20 Graphics

Introduction

This chapter deals with the **VDU** software – anything to do with how things are put on to the screen (ie the television or monitor). What 'modes' are and why they are there is covered first, followed by a section on writing text and then details on the graphics routines. Lastly the palette is covered. All the individual **VDU** commands are listed for reference in the next chapter.

Modes – what are they and why?

The screen displays things in any one of seven modes, labelled from **MODE 0** up to **MODE 6**. To change mode is easy – just type MODE followed by the mode number you want. For example

MODE 2 RETURN

changes the display to mode 2. As with all VDU commands, it can be used as a line in a program, and as it is a good idea to make sure your program starts off in the right mode, have its first line looking something like this:

```
10 MODE 1
```

Changing mode changes four things:

- The number of characters you can get on the screen.
- The number of pixels (dots) the graphics can display (and hence the resolution of the graphies).
- The number of colours available at any one time on the screen.
- The amount of memory left for programs.

A table giving details of these is listed below.

Mode	No of characters	No of graphics pixels	No of colours	Memory used
0	80 × 32	640 × 256	2	20K
1	40 × 32	320 × 256	4	20K
2	20 × 32	160 × 256	16	20K
3	80 × 25	(text only)	2	16K
4	40 × 32	320 × 256	2	10K
5	20 × 32	160 × 256	4	10K
6	40 × 25	(text only)	2	8K

If you don't understand the 'memory used' column then don't worry – basically the more detail and colours available in the mode, the more memory the screen uses and the less there is available for programs. The word 'colour' is used rather loosely to include the flashing colour effects.

Try the different modes out to see the differences. Modes 3 and 6 are for text only – no graphics can be done in these modes (nothing will actually go wrong – it just won't appear).

Why have modes? Different programs have different requirements – some just need simple text output and mode 6 then leaves free as much memory as possible for the program. Others, such as games, need lots of colours and graphics detail. The modes available give a good range across this spectrum.

Writing text

The COLOUR command and text windows

When a letter is written to the screen it has foreground and background colours – the colours of the ink and the paper. When the machine is turned on, it is always white foreground on black background. Colours (or more strictly logical colours – see the section on the palette) are labelled from 8 upwards. To take a definite mode for simplicity, mode I has four colours labelled from 8 to 3. Try the following:

```
MODE 1 RETURN
```

This puts you in mode I with white text on black background.

COLOUR 1 RETURN

This sets the foreground colour to number 1 (red). Text after this command is red on black.

COLOUR 130 RETURN

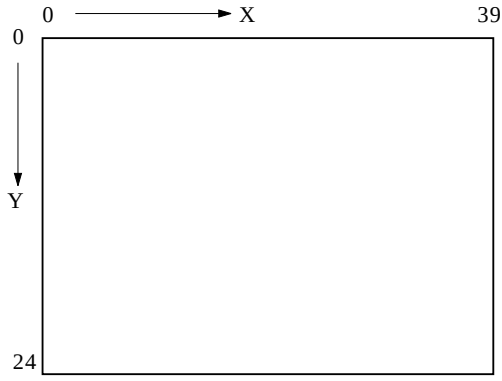
This sets the background colour to number 2 (yellow). Text after this is now red on yellow, and clearing the screen with **CLS** makes the entire screen yellow. Why 130? Because to change the background instead of the foreground colour you must add 128 to the colour number. Thus, to get background colour 2 (above), add 128 to give 136.

Changing mode resets the colours to white on black. As said before, any **VDU** commands (including **COLOUR**, **GCOL**, **MOVE**, **DRAW** etc) can be either typed straight (as a 'direct command') or used as part of a BASIC program.

Addresses on the text screen

Each letter position has its own address in the usual columns and rows format. The column numbering is from left to right starting from column 0 and the rows, as for all **VDU**s, are labelled from the top (row 8) downwards. How many rows and columns there are depends on the mode – the drawing below shows the labelling for mode 6.

The text screen for mode 6



The cursor may be positioned to any part of the screen with the **TAB(X,Y)** command, thus the following program prints out a diagonal line of As.

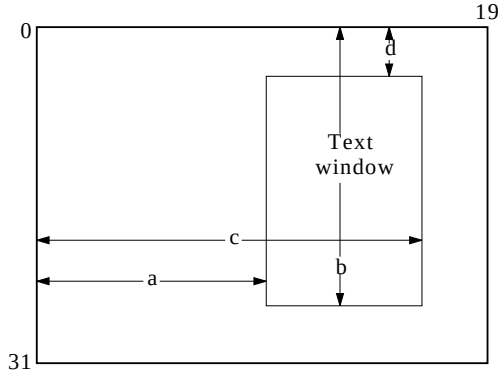
```
10 MODE 1
20 FOR I%=0 TO 20
30 REM The next line positions the text cur
sor to the position col.=I%+5 row=I%
40 REM  and prints the letter A at this pos
ition
50 PRINT TAB(I%+5,I%);"A"
60 NEXT I%
70 END
```

Text windows

Normally, the text may appear anywhere on the screen. However a text window may be set, which allows the text to appear only inside the window. To do this, the **VDU 28** command is used as follows:

VDU 28,a,b,c,d

where a,b is the bottom left and c,d the top right position inside the window (see the drawing below).



Nothing outside the text window is affected by text commands, such as screen clearing, scrolling, cursor positioning etc. Note that the **TAB(X.Y)** measures from the position of the top left of the current window. Try the following program

```
10 MODE 1
20 REM Set up a text window only 6 characte
rs square
30 VDU 28,5,10,10,5
40 REM Change the background colour to
colour 1 (red)
50 COLOUR 129
60 REM Now clear the text screen to red to
see where it is
70 CLS
80 REM Demonstrate scrolling
90 FOR I%=1 TO 20 : PRINT I% : NEXT I%
100 REM lastly, show position of character
(2,2) relative to text window
110 PRINT TAB(2,2);"*"
120 END
```

Both text and graphics windows are removed by **VDU 26**.

Defining your own characters

Each character is an 8 by 8 matrix of dots (pixels). All the normal letters, numbers and punctuation marks are defined, but it is possible to define your own. 256 bytes of RAM are set aside fob the definitions of characters whose codes are from 224 to 255. Character definitions are entered thus:

VDU 23,CODE,L1,L2,L3,L4,L5,L6,L7,L8

where CODE is the code of the character to be defined (it is then printed using either **VDU CODE** or **PRINT CHR$(CODE);**) and

L1 is the bit pattern of the top row
L2 is the bit pattern of the second row from top, and so on until . . .
L8 is the bit pattern of the bottom row.

What is a bit pattern? Each dot in any one row is given a number, and the bit pattern is the sum of the numbers corresponding to those bits in foreground. These numbers, labelling the bits from left to fight, are 128 (for the leftmost pixel), 64, 32, 16, 8, 4, 2, 1 (for the rightmost pixel). Specific examples are easiest to understand.

The space character obviously has no foreground, thus all the bit patterns are zero, so to assign the space character to the code of 224, the command

```
VDU 23,224,0,0,0,0,0,0,0,0
```

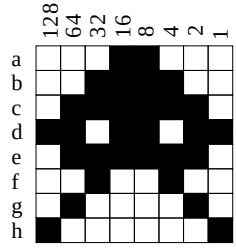
would be used. To define a large X, the top line has the left and rightmost pixels set only, thus L1 = 128+1=129. The next line has the second from left and the second from fight pixels set, thus L2=64+2=66. Similarly, L3=32+4=36 and L4=16+8=24. The fifth through to eighth rows are the mirror image of the first four, so to define the character 225 as an X, type the following fine:

```
VDU 23,225,129,66,36,24,24,36,66,129 RETURN
```

To display the character, type **VDU 225** **RETURN**

All the characters from 32 to 255 may be defined, but to define those outside the codes 224-255 it is necessary to allocate more memory for the fount. This is called ‘exploding the fount’ and is done via **FX** call number 20.

Here is another example of defining a character. The alien in the BUGZAP program on the Introductory Cassette was made up on the matrix in the drawing below.

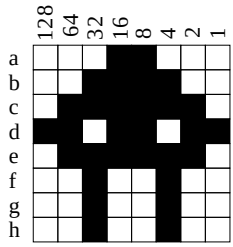


16 + 8 = 24
32 + 16 + 8 + 4 = 60
64 + 32 + 16 + 8 + 4 + 2 = 126
128 + 64 + 16 + 8 + 2 + 1 = 219
64 + 32 + 16 + 8 + 4 + 2 = 126
32 + 4 = 36
64 + 2 = 66
128 + 1 = 129

If you use the code 224 for the new character definition, here is the **VDU** statement which defines the complete character:

```
VDU 23,224,24,60,126,219,126,36,66,129 RETURN
```

By changing L7 and L8, the ‘upright’ alien shown in the drawing below can be defined. The code for this character must be a different one from the one above (eg 225), otherwise you will lose the original alien.



Have a go at defining the new character, then check the result by displaying it on the screen with

VDU 225 RETURN

The program below shows how you can produce an animated alien by using both these characters

```
10 VDU 23,224,24,60,126,219,126,36,66,129
20 VDU 23,225,24,60,126,219,126,36,36,36
30 MODE 2
40 VDU 23,1,0;0;0;0;
50 REPEAT
60 PRINT TAB(10,16);CHR$(224)
70 NOW% = TIME : REPEAT UNTIL TIME = NOW%+25
80 PRINT TAB(10,16); CHR$(225)
90 NOW% = TIME : REPEAT UNTIL TIME = NOW%+25
120 UNTIL FALSE
```

Line 40 gets rid of the flashing cursor, which would otherwise be a distraction. You can retrieve it by typing

VDU 23,1,1;0;0;0 RETURN

Graphics

Introduction

The graphics instructions are pretty extensive in the Electron, and they all have certain things in common. The easiest commands to understand are the **MOVE** and **DRAW** commands, and these will be used for illustration in the following section. The ideas presented here are true for all graphics commands (including **CLG**).

REMEMBER: when you press **BREAK**, the computer is in mode 6. This is not a graphics mode and nothing will happen when you try to plot things. Always remember to go into a graphics mode to try these things out. Mode 1 is a good one to start with. Similarly, programs should always have a **MODE** command in them, as described at the beginning of this chapter.

The graphics coordinate system

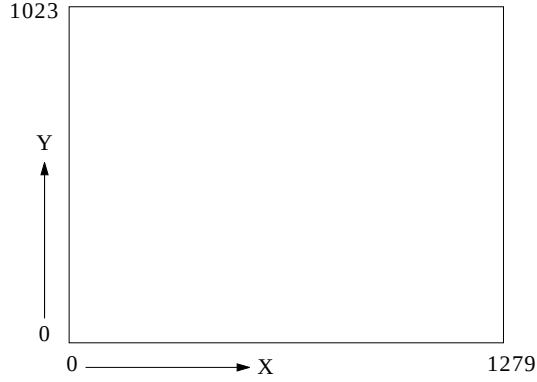
Firstly, we must describe the coordinate system, that is to say how positions of points are labelled. This is similar to the text coordinate system but there are three differences

Firstly the system has the point **(0,0)** in the bottom left hand comer, and row numbers are labelled upwards.

Secondly, the top fight hand point on the screen is **(1279,1023)**, the same in all modes (see the drawing below). This is so that drawing a line from, say **(100,100)** to **(400,400)** always draws a line in the same place, even though the pixel size varies with the mode.

Lastly, points off the screen are well defined, that is to say, drawing a line from, say **(-300,-400)** to **(300,400)** is perfectly legal, and what appears is what you would expect – that portion of the line that is in the area viewed by the screen.

The graphics screen



The graphics cursor is an invisible point on the screen, and is where you are about to draw from. Move it about the screen with the **MOVE** command, and drawing is easiest with the **DRAW** command. Thus

```
MOVE 100,100 RETURN
DRAW 400,400 RETURN
```

moves the cursor to (100,100) and draws a white line to **(400,400)**. Try lots of lines in different modes to get the feel of the coordinate system.

The GCOL command

Just as the foreground and background colours of text were defined using the **COLOUR** command, so the corresponding colours in graphics are defined using the **GCOL** command. Try the following:

```
MODE 1 RETURN
GCOL 0,1 RETURN
DRAW 300,300 RETURN
```

This draws a line in colour 1 (red) from **(0,0)** – where the graphics cursor is when the mode is changed – to **(300,300)**. However, you will notice that the **GCOL** command has two numbers after it. The second is just like the **COLOUR** command's number, that is the foreground colour number, or, if 128 is added to it, the background colour number. **CLG** is the graphics equivalent of **CLS** and clears the graphics area to the current graphics background colour. Thus

```
GCOL 0,129 RETURN
CLG  RETURN
```

sets the graphics background colour to 1 and clears the graphics screen to this colour (red). Note that the **CLG** command is much slower than the **CLS** command.

The first number in the **GCOL** command is unusual. It tells the computer what to do with the graphics point. The following values are defined:

0 – write the point to the screen (what one would normally expect).
1 – OR the point to be plotted with what is on the screen.
2 – AND the point to .be plotted with what is on the screen.
3 – EOR the point to be plotted with what is on the screen.
4 – INVERT what is on the screen, regardless of what colour is to be plotted.
5 – leave what is on the screen alone.

Other values do stripey things which may change with different releases of the software.

What is meant by OR, AND, EOR and INVERT? Each pixel has a colour — in mode 1 with four colours, this is from 0 to 3, or 00, 01, 10 and 11 in binary. What appears on the screen is the result of a logical operation between the pixel you want to plot and what is already on the screen. The OR and the AND are the same as for the BASIC conymands. EOR means 'exclusive OR', which is the same as OR unless both bits are one, in which case the result is zero. Again, all this is most easily explained by specific examples. The following assumes that you are in mode 1.

Assume there is a red screen (from **GCOL0,129:CLG**, above). Setting **GCOL 1,2** sets the foreground colour to 2 (10 in binary), and the colour 'mode' to OR. Drawing a line then takes the red pixel on the screen (red colour 1 = 01 in binary), and ORs it with the yellow pixels you are plotting. The pixel colour that appears is the 01 ORed with 10, which is 11, colour 3, which is white. (Try it).

Given this white line on a red background, set the colour with **GCOL 2,2**, which has foreground colour 2 and colour mode AND. Plotting a line then takes what is on the screen and ANDs it with the yellow pixel, colour 2 or 10 in binary. Therefore when the line is plotting on the red background, 01 (red) is ANDed with 10 (yellow), then result being 00 (black). If it crosses the white line 0 the white pixels (11) are ANDed with the yellow pixels (10) to give 10 (yellow).

Setting **GCOL 3,131** sets the background colour to 3 (white) and the colour mode to EOR. Doing a **CLG** then EORs its pixel with 11 – that is to say 00 goes to 11, 01 to 10, 10 to 01 and 11 to 00. The screen is thus inverted in colour, and repeating the command restores it to its original state.

If this does not seem too clear, playing around with it for a little should help. It has two main purposes – setting the EOR mode allows erasure of a line by plotting it again. In four colour modes, two independent two colour pictures may be drawn and selectively displayed using the palette.

The PLOT command

MOVE and DRAW are two special cases of the more general PLOT command, which is as follows.

PLOT K%,X%,Y%

where **K%** is the plot mode (ie what you are actually going to do); **X%** and **Y%** are the coordinates of the point to which you are plotting.

K% takes the following values:

0	Draw a line, relative (that is **X%** and **Y%** are displacements from the current graphics cursor position), with no change on the screen.
1	As 0, but draw in foreground colour.
2	As 0, but invert what is on the screen (colour mode 4 forced).
3	As 0, but draw in background colour.
4 to 7	As 0 to 3 but plot absolute (plot to the point **X%,Y%**).
8 to 15	As 0 to 7, but plot the last point twice. This is so that when plotting in inverted modes, the line is continuous.
16 to 31	As 8 to 15 but with a dotted line.
32 to 63	Reserved for the graphics extension ROM.
64 to 71	As 0 to 7, but plot the specified point only.
72 to 79	Fill sideways on background colour (see below).
80 to 87	Plot triangles (see below).
88 to 95	Fill right on non-background colour (see below).
96 to 255	Reserved for graphics extension ROM.

Advanced graphics

Triangle plotting

This plots a solid triangle using as vertices the point specified, the graphics cursor and the previous graphics cursor. This can be used to fill many different shapes.

Sideways filling on background colour

This plots the line sideways from the specified point, left and right, until either the edge of the window is reached or the line meets a pixel of nonbackground colour. The graphics cursor is set to one of the end points, and the previous graphics cursor (used in triangle plotting) to the other.

The values of these may be found out via **OSWORD** call number 13 (decimal). If the point specified is outside the graphics window, or is not on background colour, then the Y coordinates of the returned points are different.

Filling right
Filling fight until background colour plots fight from the specified point as far as either the edge of the graphics window or a pixel of background colour is found. The endpoints (note that it does not go left) are retrievable via **OSWORD 13** in the same way. The endpoint is actually the first pixel of background colour found, thus if the specified point is background colour, the endpoint returned is the same as the specified one.

These two routines together enable a fast routine to be constructed to fill any enclosed shape.

The VDU command
The **VDU** command writes a series of bytes to the screen in a similar way to the **PRINT** command. Thus the following two commands are exactly the same:

```
VDU 12,65,66,67
```

```
PRINT CHR$(12); CHR$(65); CHR$(66); CHR$(67
);
```

(Note the semicolon at the end of the **PRINT** statement – the **VDU** command does not send a carriage return unless you explicitly tell it to). Most numbers that you need to write to the **VDU** are single bytes (characters, for example). However, the graphics coordinates are all double byte quantities and are sent lower byte first, higher byte second. The **VDU** command enables this to be done easily. If a number in the **VDU** command is followed by a semicolon, that number is interpreted as a double byte quantity. If you are unsure of bytes and double bytes, the quick rule is that if you are doing a graphics operation using the **VDU** command, you must always follow a graphics coordinate with a semicolon. Thfs only applies to the **VDU** command.

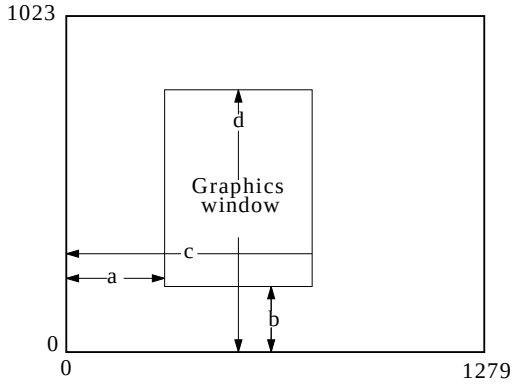
There are two more graphics commands, both of which are done via the VDU command.

Graphics windows

Just as text may have a text window defined, outside of which no text command has effect, so a graphics window may be similarly defined. This is done with

```
VDU 24,L%;B%;R%;T%;
```

where **L%,B%** and **R%,T%** are the coordinates of the lower left and upper right pixels inside the window. Setting a window thus prevents any plotting outside it. Also, because **CLG** is just another plot command, defining a graphics window and doing a **CLG** is a quick way of plotting rectangles.



The graphics origin

So far it has been said that the point **(0,0)** is at the bottom left hand comer of the screen. This point (called the origin) may be specified to five elsewhere with the origin command.

```
VDU 29, X%; Y%;
```

sets the position of the origin on the screen for future graphics commands. Thus to set the origin in the middle of the screen, use **VDU 29,640;512;**. It does not move the physical position of what is on the screen, the graphics windows or the graphics cursor.

Plotting characters

If **VDU 5** is entered, the text and graphics cursors are said to be joined, that is text appears at the graphics cursor which then moves as the text is written. This is mainly used for labelling graphs. The graphics cursor points to the top right pixel of the 8 by 8 character cell to be written, and is moved 8 pixels along by writing letters. This is seen in the following program.

```
10 MODE 1
20 VDU 5
30 REM All the text now appearing is 'plott
ed'
40 DRAW 500,500
50 PRINT "Hello mummy";
60 REM the last print statement moved the g
raphics cursor
70 REM   as can be seen by the next plot st
atement.
80 DRAW 0,0
90 END
```

VDU4 restores the text cursor.

The palette

Colours defined through the **COLOUR** and **GCOL** commands are more properly referred to as logical colours. When a mode is changed, these logical colours appear as certain physical colours, thus in mode 1, colour 1 is red and colour 2 is yellow. The palette allows this to be changed, thus colour 1 may be made to be blue and colour 2 flashing black on white. To be exact, we must distinguish between two types of colour:

The logical colour is what is output by the **COLOUR** commands. The maximum logical colour is limited by the number of colours available in the mode.

The physical colour is what appears on the screen. The physical colours and their numbers are fisted below.

Physical number	Display colour
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White
8	Flashing black/white
9	Flashing red/cyan
10	Flashing green/magenta
11	Flashing yellow/blue
12	Flashing blue/yellow
13	Flashing magenta/green
14	Flashing cyan/red
15	Flashing white/black

Each logical colour has a physical colour assigned to it, which may be changed by reprogramming the palette. This is done as follows:

```
VDU 19,L%,P%,0,0,0
```

where **L%** is the logical colour and **P%** is the physical colour. So in mode 1, to change all the pixels with logical colour 3 (usually white) to blue (logical colour 4), the command **VDU 19,3,4,0,0,0** is used. Thus while the very detailed mode 0 is a two colour mode, the colours themselves may be anything available from the palette, such as green on red. Note also that the palette reprogramming is very fast as it does not involve a lot of the screen memory being reprogrammed.