusion of rotational movement. It requires the INPUT of two integers N and ST ; N controls the rotation of consecutive triangles and ST gives the difference in radius between the triangles.

Example 1.9: A Simple Game
We now include a small game program (listing 1.13) to demonstrate the use of the techniques discussed in this chapter. A worm can move in character-size steps about the screen, horizontally or vertically, under control of the keyboard. The aim of the game is for the worm to eat the money (or target). The worm gets longer whenever it eats the target. If at any time the head of the worm runs headlong into the boundary or into its own body, then the worm dies. After ten successful meals the worm returns to its original size, with a fanfare; the game then continues.

Listing 1.12

```
 10 MODE 2
 20 VDU23,1,0;0;0;0;
 30 DIM X(96),Y(96)
 39 REM"setup 96 points on a circle
 40 THETA=0 : TD=PI/48
 50 FOR I%=1 TO 96
 60 X(I%)=COS(THETA) : Y(I%)=SIN(THETA)
 70 THETA=THETA+TD
 80 NEXT I%
 90 COLOUR128
100 REPEAT
109 REM every new triangle is rotated through N points,
       its radius changes by ST
110 INPUT "N ",N
120 IF N=0 THEN STOP ELSE INPUT "ST ",ST : CLS
130 N3=N/3-1
140 I1=1 : COL=1
150 FOR R=500 TO 5 STEP -ST
159 REM"draw triangle in colour COL
160 GCOL 0,COL
170 I2=I1+32 : I3=I2+32
180 MOVE 640+R*X(I1),500+R*Y(I1)
190 MOVE 640+R*X(I2),500+R*Y(I2)
200 PLOT 85,640+R*X(I3),500+R*Y(I3)
210 COL=(COL MOD 7 )+1
220 I1=((I1+N3) MOD 32 ) +1
230 NEXT
239 REM"wait 5 seconds
240 T=TIME+500 : REPEAT UNTIL TIME>T
249 REM"rotate picture by colour swap for 8 seconds
250 T=TIME+800 : REPEAT
260 FOR J%=1 TO 7
270 FOR I%=1 TO 7
280 VDU 19,I%,((I%+J%) MOD 7)+1,0,0,0
290 NEXT I%
300 ST=TIME+10 : REPEAT UNTIL TIME>ST
310 NEXT J%
320 UNTIL TIME>T
330 VDU 20
340 UNTIL FALSE
```

This game was developed using the modular, structured methods that are preferred by pmgraminers. These methods help quickly to produce a working and understandable program. Put simply, we must approach the program as a series of small tasks that build up block by block into the completed program. For the game below, these tasks were tentatively defined as

(1) Initialise variables
(2) Set up board
(3) Control game
(4) Update and print score

From this overview of the program we can set about solving each problem or if necessary splitting it into yet smaller, more manageable, problems. For example, task (3) above could be split into

(A) Generate target
(B) Use keyboard to change direction of worm
(C) Move worm

Task (C) could be further split into

(a) Draw worm
(b) Make worm die if it hits boundary or itself
(c) Make worm grow if it eats money
(d) Generate fanfare

No specific order is implied in this breakdown; for example, you may find that you want to regenerate the target from inside the fanfare section of the program. These headings are simply lists of tasks that reflect the problems that come to mind when attempting the solution of a larger problem.

Examine the game below and try to identify which tasks are carried out, where, in what order, and which have been further subdivided. (Throughout book the names of procedures are in lower case characters and are preceded by a red REMark when listed in MODE 7. This helps to make the program more readable and gives a clear picture of the algorithm; hence it is good general practice.)

Note the use of logical expressions (for example, UNTIL DEAD OR WON) see the user guide. Also note the use of the OSBYTE call with A% = &87 to detect collisions by examining the contents of character blocks. Figure 1.6 shows a typical state of the game.

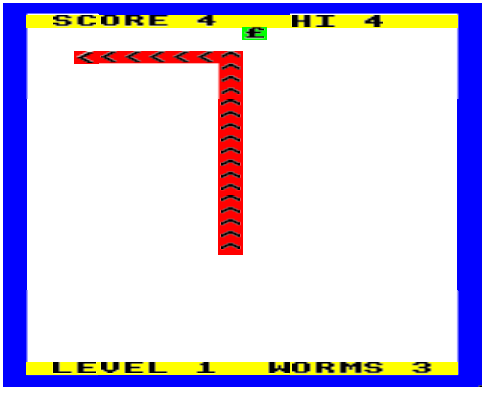


Figure 1.6

Listing 1.13

```
1000 REM"WORM GAME
1010 DIM R(55),C(55) : HSC=0
1019 REM prepare screen
1020 REPEAT : MODE 2 : SCORE=0 : WORMS=3 : LEVEL=1
1030 VDU 23,1,0;0;0;0; : GCOL 0,128+4 : CLG
1040 GCOL 0,7 : MOVE 64,32 : MOVE 64,992
1050 PLOT 85,1215,32 : PLOT 85,1215,992
1059 REM initialise worm
1060 S=5 : P=1 : R=1 : C=RND(18)
1070 RMOVE=1 : CMOVE=0 : H$="V"
1080 FOR I%=0 TO 55 : R(I%)=-1 : NEXT I%
1090 WON=FALSE : DEAD=FALSE : *FX15,1
1100 COLOUR 128+3 : PRINT TAB(1,1);SPC(18) :
     PRINT TAB(1,30);SPC(18)
1109 REM display score and set target
1110 PROCstatus : PROCtarget
1119 REM"main loop
1120 REPEAT
1130 PROCkey : PROCworm : PROCdelay(5-LEVEL)
1140 UNTIL DEAD OR WON
1150 IF WON THEN PROCfanfare : GOTO 1030
1159 REM wipe out dead worm and make crashing noise
1160 FOR I%=-15 TO 1 : SOUND 0,I%,4,1 : NEXT I%
1170 COLOUR 128+7
1180 PRINT TAB(X,Y);" "
1190 FOR I%=1 TO S
1200 IF R(P)<>-1 THEN PRINT TAB(C(P),R(P));" "
1210 P=(P MOD S)+1 : SOUND1,-15,10,1
1220 NEXT I%
1230 WORMS=WORMS-1 : IF WORMS<>0 THEN 1060
1239 REM if no worms left then wait for return to start again
1240 COLOUR 128+12 : COLOUR 11 : PRINT TAB(0,31); : *FX15,1
1250 INPUT "HIT RETURN FOR GAME" A$
1260 UNTIL FALSE
2000 REM"fanfare
2010 DEF PROCfanfare
2019 REM"congratulatory jingle
2020 DATA 2,69,117,165, 2,73,121,169, 2,81,129,177,
          5,102,150,198, 2,81,129,177, 7,102,150,198
2030 RESTORE : FOR N%=1 TO 6 : READ D
2040 FOR C%=&201 TO &203
2050 READ P : SOUND C%,-15,P,D
2060 NEXT C%
2070 NEXT N%
2080 ENDPROC

3000 REM worm
3010 DEF PROCworm
3019 REM move worm by adding new head segment and deleting tail
3020 COLOUR 128+7 : IF R(P)<>-1 THEN PRINT TAB(C(P),R(P));" "
3030 R=R+RMOVE : C=C+CMOVE
3040 IF R<2 OR R>29 OR C<1 OR C>18 THEN DEAD=TRUE : ENDPROC
3050 PRINT TAB(C,R); : A%=135 : I%=USR(&FFF4)
3060 A$=CHR$((I% AND &FF00)/&100) : IF A$=" " THEN 3100
3070 COLOUR 128+2 : COLOUR 0 : A%=135 : I%=USR(&FFF4)
3080 A$=CHR$((I% AND &FF00)/&100)
3089 REM if not space it was either a wall or edible money.
3090 IF A$="£" THEN PROCgobble ELSE DEAD=TRUE : ENDPROC
3100 R(P)=R : C(P)=C : COLOUR 128+1 : PRINT TAB(C,R);H$
3110 P=(P MOD S)+1
3120 ENDPROC
```

```
4000 REM key input
4010 DEF PROCkey
4020 A$=CHR$(ASC(INKEY$(0)) AND &4F)
4030 IF A$="I" AND CMOVE THEN RMOVE=-1 : CMOVE=0 : H$="^"
4040 IF A$="M" AND CMOVE THEN RMOVE=1 : CMOVE=0 : H$="V"
4050 IF A$="J" AND RMOVE THEN RMOVE=0 : CMOVE=-1 : H$="<"
4060 IF A$="K" AND RMOVE THEN RMOVE=0 : CMOVE=1 : H$=">"
4070 ENDPROC

5000 REM gobble
5010 DEF PROCgobble
5019 REM"make noise while chewing and add to score.
5020 FOR I%=1 TO 6 : SOUND1,I%-15,20+I%*6,1: NEXT I%
5030 SCORE=SCORE+1 : PROCstatus
5039 REM"if worm is 55 segments long you have eaten £10, you win
5040 S=S+5 : IF S=55 THEN WON=TRUE : ENDPROC
5050 PROCtarget
5060 ENDPROC

6000 REM status
6009 REM show score and no. of worms
6010 DEF PROCstatus
6020 COLOUR 128+3 : COLOUR 0
6030 IF SCORE>HSC THEN HSC=SCORE
6040 PRINT TAB(1,1);" SCORE ";SCORE," HI ";HSC
6050 PRINTTAB(1,30);" LEVEL ";LEVEL,"WORMS ";WORMS
6060 ENDPROC

7000 REM target
7010 DEF PROCtarget
7019 REM produce randomly positioned  target, not inside worm.
7020 X=RND(18) : Y=RND(28)+1 : COLOUR128+7
7030 PRINT TAB(X,Y); : A%=135 : I%=USR(&FFF4)
7040 A$=CHR$((I% AND &FF00)/&100) : IF A$<>" " THEN 7020
7050 IF X=C AND Y=R THEN 7020
7060 COLOUR 128+2 : COLOUR 0
7070 PRINT "£"
7080 ENDPROC

8000 REM delay
8010 DEF PROCdelay(T)
8020 IF T>0 THEN TT=TIME+T : REPEAT UNTIL TIME>TT
8030 ENDPROC
```

Exercise 1.8

As a final miniproject for this chapter, write a squash game or ping-pong video game (or both) using low-resolution colour graphics. The ball can be a pixel or character block, and the bat(s) should be controlled from the keyboard like the worm in listing 1.13 . You may find it useful to turn some of the program sections from this chapter into procedures, which is readily done if you approach program writing in a prepared modular manner.

In this chapter we have restricted ourselves to using the screen as a fixed area for patterns and games. To step up from pixel graphics to drawing pictures of real objects we need commands that will relate the real world to our screen. We shall now explore and develop the techniques that are needed to draw real graphics

pictures. Before we go on to this you should experiment with the three programs given in listings 1.14, 1.15 and 1.16 and ensure that you understand the graphics conmands, since these are fundamental to understanding the rest of this book.

Listing 1.14

```
 10 MODE 1 : VDU23,1,0;0;0;0;
 20 VDU 19,0,4,0,0,0
 30 PROCcircle(640,750,1000,0,1)
 40 PROCcircle(640,750,150,2,2)
 50 PROCcircle(740,650,70,3,3)
 60 PROCcircle(865,680,100,3,3)
 70 PROCcircle(950,660,80,3,3)
 80 MOVE 740,580 : MOVE 740,720
 90 PLOT 85,950,580

100 END
110 REM circle
120 DEF PROCcircle(X,Y,R,C1,C2)
130 MOVE X,Y
140 F=TRUE : GCOL 0,C1
150 FOR I=0 TO 2*PI STEP PI/50
160 MOVE X,Y
165 PLOT81,R*COS(I),R*SIN(I)
170 F=NOT F
180 IF F THEN GCOL0,C1 ELSE GCOL0,C2
190 NEXT I
200 ENDPROC
```

Listing 1.15

```
 10 MODE 1 : VDU 23,1,0;0;0;0;
 20 VDU 19,2,4,0,0,0 : VDU 19,3,2,0,0,0
 30 FOR S=48 TO 200 STEP 24
 40 FOR J=1 TO (250-S)/25
 50 PROCcube(RND(1280-3*S/2),RND(1024-3*S/2),S,RND(4)-1)
 60 NEXT J
 70 NEXT S
 80 END
100 REM fake cube
110 DEF PROCcube(X,Y,S,C1)
120 IF C1<2 THEN C2=3 ELSE C2=0
130 T=S*.5
140 PROCquad(X,Y,X,Y+S,X+S,Y+S,X+S,Y)
150 PROCquad(X+S+T,Y+S+T,X+S+T,Y+T,X+S,Y,X+S,Y+S)
160 PROCquad(X+S+T,Y+S+T,X+T,Y+S+T,X,Y+S,X+S,Y+S)
170 ENDPROC
200 REM quad_rilaterial
210 DEF PROCquad(XA,YA,XB,YB,XC,YC,XD,YD)
220 GCOL 0,C1 : MOVE XA,YA : MOVE XB,YB :
    PLOT 85,XD,YD : PLOT 85,XC,YC
230 GCOL 0,C2 : DRAW XD,YD : DRAW XA,YA :
    DRAW XB,YB : DRAW XC,YC
240 ENDPROC
```

Listing 1.16

```
 10 DIM X%(81),X(100),Y(100)
 20 A=400 : B=100 : I%=0
 29 REM calculate data for ellipse
 30 FOR I=0 TO 2*PI STEP PI/50
 40 X(I%)=A*COS(I) : Y(I%)=B*SIN(I) : I%=I%+1
 50 NEXT I
 60 MODE 1 : GCOL 0,1 : VDU 19,1,6,0,0,0 : VDU5
 69 REM"put on sky
 70 MOVE 0,504 : MOVE 1280,504
 80 PLOT 85,0,1024 : PLOT 85,1280,1024
 90 GCOL 0,2 : VDU 19,2,4,0,0,0 : DRAW 1280,500
 99 REM calculate data for planks
100 FOR I%=0 TO 81
110 X%(I%)=64*(40-I%-(I% MOD 2)*0.66)
120 NEXT I%
129 REM"draw planks
130 FOR I%=0 TO 80 STEP 2
140 MOVE I%*16,500 : MOVE I%*16+16,500
150 PLOT 85,640-X%(I%),0 : PLOT 85,640-X%(I%+1),0
160 NEXT I%
169 REM draw pyramid with outline
170 GCOL 0,3 : VDU 19,3,5,0,0,0
180 MOVE 1136,400 : MOVE 960,350
190 PLOT 85,975,600 : PLOT 85,864,424
200 GCOL 0,0 : DRAW 960,350 : DRAW 1136,400
210 DRAW 975,600 : DRAW 960,350 : MOVE 975,600 : DRAW 864,424
219 REM draw cuboid with outline
220 GCOL 0,3 : MOVE 125,290 : MOVE 480,340
230 PLOT 85,125,490 : PLOT 85,480,510
240 PLOT 85,8,510 : PLOT 85,380,522
250 MOVE 125,290 : MOVE125,490
260 PLOT 85,8,314 : PLOT 85,8,510
270 GCOL 0,0 : MOVE 125,290 : DRAW 480,340 : DRAW 480,510
280 DRAW 125,490 : DRAW 8,510 : DRAW 8,314
290 DRAW 125,290 : DRAW 125,490
300 MOVE 8,510 : DRAW 380,522 : DRAW 480,510
309 REM draw ellipse
310 X=600:Y=200
320 GCOL 0,3 : MOVE X,Y
330 FOR I%=0 TO 100
340 MOVE X,Y : PLOT 81,X(I%),Y(I%)
350 NEXT I%
359 REM join lower side of ellipse to bottom of screen
360 FOR I%=50 TO 100 : J%=(I%+1) MOD 101
370 MOVE X+X(I%),Y+Y(I%) : MOVEX+X(J%),Y+Y(J%)
380 PLOT 85,X+X(I%),0 : PLOT85,X+X(J%),0
390 NEXT I%
399 REM draw outline around top of cylinder
400 GCOL 0,0 : MOVE X+X(100),Y+Y(100)
410 FOR I%=0 TO 100
420 DRAW X+X(I%),Y+Y(I%)
430 NEXT I%
439 REM draw lines down to bottom of screen from
       lower edge of ellipse
440 FOR I%=50 TO 100
450 MOVE X+X(I%),Y+Y(I%) : DRAW X+X(I%),0
460 NEXT I%
```

Complete Programs

I Listing 1.1. Data required: a mode value between 0 and 6.
II Listing 1.2. Data required: a mode value of 0, 1, 2, 4 or 5.
III Listing 1.3. No data required.
IV Listing 1.4. No data required.
V Listing 1.5. No data required.
VI Listing 1.6. No data required.
VII Listing 1.7. Use the cursor keys to move the red point about the screen. Type 'p' to leave a trail, and 'q' if no trail is wanted.
VIII Listing 1.8. No data required.
IX Listing 1 9. Data required:

> Mode? Type the required value between 0 and 6.
> Do you want to alter colour settings? Type Y(es) or N(o) as appropriate.
> Alter LOGICAL colour? Type a mode-dependent integer that is defined to be an ACTUAL colour; that is, type an integer between 0 and 15.

X Listing 1.10. No data required.
XI Listing 1.11. No data required.
XII Listing 1.12. Data required: two integers N and ST; try N = 14 and ST = 5.
XIII Listing 1.13. Keys 'I', 'J', 'K' and 'M' move the worm.
XIV Listing 1.14. No data required.
XV Listing 1.15. No data required.
XVI Listing 1.16. No data required.