ACORNSOFT

GXR

The Graphics Extension ROM for the BBC Microcomputer

User Guide

ACORNSOFT

GXR

The Graphics Extension ROM for the BBC Microcomputer and Acorn Electron

ISBN 0 907876 28 5

First published in 1985 by Acornsoft Limited

Note: Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

FIRST EDITION
Second revision, January 2021

Printed in the United Kingdom

Acornsoft Limited, Cambridge Technopark, 645 Newmarket Road,
Cambridge CBS 8PD

Contents

Introduction

The Graphics Extension ROM (GXR) is an extension to the BBC Microcomputer and Acorn Electron operating systems. Its facilities can be divided into two parts. Half of it is devoted to providing enhanced shape plotting by using previously unassigned `PLOT` codes, `VDU` codes and `GCOL` statements. For example it uses the plot codes 144–151 to draw circle outlines and `VDU23,2` to `5` to set up colour patterns which may be used to fill triangles etc or even flood an enclosed area of the screen. These commands greatly increase the graphic capabilities of the machines.

The second half of the ROM provides a 'sprite' package. This consists of a sprite editor which is used to define or alter sprite shapes which can then be saved on tape or disc. These shapes may be used later as part of programs and can be plotted anywhere on the screen using a single `PLOT` statement. This provides a method of writing arcade style games in high level languages such as BASIC. Previously this was not particularly successful because the slower speed of the high level languages compared with machine code meant that any shapes could only be moved very slowly and unsteadily.

Whilst the Graphics Extension ROM is active these new plot commands can be accessed in exactly the same way as the operating system plot codes and from any language which supports graphics features. Although the examples in this book are all in BBC BASIC the methods used are equally applicable to other languages such as COMAL or Logo. The Graphics ROM is also compatible with all the current Acorn Second Processors.

Various examples and utilities are provided on the cassette which accompanies this pack. Disc users can make a copy of this cassette on to a disc using the INSTALL program supplied. Type

`*TAPE` **RETURN**
`CHAIN "INSTALL"` **RETURN**

with a blank formatted disc in the current disc drive and the copying will be performed automatically. You will be prompted to turn over the tape when necessary.

1 Getting Started

1.1 Fitting the Graphics Extension ROM (in a BBC Microcomputer)

Your Graphics Extension ROM may be placed in any spare ‘sideways ROM’ socket. These are located on the front right-hand side of the circuit board inside the BBC Microcomputer casing.

1. To get to the board ensure that the computer is switched off and then undo the four large screws normally marked ‘FIX’. Two of these are underneath the computer, and the other two can be found on the back.

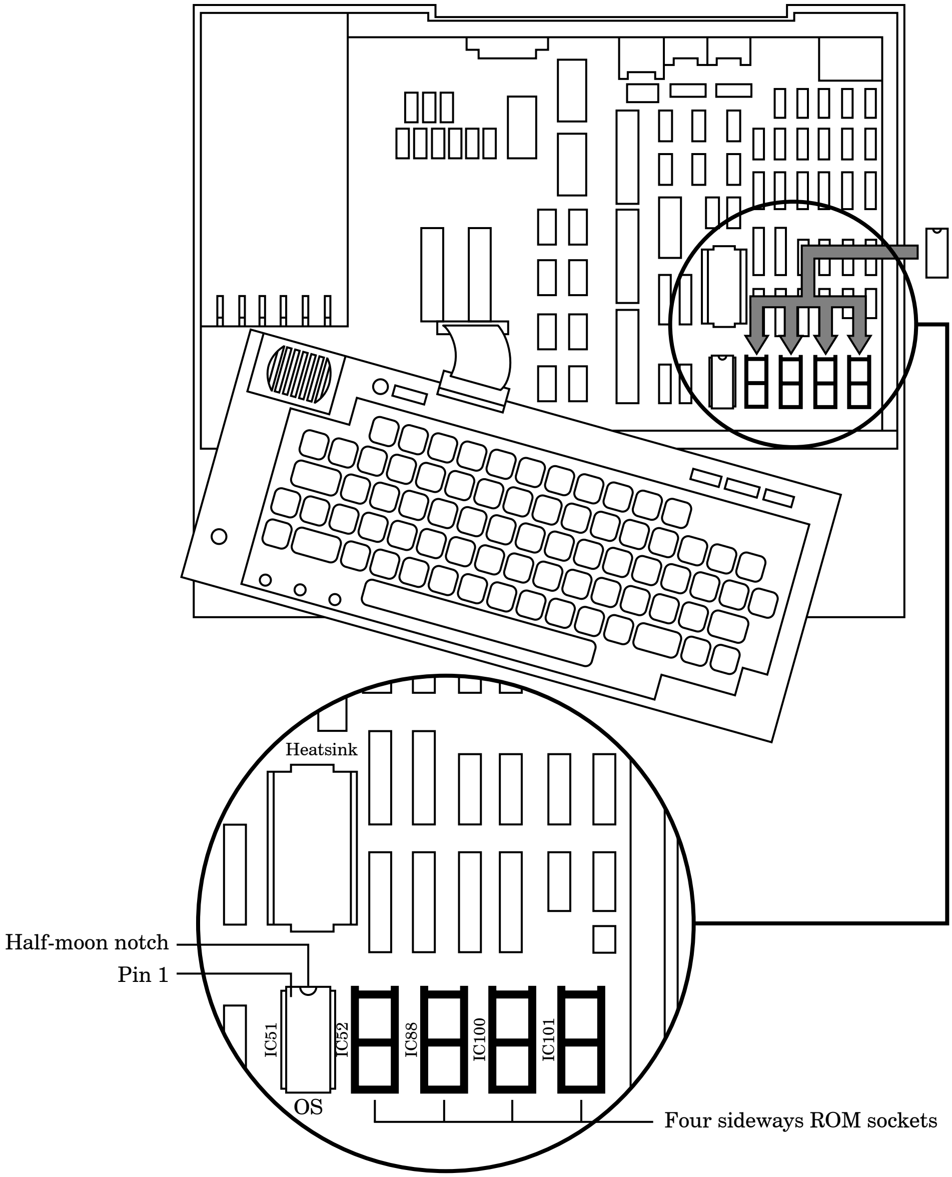
2. Once the top is removed, release the bolts holding down the keyboard assembly. These are located on either side of the keyboard. Some machines have two bolts, others may have three.

3. There is no need to disconnect the keyboard completely, so the multi-wire connector to the main board can be left in place. Carefully displace the keyboard, rotating it clockwise through about 20 degrees so that the front right-hand side is accessible.

4. Locate the row of five large sockets (see diagram). The four right-hand sockets (identified on the board as IC52, IC88, IC100, IC101) are sideways ROM sockets. The fifth from the right is the operating system socket (IC51). If you have either the Aries B20 ROM or a DFS (or DNFS) ROM already fitted then you should place the Graphics ROM to the left of these ROMs, otherwise you can place it in any spare slot. However, note the effect of putting it in different sockets described in section 1.3.

5. Before taking the ROM out of its protective packaging, identify Pin 1 on the ROM. It is either marked with a dot on the top, in the corner of Pin 1, or with a half-moon notch at one end of the ROM identifying the end nearest Pin 1. Pin 1 is on the left if the notch is held upwards.

6. Hold the ends of the ROM between finger and thumb, and line up all the pins over the destination socket. Pin 1 and the half-moon notch should point towards the back of the computer casing. Now apply firm pressure to the ROM, but try not to force it! When the ROM is in, it appears to be slightly raised. Check that all the pins do enter the socket, and that none are bent out, or underneath.


Heatsink
Half-moon notch
Pin 1
IC51
IC52
IC88
IC100
IC101
OS
Four sideways ROM sockets

1.2 Fitting the ROM Cartridge (in an Acorn Electron Plus 1)

The Graphics Extension ROM Cartridge may be fitted in either of the cartridge slots on your Plus 1. Ensure that the power is disconnected from your Acorn Electron and then gently insert the Cartridge into one of the slots, with the label on the cartridge towards the front of the machine. The power may then be reconnected and the Graphics Extension ROM Cartridge will be ready for use. See section 1.3 regarding the different effects of putting the Cartridge in the two sockets.

1.3 Using the ROM or ROM Cartridge

It is possible to fit the Graphics Extension ROM in your computer either so that it is immediately active when the machine is switched on (or hard reset),
or so that it needs to be explicitly turned on before being used. Which of these situations occurs is governed by the 'priority' of the sideways ROM socket in which the ROM is fitted. If an odd numbered socket is used (which on an unexpanded machine means the rightmost socket or the third from the right) then the default will be that the Graphics Extension ROM is switched on and three pages of workspace claimed (ie PAGE is increased by &300). If the ROM is fitted in an even numbered socket then the ROM will not be active and no workspace will be claimed until it is explicitly turned on. In an Acorn Electron the default is for the ROM Cartridge to be active if it is in the front socket on the Plus 1 and inactive if it is in the rear socket.

If the Graphics Extension ROM is not active (either by default in an even priority socket or because it has been turned off) then it will need to be activated before any of the features can be used. This is done by typing:

`*GXR` **RETURN**

You will then be asked to press the **BREAK** key.

When fully activated the Graphics Extension ROM claims three pages of work space. The value of PAGE will increase by &300 which could prevent some of your current software from running because it may not have sufficient room. If this is the case (or if you wish to deactivate the ROM for any other reason) then you can turn the ROM off by typing

`*NOGXR` **RETURN**

and then pressing **BREAK**.

Two of the three pages claimed are used purely for the flood fill routines. If you wish you can turn off the flood fill whilst having the rest of the graphics routines active and hence only increase the value of PAGE by &100. To do this type

`*NOFLOOD` **RETURN**

The flood fill routines can be reactivated by the command

`*FLOOD` **RETURN**

The Graphics Extension ROM will be reset to its default state if you perform a hard reset (**CTRL BREAK**).

1.4 Using the example programs

The following chapters describe in detail all the new features which the Graphics Extension ROM provides. These are illustrated by example programs which are listed and are also contained on the tape provided in this pack. Each program has its name on the first line of the listing and can be run by typing, for example

`CHAIN "SQUARES"` **RETURN**

Most of the example programs loop indefinitely and it is necessary to use the ESCAPE key to leave the program. In addition the tape contains some other larger programs which demonstrate the effects which may be achieved by combining the features of the Graphics Extension ROM. Listings of these programs are not given in this User Guide but details of them can be found in chapter 6, ‘Graphics examples and utilities’.

2 Solid shapes and outlines

The operating systems of the BBC Microcomputer and Acorn Electron provide routines to draw straight lines and to plot triangles. Any other outlines or shapes must be drawn using combinations of these. For simple shapes such as squares this is an easy task, however problems arise for shapes such as circles since routines to draw them are either inaccurate or very slow. The Graphics Extension ROM solves these problems by providing fast and simple to use routines for plotting some of the most commonly used outlines and solid shapes. These routines are described below.

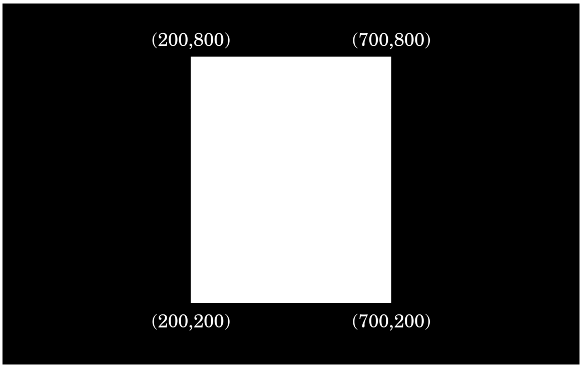
Note that the codes for these `PLOT` commands are given in groups of eight. This is the same as for the standard `PLOT` commands supplied in the operating system and governs how the coordinates are interpreted (relative or absolute) and what colour is used for the plot. The meanings of the codes within each group are as follows (see the *BBC Microcomputer System User Guide* or the *Acorn Electron User Guide* for further details):

`0`	move relative to last point (without drawing on screen)
`1`	plot relative in current graphics foreground colour
`2`	plot relative in logical inverse colour
`3`	plot relative in current graphics background colour
`4`	move to absolute position (without drawing on screen)
`5`	plot absolute in current graphics foreground colour
`6`	plot absolute in logical inverse colour
`7`	plot absolute in current graphics background colour

(The exception to this is the block `&B8`–`&BF` which is detailed in chapter 5.)

2.1 Rectangles and squares

Consider the following rectangle:



Normally to draw this you would plot two triangles using the following steps:

```
MOVE 700,200          :REM  bottom right-hand corner
MOVE 200,200          :REM  bottom left-hand corner
PLOT &55,700,800      :REM  top right-hand corner
PLOT &55,200,800      :REM  top left-hand corner
```

Although this takes four instructions only two are really needed since any axis aligned rectangle can be defined by giving two diagonally opposite corners. Using the Graphics Extension ROM the rectangle given above can be plotted in the following manner:

```
MOVE 200,200          :REM  bottom left-hand corner
PLOT &65,700,800      :REM  top right-hand corner
```

or alternatively

```
MOVE 200,800          :REM  top left-hand corner
PLOT &65,700,200      :REM  bottom right-hand corner
```

etc.

The plot codes for rectangle fills are as follows:

```
&60 - &67  (96 - 103)
```

The following example plots squares (equal-sided rectangles) recursively to produce some very striking multi-coloured patterns

```
0 REM SQUARES
5 ON ERROR VDU23,1,1;0;0;0;:END
10 MODE1
20 VDU23,1,0;0;0;0;
30 recplot=&65
40 REPEAT
50   v%=512
60   REPEAT
70     GCOL3,RND(3)
80     VDU19,RND(3),RND(7);0;
90     PROCsq(500,400,256)
100     v%=v%/2
110     UNTIL v%<16
120   UNTIL FALSE
130
140 DEFPROCsq(x%,y%,s%)
150 IF s%<v% ENDPROC
160 PROCsq(x%+s%/4,y%+s%,s%/2)
170 PROCsq(x%+s%,y%+s%/4,s%/2)
180 PROCsq(x%-s%/2,y%+s%/4,s%/2)
190 PROCsq(x%+s%/4,y%-s%/2,s%/2)
200 VDU29,x%;y%;
210 MOVE0,0
220 PLOTrecplot,s%,s%
230 ENDPROC
```

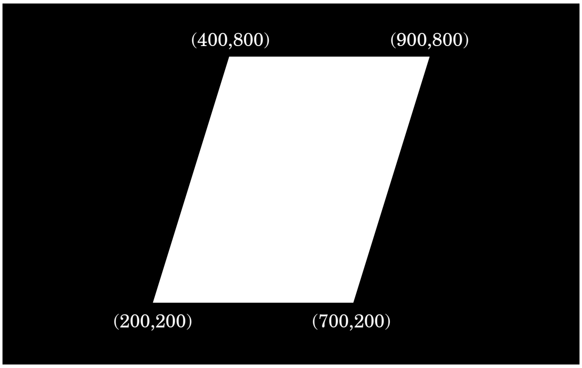
Rectangles can be put to more serious uses than just graphics displays. The following example plots a bar chart showing two sets of figures for each month, these could be the sales figures for a certain item for the last two years, or maybe the average temperatures of two resorts. The example could easily be altered to plot more figures or have different scales.

```
0 REM CHART
10 ON ERROR VDU23,1,1;0;0;0;:END
20 MODE 1
30 VDU23,1,0;0;0;0;
40 recplot=&65
50 DATAJ,F,M,A,M,J,J,A,S,O,N,D
60 DATA29,24,13,26,48,61,61,59,58,52,40,28
70 DATA35,30,31,42,63,74,80,71,64,70,43,32
80 PROCaxis
```

```
90 PROCscales
100 VDU23,1,0;0;0;0;
110 PROCbars(2,5)
120 PROCbars(1,7)
130 REPEATUNTILFALSE
140
150 DEFPROCaxis
160 MOVE128,128 : DRAW128,928
170 MOVE128,128 : DRAW1100,128
180 ENDPROC
190
200 DEFPROCscales
210 VDU5
220 FOR mon%=1 TO 12
230   READmon$
240   VDU24,(5+mon%*5)*16;48;(8+mon%*5)*16;96;
250   CLG
260   MOVE(5+mon%*5)*16,80
270   PRINT;mon$
280   NEXT
290 FOR scale%=10 TO 80 STEP10
300   VDU24,48;scale%*10+96;80;scale%*10+144;
310   CLG
320   MOVE48,scale%*10+128
330   PRINT;scale%
340   NEXT
350 VDU4,26
360 ENDPROC
370
380 DEFPROCbars(col%,x%)
390 GCOL0,col%
400 FOR month%=1 TO 12
410   READvalue%
420   MOVE(x%+month%*5)*16,132
430   PLOT&65,(x%+month%*5)*16+16,128+10*value%
440   NEXT
450 ENDPROC
```

2.2 Parallelograms

Parallelograms are rectangles which have been sheared sideways, for example:



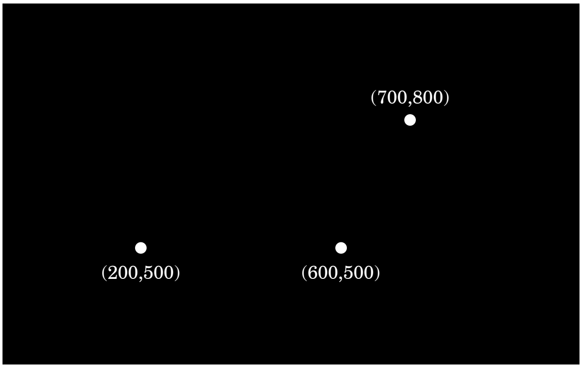
These require three points to define them and have been assigned the following plot codes:

```
&70 - &77  (112 - 119)
```

Thus to plot the parallelogram shown above the following could be used:

```
MOVE 200,200           :REM bottom Left
MOVE 700,200           :REM bottom right
PLOT &75,900,800       :REM top right
```

Although any three corners of the parallelogram may be used to define it, the order in which these are given will affect which way round the parallelogram appears. Consider the three points given below:



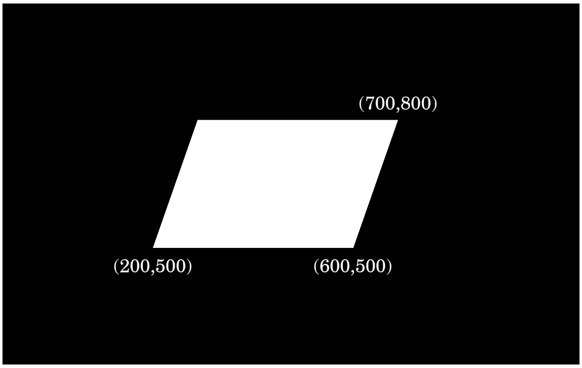
These could produce any of the following three parallelograms, depending on the order they were used:

1)

```
MOVE 200,500          :REM bottom left
MOVE 600,500          :REM bottom right
PLOT &75,700,800      :REM top right
```

or

```
MOVE 700,800          :REM top right
MOVE 600,500          :REM bottom right
PLOT &75,200,500      :REM bottom left
```

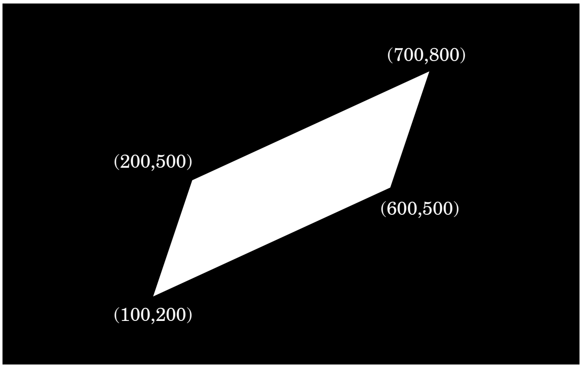


2)

```
MOVE 200,500          :REM top left
MOVE 700,800          :REM top right
PLOT &75,600,500      :REM bottom right
```

or

```
MOVE 600,500          :REM bottom right
MOVE 700,800          :REM top right
PLOT &75,200,500      :REM top left
```

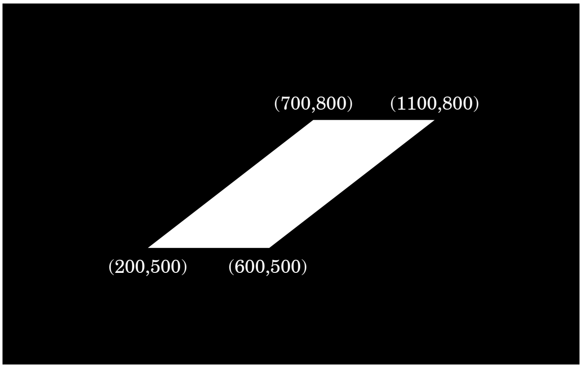


3)

```
MOVE 600,500          :REM bottom right
MOVE 200,500          :REM bottom left
PLOT &75,700,800      :REM top left
```

or

```
MOVE 700,800          :REM top left
MOVE 200,500          :REM bottom left
PLOT &75,600,500      :REM bottom right
```



The points are taken as moving around the edge of the parallelogram in sequence, the final point being opposite the middle one defined.

The following example uses parallelograms with all four sides the same size to produce a simple tiling pattern.

```
0 REM TILES
10 ON ERROR VDU23,1,1;0;0;0;:END
20 MODE1
30 VDU23,1,0;0;0;0;
40 parplot=&75
50 DATA8,0,8,8,0,8,-8,8,-8,0,0,0
60 PROCcentre(640,512,100,3)
70 PROCcentre(972,512,100,1)
80 PROCcentre(804,796,100,2)
90 PROCcentre(476,796,100,1)
100 PROCcentre(302,512,100,2)
110 PROCcentre(476,228,100,1)
120 PROCcentre(804,228,100,2)
130 REPEATUNTILFALSE
140
150 DEFPROCcentre(xc%,yc%,size%,col%)
160 GCOL0,col%
170 RESTORE 50
180 FOR n%=0 TO 5
190   READaddx%,addy%
200   x%=xc%+addx% : y%=yc%+addy%
210   ang1=(n%-1)*PI/3 : ang2=n%*PI/3
220   MOVEx%+size%*COS(ang1),y%+size%*SIN(ang1)
230   MOVEx%,y%
240   PLOTparplot,x%+size%*COS(ang2),y%+size%*SIN(ang2)
250   NEXT
260 ENDPROC
```

2.3 Circle fills

A circle is defined by two points: its centre and any point on the circumference. To plot a circle you should initially visit the point you want as its centre and then use `PLOT` with the relevant plot code and the coordinates of a point on its circumference. For example to plot a circle in the centre of the screen with radius of 100 type

```
MOVE 640,512         :REM centre
PLOT &9D,740,512     :REM Xcentre+radius,Ycentre
```

or alternatively you could use relative plotting :

```
MOVE 640,512        :REM centre
PLOT &99,100,0      :REM radius,0
```

The plot codes for solid circles are as follows:

```
&98 - &9F  (152 - 159)
```

The following short program draws solid circles in random colours at random positions on the screen.

```
0 REM CIRCLES
5 ON ERROR VDU23,1,1;0;0;0;:END
10 MODE2
20 VDU23,1,0;0;0;0;
30 cirplot=&9D
40 REPEAT
50   GCOL0,RND(7)
60   x%=RND(1280)
70   y%=RND(1024)
80   r%=RND(350)+50
90   MOVE x%,y%
100   PLOTcirplot,x%+r%,y%
110   UNTIL FALSE
```

2.4 Circle outlines

Circle outlines can be drawn in a similar way. The plot codes for these are as follows:

```
&90 - &97  (144 - 151)
```

The example given below shows circle outlines being drawn spiralling outwards from a centre point.

```
0 REM SPIRAL
5 ON ERROR VDU23,1,1;0;0;0;:END
10 MODE1
20 VDU23,1,0;0;0;0;
30 cirdraw=&95
40 VDU19,2,2,0,0,0,19,3,3,0,0,0
50 FORN%=1TO45
60   GCOL0,N% MOD 3 +1
70   X=640+N%*10*COS(0.25*N%)
80   Y=600+N%*10*SIN(0.25*N%)
90   MOVEX,Y
100   PLOTcirdraw,X,Y+40+2*N%
110   NEXT
120 REPEATUNTILFALSE
```

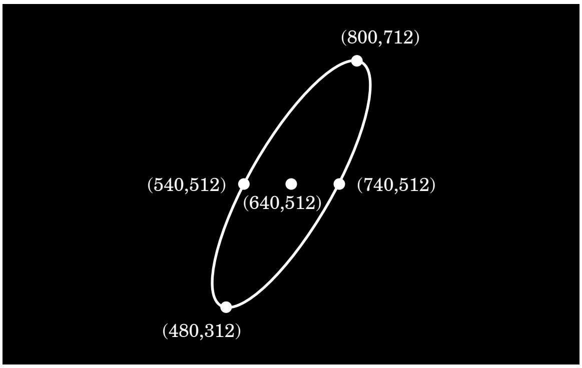
2.5 Ellipse outlines

Ellipses are more complicated to define than circles. The plot codes used for ellipse outlines are as follows:

```
&C0 - &C7  (192 - 199)
```

To plot an ellipse the Graphics Extension ROM requires the following information:

1) the centre point
2) an outermost point (either to the right or left) at the same height as the centre
3) the highest or lowest point of the ellipse



For example to draw the above ellipse you could:

give the centre,

```
MOVE 640,512
```

the right-hand point

```
MOVE 740,512
```

and the top point

```
PLOT &C5,800,712
```

or alternatively, specify the centre, left and bottom points:

```
MOVE 640,512
MOVE 540,512
PLOT &C5,480,312
```

Note that since the second point is taken as being at the same height as the centre the Y value is ignored. However, for clarity, it is best to give the same height as the centre in your programs.

The following example creates patterns using a number of different shaped ellipses.

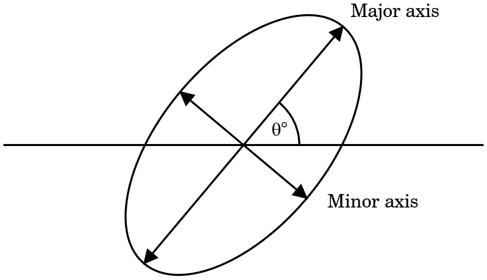
```
0 REM ELLIPSE
5 ON ERROR VDU23,1,1;0;0;0;:END
10 MODE 0
20 VDU23,1,0;0;0;0;
30 elldraw=&C5
40 REPEAT
50   PROCpattern1 : wait=INKEY(200)
60   PROCpattern2 : wait=INKEY(200)
70   PROCpattern3 : wait=INKEY(200)
80   UNTIL FALSE
90
100 DEFPROCpattern1
110 CLS
120 FOR step%=0 TO 400 STEP25
130   MOVE640,512
140   MOVE215+step%,512
150   PLOTelldraw,640,512+step%
160   NEXT
170 ENDPROC
180
190 DEFPROCpattern2
200 CLS
210 FOR step%=-400 TO 400 STEP40
220   MOVE640,512
230   MOVE670,512
240   PLOTelldraw,640+step%,800
250   NEXT
260 ENDPROC
270
280 DEFPROCpattern3
290 CLS
300 FOR step%=-512 TO 512 STEP32
310   MOVE640+step%,512
320   MOVE670+step%,512
330   PLOTelldraw,640+step%,1024-ABS(step%)
340   MOVE640,512+step%
350   MOVE1152-ABS(step%),512+step%
360   PLOTelldraw,640,544+step%
370   NEXT
380 ENDPROC
```

2.6 Ellipse fills

These are defined in a similar way to ellipse outlines, their plot codes are as follows:

```
&C8 - &CF  (200 - 207)
```

For some applications it may be inconvenient to give the details of the ellipse in the form required. The following example includes an algorithm which allows the ellipse to be defined by its centre point, the size of its major and minor axes and the angle by which it is rotated by from the horizontal.



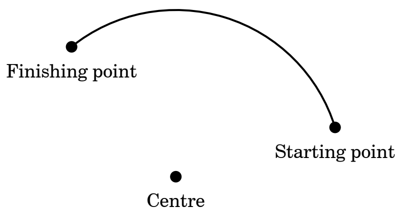
This algorithm is used to plot ellipses of similar size with the same centre point to form eight petalled flowers.

```
0 REM FLOWERS
5 ON ERROR VDU23,1,1;0;0;0;:END
10 MODE1
20 VDU23,1,0;0;0;0;
30 ellplot=&CD
40 cirplot=&9D
50 PROCflower(640,512,100,30)
60 PROCflower(320,256,50,15)
70 PROCflower(320,768,50,15)
80 PROCflower(960,768,50,15)
90 PROCflower(960,256,50,15)
100 REPEATUNTILFALSE
110
120 DEFPROCflower(cx%,cy%,sx%,sy%)
130 PROCpetals(3,0)
140 PROCpetals(2,45)
150 PROCpetals(1,22.5)
```

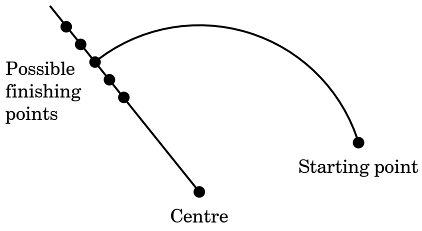
```
160 PROCpetals(1,67.5)
170 GCOL0,2
180 MOVEcx%,cy%
190 PLOTcirplot,cx%+sx%/2,cy%
200 ENDPROC
210
220 DEFPROCpetals(col%,angle)
230 GCOL0,col%
240 FOR n%=0 TO 3
250   x%=sx%*COS(RAD(90*n%+angle))
260   y%=sx%*SIN(RAD(90*n%+angle))
270   PROCell(cx%+x%,cy%+y%,sx%,sy%,90*n%+angle)
280   NEXT
290 ENDPROC
300
310 DEFPROCell(cx%,cy%,maj%,min%,ang%)
320 cosang=COS(RAD(ang%))
330 sinang=SIN(RAD(ang%))
340 maxy%=SQR((min%*cosang)^2+(maj%*sinang)^2)
350 shearx%=(maj%*maj%-min%*min%)*cosang*sinang/maxy%
360 slicew%=maj%*min%/maxy%
370 MOVEcx%,cy%
380 MOVEcx%+slicew%,cy%
390 PLOTellplot,cx%+shearx%,cy%+maxy%
400 ENDPROC
```

2.7 Arcs

We saw above how circle outlines are defined and drawn. In a similar way just a portion of the outline may be drawn to produce an arc. In this case three points are required; the centre of the circle, and two points to indicate the starting and finishing points of the arc, for example:



However in the above example both the starting and finishing points are on the arc itself which requires a large amount of calculation to get right. It is easier for the starting point to be taken as being on the arc and used to calculate the radius, the finishing point being used just to indicate the angle the arc subtends:



This is the method used by the Graphics Extension ROM. To draw an arc you need to visit the centre of the circle that it is based upon, then go to the starting point of the arc and finally PLOT to a third point which indicates the angle. The plot codes to use are:

```
&A0 - &A7  (160 - 167)
```

The following will draw an arc based on a circle whose centre is at 640,512 and whose radius is 200. It will draw the portion of the arc from 0° to 270°. Since arcs are drawn anticlockwise this means that its starting position is the point 440,512 (270°) and its finishing position is 640,712 (0°).

```
MOVE 640,512
MOVE 440,512
PLOT &A5,640,1000    :REM arbitrary y-coord, sets
                          direction only
```

The following example uses arcs and ellipses to produce the curved outline and sails of a sailing ship.

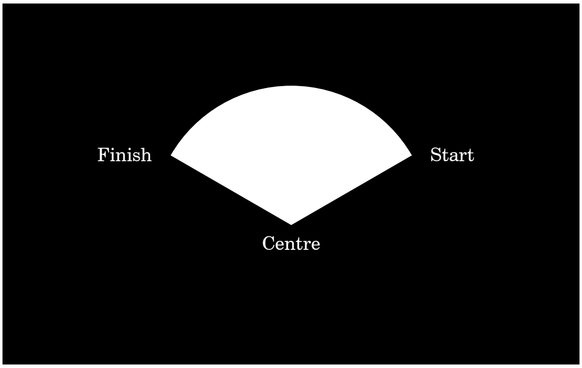
```
0 REM SHIP
5 ON ERROR VDU23,1,1;0;0;0; :END
10 MODE 1
20 VDU23,1,0;0;0;0;
30 arcdraw=&A5
40 recplot=&65
50 ellplot=&CD
```

```
60 PROCship
70 PROCmast
80 REPEATUNTILFALSE
90
100 DEFPROCship
110 FOR n%=1 TO 5
120   MOVEn%*180+140,280
130   MOVEn%*180-20,240
140   PLOTarcdraw,n%*180+40,120
150   NEXT
160 MOVE840,260 : MOVE960,140
170 PLOTarcdraw,1000,240
180 MOVE800,360 : MOVE960,140
190 PLOTarcdraw,1080,220
200 MOVE232,140 : DRAW960,140
210 MOVE168,240 : DRAW1040,240
220 MOVE1000,240 : DRAW1160,300
230 ENDPROC
240
250 DEFPROCmast
260 GCOL0,1
270 MOVE580,240 : DRAW580,800
280 MOVE320,240 : DRAW320,600
290 MOVE840,240 : DRAW840,600
300 MOVE840,600 : DRAW1160,300
310 MOVE840,600 : DRAW920,240
320 MOVE580,748 : DRAW720,240
330 MOVE320,600 : DRAW400,240
340 GCOL0,3
350 PROCsail(800,500,80,80)
360 PROCsail(740,300,160,160)
370 PROCsail(500,560,160,160)
380 PROCsail(480,312,200,200)
390 PROCsail(280,500,80,80)
400 PROCsail(240,300,160,160)
410 GCOL0,1
420 MOVE840,600 : DRAW760,240
430 MOVE580,748 : DRAW440,240
440 MOVE320,600 : DRAW240,240
450 GCOL0,3
460 PROCnest
470 ENDPROC
480
```

```
490 DEFPROCsail(x1%,y1%,xs%,ys%)
500 MOVEx1%,y1%
510 PLOTrecplot,x1%+xs%,y1%+ys%
520 GCOL0,0
530 MOVEx1%,y1%+ys%/2
540 MOVEx1%+xs%/4,y1%+ys%/2
550 PLOTellplot,x1%,y1%+ys%
560 GCOL0,3
570 MOVEx1%+xs%,y1%+ys%/2
580 MOVEx1%+xs%*5/4,y1%+ys%/2
590 PLOTellplot,x1%+xs%,y1%+ys%
600 ENDPROC
610
620 DEFPROCnest
630 MOVE540,760 : DRAW620,760
640 FOR n%=4 TO 20 STEP4
650   MOVE580,780
660   MOVE580-n%,780
670   PLOTarcdraw,580+n%,780
680   NEXT
690 ENDPROC
```

2.8 Sectors

Sectors are solid shapes which are portions of circles, for example:



The plot codes they have been assigned are as follows:

```
&B0 - &B7  (176 - 183)
```

Sectors are defined in a similar manner to arcs, for example:

```
MOVE 640,512          :REM centre point
MOVE 440,512          :REM starting point on the
                           circumference
PLOT &B5,640,1000     :REM point indicating angle of
                           sector
```

Again the sector is taken as going anti-clockwise from the starting point to the finishing point.

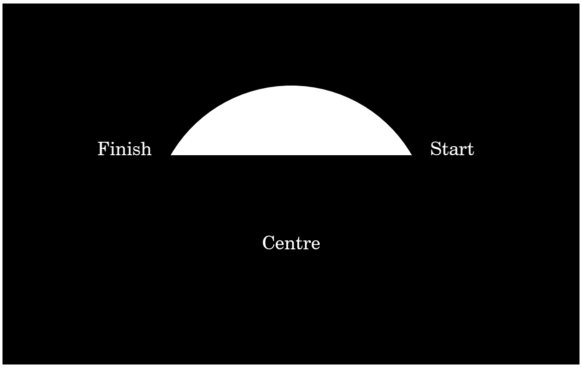
An important use of sectors is for piecharts. The following program allows you to enter up to 12 numbers and will then display them in their correct ratios in pie-chart form. Note that values which are less than or equal to zero are invalid.

```
0 REM PIE
5 ON ERROR VDU23,1,1;0;0;0;: END
10 MODE1
30 secplot=&B5
40 DIMval(12)
50 cx%=640 : cy%=512
60 total%=0 : oldpc=0
70 rad%=404
80 PROCinput
90 PROCpie
100 REPEATUNTILFALSE
110
120 DEFPROCinput
130 REPEAT
140   INPUT"How many items of data do you have :"num%
150   IF num%<1 OR num%>12 PROCinvalid : UNTIL FALSE ELSE UNTIL TRUE
160 FOR count%=1 TO num%
170   PRINT'count%;
180   INPUT" : "val(count%)
190   IF val(count%)<=0 VDU7 : GOTO170
200   total=total+val(count%)
210   NEXT
220 FOR count%=1 TO num%
230   val(count%)=val(count%)*100/total
240   NEXT
250 ENDPROC
260
270 DEFPROCpie
280 CLS:VDU23,1,0;0;0;0;
```

```
290 FOR count%=1 TO num%
300   pc=val(count%)+oldpc
310   IF val(count%)<.2 NEXT : ENDPROC
320   col%=col%+1
330   IF count%=num% AND count%MOD3=1 col%=col%+1
340   GCOL0,col%MOD3+1
350   ang1=RAD(oldpc*3.6)
360   ang2=RAD(pc*3.6)
370   avang=RAD((oldpc+pc)*1.8)
380   MOVEcx%,cy%
390   MOVEcx%+rad%*COS(ang1),cy%+rad%*SIN(ang1)
400   PLOTsecplot,cx%+rad%*COS(ang2),cy%+rad%*SIN(ang2)
410   VDU5
420   MOVEcx%+440*COS(avang),cy%+440*SIN(avang)
430   PRINT;count%
440   VDU4,23,1,0;0;0;0;
450   oldpc=pc
460   NEXT
470 ENDPROC
480
490 DEFPROCinvalid
500 PRINT''"Invalid entry - please try again."
510 wait=INKEY(200) : CLS
520 ENDPROC
```

2.9 Segments

A segment is an area of a circle between the circumference and a chord as shown below:



They are defined in exactly the same way as arcs and sectors using the following plot codes:

```
&A8 - &AF  (168 - 175)
```

The following example shows how they can be used for drawing a familiar shape.

```
0 REM ACORN
5 ON ERROR VDU23,1,1,0;0;0;0;:END
10 MODE1
20 VDU23,1,0,0;0;0;0;
30 VDU19,2,2;0;
40 x%=640
50 y%=700
60 s%=300
130 GCOL 0,2
140 MOVE x%,y%
150 MOVE x% + 2*s%/3,y%
160 PLOT &CD,x%,y%+s%
170 GCOL 0,0
180 MOVE x%,y%-s%/1.5
190 PLOT 0,-s%/2,0
200 PLOT &A9,s%,0
210 MOVE x%,y%-s%/10
220 PLOT 0,-3*s%/4,0
230 PLOT &A9,3*s%/4,0
240 GCOL 0,2
260 MOVE x%,y%-s%/6
270 PLOT 0,-3*s%/4,0
280 PLOT &A9,3*s%/4,0
290 MOVE x%,y%-s%
300 PLOT 1,s%/4,-s%/10
310 PLOT &51,-s%/30,-s%/20
320 PLOT 1,-s%/4,s%/8
330 PLOT &51,s%/30,s%/30
340 PLOT &51,-s%/2,s%/16
350 PLOT &51,-s%/50,s%/30
360 REPEATUNTILFALSE
```

3 Pattern fills and line styles

On the BBC Microcomputer or Acorn Electron there are eight pure colours and eight flashing colours available for use in any graphics operations. When a triangle, for example, is plotted it is filled with a solid block of just one of these colours. The Graphics Extension ROM allows the colours to be interwoven to give a tremendous range of colour patterns.

Some of these patterns appear almost as pure colours, for example a pattern made up of alternate red and yellow pixels will look orange. Similarly red and magenta gives a deep pink colour.

3.1 Using colour patterns

There are four default patterns set up for you which depend on the mode you are in:

MODE 0 and MODE 4

`GCOL 16,0`	3 parts black	1 part white (cross hatch)
`GCOL 32,0`	1 part black	1 part white (cross hatch)
`GCOL 48,0`	1 part black	3 parts white (cross hatch)
`GCOL 64,0`	1 part black	1 part white (diagonal stripes)

MODE 1 and MODE 5

`GCOL 16,0`	3 parts red	1 part yellow (cross hatch)
`GCOL 32,0`	1 part red	1 part yellow (cross hatch)
`GCOL 48,0`	1 part red	3 parts yellow (cross hatch)
`GCOL 64,0`	1 part white	1 part yellow (cross hatch)

MODE 2

`GCOL 16,0`	1 part red	1 part yellow (cross hatch)
`GCOL 32,0`	1 part red	1 part magenta (cross hatch)
`GCOL 48,0`	1 part yellow	1 part green (cross hatch)
`GCOL 64,0`	1 part yellow	1 part white (cross hatch)

The second parameter which has been set to 0, has no effect on the pattern produced. It is used instead to determine whether it is the foreground or background colour which is being set. Values `0-127` apply to the foreground colour and `128-255` to the background.

All the shapes which are available in the operating system or Graphics Extension ROM can use these colour patterns; try the following:

```
MODE 2
GCOL 16,0
MOVE 100,100
MOVE 800,800
PLOT &55,700,200
```

or

```
MODE 1
GCOL 32,0
MOVE 640,512
PLOT &9D,740,512
```

It is possible to plot lines using these colour patterns in a similar manner, however the effects may be rather difficult to control. Consider, for example, a line drawn at 45° in MODE 1. If the pattern being used was alternate black and white pixels then this line would be drawn in either all white or all black, the latter not being particularly visible on a black background.

3.2 Defining your own patterns

You may define your own colour patterns using VDU commands as follows:

`VDU 23,2,n1,n2,n3,n4,n5,n6,n7,n8`	defines	`GCOL 16,0`	pattern 1
`VDU 23,3,n1,n2,n3,n4,n5,n6,n7,n8`	defines	`GCOL 32,0`	pattern 2
`VDU 23,4,n1,n2,n3,n4,n5,n6,n7,n8`	defines	`GCOL 48,0`	pattern 3
`VDU 23,5,n1,n2,n3,n4,n5,n6,n7,n8`	defines	`GCOL 64,0`	pattern 4

To work out the values of the parameters needed to produce certain patterns we need to look at what the pattern fill is actually doing.

The pattern fill works with blocks of pixels, the size of these blocks depending on the number of colours available in the mode:

MODE	**block size (pixels)**
0	8 × 8
1	4 × 8
2	2 × 8
4	8 × 8
5	4 × 8

In all the cases each pixel may be assigned a colour independently of the others. Each parameter of the VDU command corresponds to a row in the pixel block. The first parameter contains the value of the top row etc. The way the value of the parameter is interpreted depends on the mode being used.

The easiest case to understand is MODE 0. In this case each pixel in the block corresponds to one bit of the parameter. For example, to set a row to the pattern : black, white, white, white, black, black, black, white the following should be used:

black	white	white	white	black	black	black	white
0	1	1	1	0	0	0	1
128	64	32	16	8	4	2	1

The value is &71 or 113

Hence, defining a MODE 0 pattern is equivalent to setting up a user-defined character.

The following example sets up the rows to be alternate double pixels of black and white:

	hex	**decimal**
■■□□■■□□	&33	51
□□■■□□■■	&CC	204
■■□□■■□□	&33	51
□□■■□□■■	&CC	204
■■□□■■□□	&33	51
□□■■□□■■	&CC	204
■■□□■■□□	&33	51
□□■■□□■■	&CC	204

```
VDU 23,2,&33,&CC,&33,&CC,&33,&CC,&33,&CC
```

or

```
VDU 23,2,51,204,51,204,51,204,51,204
```

This example deliberately used double pixels since most colour television sets do not have the resolution to cope with single pixels in MODE 0. Using single ones produces a shimmering effect on the screen. Hence in this mode it is advisable to always work with double pixels unless your program is only to be used on medium to high resolution monitors.

MODE 4 is essentially the same as MODE 0 except that now the resolution is lower and the advice on using double pixels with television sets doesn't apply. Hence, more complicated patterns are possible as illustrated by the following program.

```
0 REM PATTERN
5 ON ERROR VDU23,1,1,0;0;0;0;:END
10 MODE4
20 VDU23,1,0;0;0;0;
30 cirplot=&99
40 cirdraw=&91
50 VDU19,0,6,0,0,0,19,1,0,0,0,0
60 REPEAT
70   RESTORE RND(16)*20+200
80   VDU23,2
90   FOR item%=1 TO 8
100     READpar% : VDUpar%
110     NEXT
120   x%=RND(1280) : y%=RND(1024)
130   r%=RND(350)+50
140   GCOL0,0
```

```
150   MOVEx%,y% : PLOTcirplot,r%+20,0
160   GCOL16,0
170   MOVEx%,y% : PLOTcirplot,r%,0
180   GCOL0,1
190   MOVEx%,y% : PLOTcirdraw,r%+20,0
200   MOVEx%,y% : PLOTcirdraw,r%,0
210   UNTIL FALSE
220 REM Basket Weave
230 DATA&F8,&74,&22,&47,&8F,&17,&22,&71
240 REM Tiles
250 DATA&80,&80,&41,&3E,&08,&08,&14,&E3
260 REM Scales
270 DATA&03,&84,&48,&30,&00,&02,&01,&01
280 REM Bricks
290 DATA&FF,&80,&80,&80,&FF,&08,&08,&08
300 REM Polka dots
310 DATA&B1,&30,&03,&1B,&D8,&C0,&0C,&8D
320 REM Wigwam
330 DATA&88,&14,&22,&41,&88,&00,&AA,&00
340 REM Faces
350 DATA&00,&44,&44,&10,&10,&44,&38,&00
360 REM Little men
370 DATA&10,&38,&10,&7C,&10,&28,&44,&00
380 REM Open weave
390 DATA&80,&40,&20,&10,&01,&02,&04,&08
400 REM Pattern 1
410 DATA&1C,&60,&41,&99,&99,&82,&06,&38
420 REM Sloped Bricks
430 DATA&08,&14,&22,&41,&80,&01,&02,&04
440 REM Squiggles
450 DATA&EE,&AA,&BB,&00,&EE,&AA,&BB,&00
460 REM Aliens
470 DATA&00,&38,&54,&7C,&28,&54,&54,&00
480 REM Squares
490 DATA&FE,&82,&BA,&AA,&BA,&82,&FE,&00
500 REM Pattern 2
510 DATA&9E,&20,&9E,&C3,&79,&08,&79,&C3
520 REM Watson
530 DATA&00,&00,&00,&62,&FC,&3C,&24,&00
```

In MODE 2 the situation is different. There are just two pixels in a row, four bits of the parameter being used to hold the value of each colour. However it is not the case that the top four bits correspond to the first colour and the bottom four bits to the other. Instead the bits of each are interleaved:

first colour	**second colour**
green (2)	white (7)
0010	0111

```
0 0 1 0
 0 1 1 1
```

00011101

&1D or 29

To get the colours the other way around different numbers are required:

first colour	**second colour**
white (7)	green (2)
0111	0010

```
0 1 1 1
 0 0 1 0
```

00101110

&2A or 42

Hence a cross-hatch pattern of alternate white and green pixels can be defined as follows:

```
VDU 23,2,29,42,29,42,29,42,29,42
```

or if you wish to use hexadecimal notation:

```
VDU 23,2,&1D,&2A,&1D,&2A,&1D,&2A,&1D,&2A
```

MODE 1 and MODE 5 patterns are constructed in yet another way. In this case each colour is defined using two bits as follows:

colour 1	**colour 2**	**colour 3**	**colour 4**
yellow (2)	red (1)	white (3)	yellow (2)
10	01	11	10

```
1       0
  0       1
    1       1
      1       0

1 0 1 1 0 1 1 0

&B6 = 182
```

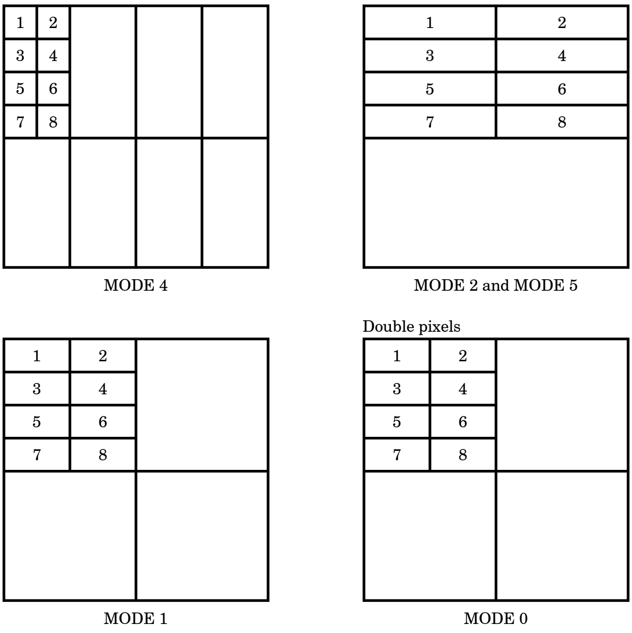
The following example shows how 'shading' can be obtained using different amounts of three colours. The colours change at random showing how well the different colours combine together. Some colour mixtures like blue and yellow look like they are made up of separate pixels whereas others like cyan and yellow blend together very well.

```
0 REM SHADES
5 ON ERROR VDU23,1,1,0;0;0;0;:END
10 MODE1
20 VDU23,1,0;0;0;0;
30 triplot=&55
40 VDU19,1,2,0,0,0
50 GCOL16,0
60 y%=0
70 FOR n%=0 TO 4
80   FOR x%=n% TO (8-n%) STEP2
90     VDU23,2
100      FOR par%=1 TO 8
110        READ v% : VDU v%
120        NEXT
130      MOVEx%*128,y%*200
140      MOVE(x%+2)*128,y%*200
150      PLOTtriplot,(x%+1)*128,(y%+1)*200
160      NEXT
170    y%=y%+1
180    NEXT
190 y%=1
```

```
200 FOR n%=1 TO 4
210   FOR x%=n% TO (8-n%) STEP2
220     VDU23,2
230     FOR par%=1 TO 8
240       READ v% : VDU v%
250       NEXT
260     MOVEx%*128,y%*200
270     MOVE(x%+2)*128,y%*200
280     PLOTtriplot,(x%+1)*128,(y%-1)*200
290     NEXT
300   y%=y%+1
310   NEXT
320 REPEAT
330   wait=INKEY(200)
340   VDU19,RND(3),RND(7),0,0,0
350   UNTIL FALSE
360
370 DATA&F,&F,&F,&F,&F,&F,&F,&F
380 DATA&A5,&F,&A5,&F,&A5,&F,&A5,&F
390 DATA&A5,&5A,&A5,&5A,&A5,&5A,&A5,&5A
400 DATA&F0,&5A,&F0,&5A,&F0,&5A,&F0,&5A
410 DATA&F0,&F0,&F0,&F0,&F0,&F0,&F0,&F0
420 DATA&8F,&2F,&4F,&1F,&8F,&2F,&4F,&1F
430 DATA&97,&AD,&6B,&5E,&97,&AD,&6B,&5E
440 DATA&DA,&B6,&E5,&79,&DA,&B6,&E5,&79
450 DATA&F8,&F2,&F4,&F1,&F8,&F2,&F4,&F1
460 DATA&AF,&5F,&AF,&5F,&AF,&5F,&AF,&5F
470 DATA&D7,&BD,&EB,&7E,&D7,&BD,&EB,&7E
480 DATA&FA,&F5,&FA,&F5,&FA,&F5,&FA,&F5
490 DATA&7F,&DF,&BF,&EF,&7F,&DF,&BF,&EF
500 DATA&F7,&FD,&FB,&FE,&F7,&FD,&FB,&FE
510 DATA&FF,&FF,&FF,&FF,&FF,&FF,&FF,&FF
520 DATA&8F,&2D,&4F,&1E,&8F,&2D,&4F,&1E
530 DATA&5A,&87,&5A,&2D,&5A,&87,&5A,&2D
540 DATA&A5,&78,&A5,&D2,&A5,&78,&A5,&D2
550 DATA&F8,&D2,&F4,&E1,&F8,&D2,&F4,&E1
560 DATA&5F,&8F,&5F,&2F,&5F,&8F,&5F,&2F
570 DATA&B6,&7D,&DA,&A7,&DA,&7D,&BE,&6D
580 DATA&F5,&F8,&F5,&F2,&F5,&F8,&F5,&F2
590 DATA&AF,&7F,&AF,&DF,&AF,&7F,&AF,&DF
600 DATA&FA,&F7,&FA,&FD,&FA,&F7,&FA,&FD
610 DATA&BF,&FD,&F7,&EF,&BF,&FD,&F7,&EF
```

3.3 Simple patterns

Often the most effective way of using the pattern fills is for simple cross-hatch patterns. For people wanting to use these sorts of colour patterns a simpler way of defining the patterns is available. In this method just a small block of eight pixels is defined which is used to form the normal sized block as seen below.



The eight pixel colours are set up using:

```
VDU 23,12,p1,p2,p3,p4,p5,p6,p7,p8 defines GCOL 16,0 pattern 1
VDU 23,13,p1,p2,p3,p4,p5,p6,p7,p8 defines GCOL 32,0 pattern 2
VDU 23,14,p1,p2,p3,p4,p5,p6,p7,p8 defines GCOL 48,0 pattern 3
VDU 23,15,p1,p2,p3,p4,p5,p6,p7,p8 defines GCOL 64,0 pattern 4
```

The following example shows the different patterns which can be obtained using a simple cross-hatch in MODE 2.

```
0 REM HATCH
5 ON ERROR VDU23,1,1,0;0;0;0;:END
10 MODE 2
20 VDU23,1,0;0;0;0;
30 recplot=&65
40 FOR x%=0 TO 7
50   PROCdrawblock(x%,x%,300+96*x%,100)
60   NEXT
70 FOR y%=0 TO 7
80   PROCdrawblock(y%,y%,150,250+96*y%)
90   NEXT
100 FOR x%=0 TO 7
110   FOR y%=0 TO 7
120     PROCdrawblock(x%,y%,300+96*x%,250+96*y%)
130     NEXT
140   NEXT
150 REPEATUNTILFALSE
160
170 DEFPROCdrawblock(c1%,c2%,x%,y%)
180 VDU23,12,c1%,c2%,c2%,c1%,c1%,c2%,c2%,c1%
190 GCOL16,0
200 MOVEx%,y%
210 PLOTrecplot,x%+80,y%+80
220 GCOL0,7
230 MOVEx%,y%
240 DRAWx%+80,y%
250 DRAWx%+80,y%+80
260 DRAWx%,y%+80
270 DRAWx%,y%
280 ENDPROC
```

3.4 Dot-dash lines

The linestyles in which straight lines are drawn can be set up to be a pattern of dots and dashes. This is done in a similar way to that in which the colours for solid shapes are plotted as patterns made up of different coloured pixels.

The dot-dash pattern is set up using

```
VDU 23,6,n1,n2,n3,n4,n5,n6,n7,n8
```

where `n1..n8` define a bit pattern. Each bit which is set to 1 represents a point plotted and each set to 0 no point. For example

n1	n2	n3	n4	n5	n6	n7	n8
&54	&48	&3F	&A9	&26	&BB	&C3	&9D

The maximum pattern repeat length is 64 pixels, and you can set up any repeat between 1 and 64 using:

```
*FX163,242,n
```

If you set `n` to `0` then this sets up the default pattern which has a repeat of 8 and is alternately on and then off ie `10101010` (`&AA`).

There are four different methods which may be used to plot the line.

`PLOT` *range*	*Effect*
`&10 - &17`	Both end points included, the pattern being restarted when each new line is drawn.
`&18 - &1F`	Final point omitted, the pattern being restarted when each new line is drawn.
`&30 - &37`	Initial point omitted, the pattern being continued when each new line is drawn.
`&38 - &3F`	Both end points omitted, the pattern being continued when each new line is drawn.

4 Flood fills

This chapter is concerned with how to fill the inside of any closed region, however awkward the shape. The method used is to ‘flood fill’ it. Using this method you can start off at any point in the interior of the shape to be filled and the whole shape will be shaded in one go. The Graphics Extension ROM contains two different flood fill options which are described in this chapter.

Full flood fills can only be used in the ‘FLOOD ON’ state. This is the default state when the GXR is active. It can be turned on or off using:

```
*FLOOD
*NOFLOOD
```

4.1 Flood to non-background

This can be used on shapes which are in the current background colour and bordered by non-background colours. The shape will be filled with the colour specified. The plot codes which are used for this are as follows:

```
&80 - &87  (128 - 135)
```

To use one of these plots then the following sort of command should be given:

```
PLOT &85,640,512
```

This will start filling from the point 640,512 ie the middle of the screen. If this point is in a non-background colour then it will return immediately, otherwise it will fill in all directions until it reaches either a non-background colour, the edge of the screen, or the edge of the graphics window.

Flood fills may be performed using either pure colours or colour patterns. The following program draws a random maze and flood fills each of the separate regions in a different cross hatch pattern.

```
0 REM MAZE
5 ON ERROR VDU23,1,1,0;0;0;0;:END
10 MODE2
20 VDU23,1,0;0;0;0;
30 floodnbg=&85
40 point=&45
50 c1%=1 : c2%=0
60 PROCgrid
```

```
70 PROCgaps
80 PROCflood
90 REPEATUNTILFALSE
100
110 DEFPROCgrid
120 FOR x%=2 TO 38 STEP2
130   MOVEx%*32,64 : DRAWx%*32,960
140   NEXT
150 FOR y%=2 TO 30 STEP2
160   MOVE64,y%*32 : DRAW1216,y%*32
170   NEXT
180 ENDPROC
190
200 DEFPROCgaps
210 FOR y%=2 TO 28 STEP2
220   FOR x%=2 TO 36 STEP2
230     REPEAT
240       e%=RND(4)-1
250       UNTIL NOT(e%=0 AND x%=36) AND NOT(e%=3 AND y%=2) AND
NOT(e%=2 AND x%=2) AND NOT(e%=1 AND y%=28)
260     IF e%=0 PROCjoiny(x%+2,y%)
270     IF e%=1 PROCjoinx(x%,y%+2)
280     IF e%=2 PROCjoiny(x%,y%) : PROCdot
290     IF e%=3 PROCjoinx(x%,y%) : PROCdot
300     NEXT : NEXT
310 ENDPROC
320
330 DEFPROCflood
340 GCOL16,0 : GCOL0,128
350 FOR x%=3 TO 37 STEP2
360   FOR y%=3 TO 29 STEP2
370     IF POINT(x%*32,y%*32)=0 PROCcol : PLOTfloodnbg,x%*32,y%*32
380     NEXT
390   NEXT
400 ENDPROC
410
420 DEFPROCjoinx(x%,y%)
430 GCOL0,0
440 MOVEx%*32+8,y%*32
450 DRAW(x%+2)*32-8,y%*32
460 ENDPROC
470
480 DEFPROCjoiny(x%,y%)
```

```
490 GCOL0,0
500 MOVEx%*32,y%*32+4
510 DRAWx%*32,(y%+2)*32-4
520 ENDPROC
530
540 DEFPROCcol
550 c2%=c2%+1
560 IFc2%=8 c1%=c1%+1 : c2%=c1%
570 IFc1%=8 c1%=1 : c2%=1
580 VDU23,12,c1%,c2%,c2%,c1%,c1%,c2%,c2%,c1%
590 ENDPROC
600
610 DEFPROCdot
620 p1%=POINT(x%*32-8,y%*32)
630 p2%=POINT(x%*32+8,y%*32)
640 p3%=POINT(x%*32,y%*32-8)
650 p4%=POINT(x%*32,y%*32+8)
660 GCOL0,p1% OR p2% OR p3% OR p4%
670 PLOTpoint,x%*32,y%*32
680 ENDPROC
```

Note that if you wish to colour in a shape it must be totally enclosed by a solid border. If there is a gap anywhere then the colour will ‘leak out’ into other regions.

4.2 Flood until foreground

Whereas the previous flood fill would fill an area which was in the background colour, this one will fill a shape currently in any colour, except the present foreground colour, with the colour specified. The plot codes used are:

```
&88 - &8F  (136 - 143)
```

The following example illustrates how to use the two methods of flooding. A Sierpinski curve is drawn in white on a black background then the area enclosed by the curve is filled in orange using flood to non-background. The next stage is to reverse the procedure by changing the orange back to black. This is done by setting the background colour to orange, the foreground colour to black and again using flood to non-background. The third stage separates the area of the curve into four parts using a red cross and fills each of the areas in a different colour in a similar manner to stage 1. Finally the whole area is flooded with white. This could be done in four sections, each quarter being filled in a similar manner to stage 2. However a quicker method is to fill it in one go using flood to foreground with the foreground colour being white.

```
0 REM SIERPIN
5 ONERRORVDU4:END
10 MODE1
20 VDU23,1,0;0;0;0;
30 floodnbg=&85
40 floodfg=&8D
50 order=4
70 size=(2^order-1)*4+2
80 h=INT(1000/size)
90 h2=h*2
100 PROCsierpinski(order,640,512)
110 wait=INKEY(200)
120 GCOL32,0
130 PLOTfloodnbg,640,512
140 wait=INKEY(200)
160 GCOL0,0 : GCOL32,128
170 PLOTfloodnbg,640,512
180 wait=INKEY(200)
190 GCOL0,1 : GCOL0,128
210 MOVE612,512 : DRAW668,512
220 MOVE640,484 : DRAW640,540
230 GCOL0,1 : PLOTfloodnbg,644,516
240 GCOL16,0 : PLOTfloodnbg,636,516
250 GCOL32,0 : PLOTfloodnbg,636,508
260 GCOL48,0 : PLOTfloodnbg,644,508
```

```
270 wait=INKEY(200)
280 GCOL0,3
290 PLOTfloodfg,640,512
300 REPEATUNTILFALSE
310
320 DEFPROCsierpinski(n,x,y)
330
340 LOCAL k
350 IF n=0 MOVEx-h,y : DRAWx,y+h : DRAWx+h,y : DRAWx,y-h : DRAWx-
h,y : ENDPROC
360 k=2^n*h
370 PROCsierpinski(n-1,x-k,y-k)
380 PROCsierpinski(n-1,x-k,y+k)
390 PROCsierpinski(n-1,x+k,y+k)
400 PROCsierpinski(n-1,x+k,y-k)
410 MOVEx-h2,y-h
430 GCOL0,3
440 PLOT0,0,4 : PLOT1,0,h2-4
450 GCOL0,0
460 PLOT0,4,4 : PLOT1,h-4,h-4
470 GCOL0,3
480 PLOT1,h2,0
490 GCOL0,0
500 PLOT0,4,-4 : PLOT1,h-4,-h+4
510 GCOL0,3
520 PLOT1,0,-h2
530 GCOL0,0
540 PLOT0,-4,-4 : PLOT1,-h+4,-h+4
550 GCOL0,3
560 PLOT1,-h2,0
570 GCOL0,0
580 PLOT0,-4,4 : PLOT1,-h+8,h-8
590 GCOL0,3
600 PLOT0,4,4
610 ENDPROC
```

Note

Flooding a region with a colour which it already contains will not work. For example if you are attempting a flood to non-background with the background colour being black then you should not try to flood in black or a pattern which contains black pixels. If you do this then it will try to fill a region, find that pixels in it are still in the background colour and try again. When this occurs a small region of the screen will be flooded and then the flood fill will give up.

4.3 Horizontal line fills

Four methods are provided for filling a line horizontally until certain colours are met.

4.3.1 Line fill left and right to non-background

The plot codes for this are as follows:

```
&48 - &4F  (72 - 79)
```

The effect of these plots is to move horizontally outwards from the specified point both left and right until non-background points are found. A line is then drawn between these two points in accordance with the `PLOT` action and the `GCOL` settings.

4.3.2 Line fill right to background

The plot codes are:

```
&58 - &5F  (88 - 95)
```

These plot codes are used to rub out objects from the screen. The action is to start at the specified point and travel rightwards in accordance with the `PLOT` action and the `GCOL` settings until a background point is found.

4.3.3 Line fill left and right to foreground

The plot codes are:

```
&68 - &6F  (104 - 111)
```

These plots start from the specified point and move outwards to the left and right until foreground points are found. A line is drawn between these points in accordance with the `PLOT` action and the `GCOL` settings.

4.3.4 Line fill right to non-foreground

The plot codes are:

```
&78 - &7F  (120 - 127)
```

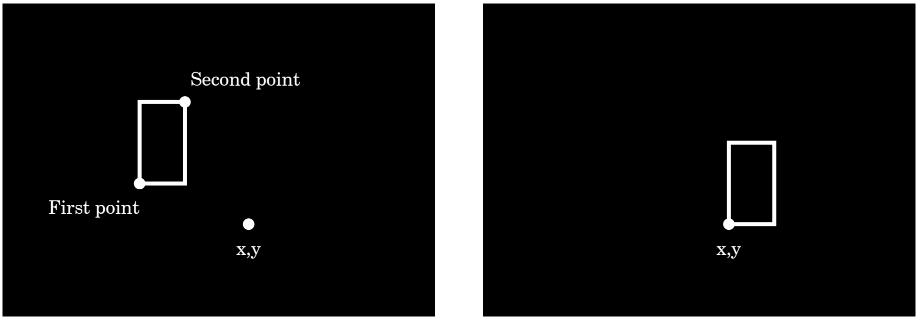
These codes are also used for erasing items from the screen. The action is to start at the point specified and travel rightwards in accordance with the `PLOT` action and the `GCOL` settings until a non-foreground point is found.

5 Moving areas of the screen

5.1 Copy / move

The Graphics Extension ROM enables you to pick up a rectangular area of the screen and either make a copy of it elsewhere on the screen or move it to another position, replacing it with a block of the background colour.

For either of these routines you need first to mark the rectangular area which is to be used by moving to two diagonally opposite corners of it. Then use the appropriate plot code followed by the X and Y coordinates of the point on the screen which marks where the rectangle is to be placed. The bottom left hand corner of the rectangle visited will be placed on the X and Y coordinates given in the plot statement.



The plot code block used is:

```
&B8 - &BF  (184 - 191)
```

However the usual conventions for what each of the eight different values mean do not apply, instead they are as follows:

&B8	(184)	Move only, relative
&B9	(185)	Move rectangle relative
&BA	(186)	Copy rectangle relative
&BB	(187)	Copy rectangle relative
&BC	(188)	Move only, absolute
&BD	(189)	Move rectangle absolute
&BE	(190)	Copy rectangle absolute
&BF	(191)	Copy rectangle absolute

The following example shows part of the screen being moved around. It illustrates how a large area may be repositioned quickly and smoothly.

```
0 REM CLOUD
10 ONERRORVDU4:END
20 MODE2
30 *fx9
40 *fx10
50 VDU23,1;0;0;0;0
60 VDU23,13,8,9,9,8,8,9,9,8
70 VDU19,8,3;0;19,9,7;0;
80 VDU23,12,6,7,7,6,7,6,6,7
90 PROCmount
100 PROCsun(640,970)
110 PROCcloud(640,712)
120 YY%=568:SY%=300:SX%=596
130 TY%=YY%
140 XX%=352
150 step%=16
160 FORmove%=1TO2
170   REPEAT
180     TX%=XX%+step%
230     MOVEXX%,YY%
240     MOVEXX%+SX%,YY%+SY%
250     PLOT&BD,TX%,TY%
280     XX%=XX%+step%
290     UNTIL XX%>=1248-SX% OR XX%<=32
300   step%=-step%
310   NEXT
```

```
320 PROCnighttime
330 REPEATUNTILFALSE
340
350 DEFPROCcloud(x%,y%)
360 RESTORE
370 VDU29,x%;y%;
380 REPEAT
390   READ x,y
400   GCOL16,0:PROCell(x,y,100,50,1)
410   GCOL0,4:PROCell(x,y,100,50,0)
420   UNTIL x=-1
430 GCOL16,0:PROCell(x,y,232,116,1)
440 VDU29;0;0;
450 ENDPROC
460 DATA 60,96,160,60,-184,40,-88,-88
470 DATA -180,-36,-64,92,88,-92,204,-20
480 DATA -1,-1
490
500 DEFPROCell(x%,y%,xr%,yr%,fill%)
510 MOVE x%,y%
520 MOVE x%+xr%,y%
530 IF fill%=1 PLOT &CD,x%,y%+yr%: ENDPROC
540 PLOT &C5,x%,y%+yr%
550 ENDPROC
560
570 DEFPROCmount
580 GCOL0,128+6
590 COLOUR128+6
600 CLS
610 X%=RND(-456)
620 X%=0:Y%=200
630 DY%=1
640 MOVEX%,Y%
650 GCOL0,0
660 FORI%=1TO40
670   FORJ%=0TO3
680     X%=X%+8
690     Y%=Y%+DY%*4
700     IFY%>500 Y%=500
710     DRAWX%,Y%
720     NEXT
730   DY%=4-RND(7)
740   NEXT
```

```
750 GCOL0,2:GCOL0,128+6
760 PLOT&85,10,4
770 ENDPROC
780
790 DEFPROCsun(X,Y)
800 GCOL32,0
810 MOVEX,Y
820 PLOT&9D,X+50,Y
830 ENDPROC
850
860 DEFPROCnighttime
870 FORI%=640TO1350STEP8
880   X%=I%
890   MOVEX%-50,920+640-X%:PLOT0,116,116
891   *FX19
900   PLOT&BD,X%+8-50,920+640-X%-8
910   IFX%=1104 VDU19,8,1;0;
920   IFX%=1000 VDU19,6,4;0;
930   IFX%=904 VDU19,9,1;0;
940   IFX%=800 VDU19,9,3;0;
950   NEXT
960 VDU19,6;0;0,19,4;0;0
970 ENDPROC
```

6 Graphics examples and utilities

Side 1 of the cassette accompanying this pack contains the following files:

`INSTALL`	(Program to transfer tape contents to disc)
`SQUARES`	
`CHART`	
`TILES`	
`CIRCLES`	
`SPIRAL`	
`ELLIPSE`	
`FLOWERS`	
`SHIP`	
`PIE`	
`ACORN`	
`PATTERN`	
`SHADES`	
`HATCH`	
`MAZE`	
`SIERPIN`	
`CLOUD`	
`CASTLE`	
`HOUSES`	
`EDITOR`	
`PAINT`	
`MAIN`	(this file is used by `PAINT` and cannot be `CHAIN`ed)

The majority of these are the short examples which have been used to illustrate a particular feature and are listed in the previous section. However, the last few have been added to provide further examples of the power of the Graphics Extension ROM and to help you to use it more easily.

6.1 Program 1 CASTLE

An impressive example of the type of design made possible with the many features of the Graphics Extension ROM.

6.2 Program 2 HOUSES

This is another general example program showing particularly the use of flood fills. It shows one method of getting around the problem of not being able to flood with a pattern which contains any of the background colour.

The problem is how to fill the roof of the house in a tile pattern of red and black. To do this what essentially happens is that the roof is filled with yellow and then filled with the pattern. Since it is the logical colour rather than the actual colour which is important to the flood routine the palette can be changed so that logical colour 2 (which is initially yellow in MODE 1) appears as black and so when the picture is drawn there appears to be a delay while the fill in logical colour 2 is taking place and then the roof is filled in the tile pattern.

This trick is also used to make the house appear to be built leaving gaps for the doors and windows. In this case logical colour 2 is set to black and used to draw the outline of the house which is then flooded with the brick pattern.

6.3 Program 3 EDITOR

Working out the values required for a particular pattern can be difficult and time consuming. The following program does all the hard work for you by calculating the values. You enter the graphics mode which you wish to use and a grid is displayed on the screen, the size of which depends on the mode selected. Your current position within the grid is indicated by a flashing cross and you can move around by using the cursor arrow keys. To enter a colour in a block of the grid move the cross to that block and press the number of the colour which you would like. The range of colours to choose from depends on the mode you are using:

MODE	**Colours available**
0	0 or 1
1	0, 1, 2 or 3
2	0 .. 9 or A .. F
4	0 or 1
5	0, 1, 2 or 3

Down the right-hand side of the screen you will see a strip of coloured blocks; this is the palette. It allows you to change the relationship between the colour-number and the actual colour which is displayed. To do this type P. The pointer by the palette will start flashing and can be moved up and down using the appropriate arrow key. To change the appearance of a colour-number press one of the keys 0 .. 9 or A .. F. This will select the actual colour from the 16 available. To return to the grid press **RETURN**.

While you are defining your pattern a block plotted using the pattern so far will be displayed at the top of the screen and the values of each of the rows will be displayed in hexadecimal notation to the right of the grid. Having designed a good pattern you can note down the numbers and use them in your own programs to plot a shape or perform a flood fill.

6.4 Program 4 PAINT

This is a painting program which allows you to create and save your own pictures very simply.

The screen is divided into two parts, the control panel down the left-hand side and the ‘canvas’ on the right. The control panel is further divided into three columns. The middle column enables you to select what you want to do, eg draw a circle, move part of the picture or print text. The other two columns allow you to specify other details, eg what colour or linestyle you wish to use.

You can move around the screen using the arrow keys. Pressing the **SHIFT** key at the same time will have the effect of moving you at eight times the normal speed. When you are on the control panel your position is indicated by an arrow and you can select a particular object by pressing the **CTRL** key whilst pointing to that object. When you are on the canvas your position is indicated by a symbol which alters depending on which central function you have selected.

The colours and colour patterns in the left-hand column determine what colour all the lines, shapes and flood fills will use. The currently selected one is indicated in the box along the bottom. The palette can be changed using the boxes at the top of the right-hand column, eg to change red into green move to the box containing red and press **CTRL** until the box becomes green.

When other options are selected this is indicated by the colours of the boxes they are in being inverted.

The series of boxes beneath the palette determine whether straight lines are to be drawn in a solid or dot-dash pattern. The symbols in the centre column are more complicated and are dealt with individually below.

The canvas can be cleared and all the options reset to their defaults at any time by pressing R. The best way to find out about the facilities available in PAINT is to ‘try it and see’.

The *spray gun* allows you to draw one or several lines at a time; each dot of the spray gun leaves one line behind it when it moves. Press **COPY** to increase the number of dots and **DELETE** to reduce it. To draw using the spray gun first you should move to the place where you want to start from and press **CTRL**. Then with **CTRL** held down, move around the screen until you are happy with the position of the line(s). Finally release **CTRL**.

The *hand* will move the whole screen in any direction. To start press **CTRL** and with it held down move the hand around. When you release **CTRL** the whole screen will move so that the initial position ends up at the final hand position. Note that any part of the picture which is shifted off the screen will be lost – you cannot scroll it on again.

The *flood fill* can be used to fill any enclosed area of the screen with the current colour or colour pattern. Simply move to any point within the area you want to flood and press **CTRL**. Areas can be 'unflooded' by pressing **TAB**, however this often does not only reverse the action of the flood but affects other shapes on the canvas depending on the colours used.

The *line* allows a series of lines to be drawn, each one beginning where the previous one finished, so producing a continuous line drawing. Press and release **CTRL** to begin and then each time a line is to be drawn. The line styles in the right-hand column are used with the line drawing option to give various 'dot-dash' patterns.

Ellipse outlines and *ellipse fills* are produced by moving to the position which is to be the centre of the ellipse and pressing **CTRL**. Then whilst holding down **CTRL** the width of the ellipse can be altered using the arrow keys. Releasing **CTRL** fixes the 'width' of the ellipse. Now move the cursor to 'carry' the highest point on the ellipse to the required position. Press and release **CTRL** to fix the final ellipse (and fill it if selected).

Circle outlines and *circle fills* are also drawn in a similar manner. Press **CTRL** at the centre and alter the position of the circle symbol to indicate the size of the circle you want. Then release **CTRL**.

The *triangle* performs a triangle fill. Move the cursor to one corner and press and hold **CTRL**. Move to the second corner and release **CTRL**. Select the position of the third corner and press and release **CTRL** to perform the plot.

The *arc*, *segment* and *sector* are all used in a similar way. Firstly, move the cursor to the centre of the circle and press and hold **CTRL**. Then move to the start point on the circumference and release **CTRL**. Move the cursor to open up the required angle and press and release **CTRL** to fix the shape.

The *camera* allows copies of any rectangular area of the canvas to be made. Move the symbol to one corner of the area and press **CTRL**, hold down **CTRL** whilst moving the cursor keys to produce a box of the right size and release **CTRL** when you have enclosed the area to be copied. Then pressing the cursor keys will move a second rectangle around which should be placed where you want the copy to be put. To perform the copy press **CTRL**. The second rectangle may be moved again and further copies of the first rectangle made by pressing **CTRL**.

The *scissors* will move a rectangular area of the screen, replacing it by a block of the background colour. This is performed in a similar way to the copy rectangle routine described above.

The *typewriter* is used to print text on the screen. Press **CTRL** to start printing (a pencil will replace the typewriter) and **RETURN** to indicate that you have finished. Note that text can only be placed between the starting point and the right-hand side of the canvas – it is not allowed to wrap round to the beginning of the next line.

The *grid* when selected restricts the symbols to move only between certain points on the canvas so making it easier to draw several circles with exactly the same centre point etc.

Loading and saving can be carried out by pressing **L** or **S**. They clear the control panel and allow you to type in the name you wish to use.

7 Sprite graphics

7.1 Initial preparations

If you wish to write a program which uses sprite graphics then you should begin by designing the sprites.

When the sprites are designed the information about them has to be saved in memory. Therefore, before you can design a sprite you must reserve a certain amount of memory space for it. This memory is allocated in units of 1 page (256 bytes) and is located below the bottom of user memory so it increases the value of `PAGE`.

The amount of space which you need to allocate depends upon the number of different sprites you wish to use and the size each one is to be. Initially, when you are learning to use the sprite editor, just one page should be sufficient and this is reserved by typing

```
*SSPACE 1
```

and then pressing **BREAK** when prompted to do so.

Any value can be reserved as long as there is enough room between `PAGE` and the bottom of the screen memory, but remember that for each page which is used you lose a page for your program. Care should be taken not to try to reserve more space than is available since this will have unpredictable results. Ten pages of work space should be sufficient even for very large numbers of sprites, and three to five pages is adequate provision for normal use.

You are now ready to design your first sprite.

7.2 Designing a sprite

7.2.1 Entering the sprite editor

Sprites are designed using the sprite editor. This will display two versions of the sprite, one actual sized one which is there to show you what your sprite will look like when you use it in your program, and another much enlarged version which you use to make your alterations. The sprite editor is entered using the command

```
*SEDIT n
```

where n is the number of the sprite you wish to create or alter. Initially no sprites will be in existence so a sensible value to give it is 1. Any number between 0 and 255 may be used.

It is important to remember that you should design your sprite in the same screen mode as that in which your program will plot it. The sprite editor does not allow you to change mode once you have started to edit a sprite so you must select the mode before entering the editor, for example, to design sprite 1 in MODE 1 you should type

```
MODE 1
*SEDIT 1
```

You cannot enter the editor in non-graphics modes, ie MODE 3, MODE 6 or MODE 7.

On entering the editor a rectangle will be drawn in the bottom left-hand corner of the screen, this contains the initial state of the sprite. Initially the sprite is 1 byte in size, each of the pixels in it being set to black. The number of pixels per byte is always one vertically but can be either 2, 4 or 8 horizontally depending on which mode you are in. Hence, the initial sprite contains one row but a varying number of columns, each one pixel wide. In the left-most pixel there will be a circular cursor which shows your position within the sprite.

Displayed at the top of the screen will be some information representing the status of the current sprite. On the top line is printed the `Mode` you are using. Beneath this is the sprite number and the size of the sprite in bytes, initially [1,1] which means one byte wide and one byte high. The ‘`U`’ at the top left indicates that the cursor’s pen is raised (Up). This means that when the cursor is moved it will not leave a trail. Next to this there is a rectangle of the current colour followed by the current position of the cursor which says which byte of the sprite it is in. There are two points to note about this. The bottom left-hand byte of the sprite has coordinates (0,0). Hence if the size of a sprite is increased to 6 bytes say, in either direction, then the position of the cursor can be between 0 and 5. Also moving between two pixels within the same byte will not alter the values given as the cursor’s coordinates.

In the editor the function keys **f0** to **f8** and **SHIFT f3** to **SHIFT f6** are set up so that each represents a different command. A function key card is provided in the pack and if you haven’t already done so you should put this under the clear plastic strip above the keyboard so that the block ‘PEN UP/DOWN’ lines up with **f0**. Using this you should be able to select the appropriate key very easily.

7.2.2 Colouring the pixels

The circular cursor can be moved around the rectangle using the arrow keys. To colour a pixel press **RETURN** and this will enter the current colour (which is initially white) at the cursor position. The current colour can be altered by simply entering the value corresponding to the actual colour required. This value should be entered in hexadecimal notation so that in MODE 2 flashing cyan-red for example which has the decimal value 14 can be selected by pressing '**E**'. Entering a value will alter the colour block being displayed at the top of the screen so you always have a record of what the current colour is.

In addition to this a block in colour 7 (normally white) can be added at any time by pressing **COPY** and one in colour 0 (normally black) by pressing **DELETE**.

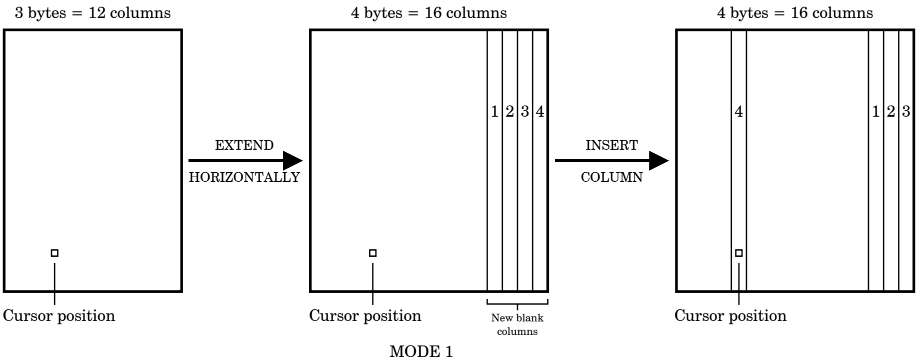
An alternate method of moving to a square and colouring it in is to select PEN DOWN by pressing **f0**. Then, when the cursor is moved right, left, up or down one pixel (if it can) the square it is on will be coloured in automatically. Pen down mode is indicated by the letter at the top of the screen changing to `D` instead of `U`.

7.2.3 Increasing the size of a sprite

The initial size of the sprite is undoubtedly too small to design any interesting shapes. Its size can be altered in a number of different ways. To add another byte at the top press EXTEND VERTICALLY (**f3**), and to add another byte to the right press EXTEND HORIZONTALLY (**f4**). If you carry on pressing these keys you will notice that two things happen. The first is that when the rectangle reaches a certain size increasing it further will not affect the screen display. The sprite is in fact being enlarged, as you can tell if you watch the size at the top of the screen change, but the extra rows/columns are being added 'off the screen'. To move to part of the sprite which is not contained in the rectangle just use the arrow keys in the normal way and when the cursor reaches the right-hand edge or top of the rectangle the left-hand edge or bottom of the sprite will be scrolled out of view. Secondly when the sprite reaches another critical size limit then it will not allow you to increase its size any further. This occurs when the size of a sprite (in bytes) would be greater than the amount of memory left in the space you reserved for your sprites.

Often, when designing symmetrical shapes it is easier to work inwards from the edges towards the centre. This invariably means that you will end up wanting to add or delete a number of rows and columns in the middle of the sprite.

Adding an extra row is straightforward, simply press INSERT ROW (**f5**) and a new row will be added at the current cursor position, if there is sufficient memory to increase the size of the sprite further. Adding an extra column works in a different manner since it does not cause the overall size of the sprite to alter. Pressing INSERT COLUMN (**f6**) will insert a blank column at the current cursor position and will delete the column at the right-hand edge of the sprite. Hence if you do want to increase the size of your sprite by adding another column in the centre you should use EXTEND HORIZONTALLY followed by INSERT COLUMN. This will have the overall effect of adding one column at the current cursor position and either 1, 3 or 7 columns at the right-hand edge depending on which mode you are in and hence how many pixels there are in a byte.



DELETE ROW (**SHIFT f5**) and DELETE COLUMN (**SHIFT f6**) have the opposite effects.

When you have decided upon the size of the sprite then any blank rows or columns of bytes at the top or right of the rectangle should be deleted using REDUCE VERTICALLY (**SHIFT f3**) or REDUCE HORIZONTALLY (**SHIFT f4**). If they are left there then they waste memory since the data for them will be stored as part of the sprite. In addition, since they will make the sprite bigger this will also mean that it will take longer to plot the sprite.

7.2.4 Other useful commands

When moving around a large sprite it can seem to take quite a while to get to the place you want to be using just the arrow keys. To speed things up you can use **SHIFT** arrow keys to move several bytes at a time, scrolling if possible, or **CTRL** arrow keys to go to the edge of the sprite.

To speed up the colouring process FLOOD ROW (**f1**) can be used. This fills the row sideways out from the cursor position in the current colour until it finds a pixel which is in a different colour to the one it is changing. Hence if the cursor is on top of a white pixel and the current colour is red then it will look along to the left and right to find the first non-white pixel in each direction and change all the pixels in between to red. FLOOD COLUMN (**f2**) works in a similar manner.

The other two commands are MIRROR VERTICALLY (**f7**) and MIRROR HORIZONTALLY (**f8**). These allow you to reflect your sprite so that it either faces the other way or is turned upside down. The former of these is particularly useful for games since often the shape you are moving around the screen, eg a man, will need to face the way it is moving, hence two copies will be required which are mirror images of each other.

Finally when you are happy with the sprite you have designed, press **ESCAPE** to leave the sprite editor.

7.3 Loading and saving sprites

All the sprites which you currently have in memory can be saved as a file by typing

```
*SSAVE filename
```

Similarly you can load a sprite file back into memory so that you can either plot or edit the sprites in it by first reserving a sufficient area of memory for it using `*SSPACE` and then typing

```
*SLOAD filename
```

Note that when saving or loading sprites to or from tape, two files of the same name are transferred. The first of these files is merely a very small header file.

If you wish to merge a file of sprites with those you have in memory then you can use

```
*SMERGE filename
```

However, use this with care since if you have a sprite currently in memory which has the same number as one of the sprites in the file then when you merge the two you will lose the version you had in memory.

7.4 Altering sprites

Sprites are altered in the same way as they are designed, using the sprite editor. To alter an existing sprite use

```
*SEDIT n
```

and instead of an empty rectangle being drawn in the bottom corner the current version of the sprite will appear ready to be edited.

Note that when you edit a sprite you are updating the only copy of it which exists in memory. Hence it is a good idea to save a copy of the sprite to disc or tape first so that if you make a mistake whilst editing it and wish to return to the original you can load it back into memory again.

Alternatively you can edit sprite 1 say but have it saved as sprite 2 when you press **ESCAPE** from the editor. To do this type

```
*SEDIT n,m
```

where n is the sprite to be edited (1 in the example) and m is the number it is to be saved with (2 in the example).

You can change the number of a particular sprite by using

```
*SRENUMBER n,m
```

This changes sprite n into sprite m. Note that if there was already a sprite m in memory this will be lost.

Deleting sprites is simply a matter of typing

```
*SDELETE n
```

which deletes sprite n, or

```
*SDELETE n,o,p, ...
```

which deletes all the sprites listed (n,o,p,...).

To delete all the sprites type

```
*SNEW
```

7.5 Plotting sprites

To plot one of the sprites which you currently have in memory is very simple. All you have to do is select which sprite you wish to plot and where you want it to appear on the screen. To select a particular sprite use:

```
*SCHOOSE n
```

A PLOT command is then used to put it on the screen. The possible plot numbers which may be used are:

```
&E8 - &EF  (232 - 239)
```

For example:

```
*SCHOOSE 1
PLOT &ED,640,512
```

will plot sprite 1 with its bottom left-hand corner exactly in the centre of the screen.

As an alternative to *SCHOOSE the following VDU command may be used

```
VDU 23,27,0,n,0,0,0,0,0,0
```

this is entirely equivalent to `*SCHOOSE n`. The advantage of using the `VDU` command is that it can contain variables, hence if you wish to plot eight sprites, numbered 1 to 8 then it is possible to use, for example

```
FOR sprite_num% = 1 TO 8
VDU 23,27,0,sprite_num%,0,0,0,0,0,0
PLOT &ED,x%,y%
NEXT
```

However, since any `*` command can only be used with constant parameters to use `*SCHOOSE` to perform an equivalent task you would have to type

```
*SCHOOSE 1
PLOT &ED,x%,y%
*SCHOOSE 2
PLOT &ED,x%,y%
...
...
*SCHOOSE 8
PLOT &ED,x%,y%
```

7.6 Plotting within graphics windows

If you run some of the example sprite programs described later, or try writing some for yourself, you will notice that the sprites scroll on and off the screen properly at the edges. This is handled for you by the plotting routines which only display the part of the sprite which should be on the screen.

The sprites are ‘clipped’ to the current graphics window which is assumed to be byte aligned. This means that the edge of the graphics window must be at a position such that it is a whole number of bytes from the edge of the screen. In MODE 1, for example, this means that the left-hand x-coordinate of the graphics window should be a multiple of 16 (a byte being 4 pixels wide and a pixel covering 4 coordinate positions) and the right-hand x-coordinate should be 1276 – a multiple of 16. If this is not the case then the sprite will start to disappear at the nearest byte boundary outside the window and so will overlap the graphics window.

7.7 Defining sprites from the screen

As well as using the editor to design a sprite it is also possible to ‘pick up’ any rectangular area of the screen and make this into a sprite. To do this you should mark the rectangle by moving to its bottom left-hand corner and then its top right-hand corner. Then use the command

```
*SGET n
```

or alternatively

```
VDU 23,27,1,n,0,0,0,0,0,0
```

Either of these will define sprite `n` to contain whatever is currently in the rectangle. Then this sprite may be used in exactly the same way as any other.

8 Sprite examples

There are many different ways in which sprites may be used for animation. A few examples are given and discussed below. These example programs and their sprite files can be found on side 2 of the cassette. To run one of the examples type, for example

```
CHAIN "CHOPPER"
```

Before running the sprite programs it is necessary to reserve an area in memory for the sprites. The number of pages required in each case is shown at the beginning of the program, however if you reserve 6 pages by typing

```
*SSPACE 6
```

this will provide plenty of room for all of them.

8.1 Example 1 CHOPPER

The first example shows one method of making a sprite move. It allows you to control a helicopter using the keyboard to move it around the screen. The keys to use are:

`Z`	to move left
`X`	to move right
`:`	to move up
`/`	to move down

It was stated above that when designing a sprite any unused columns and rows should be deleted to make the sprite smaller and hence quicker to plot. However, this example shows one of the exceptions to this rule. Consider what would have happened if the helicopter had been designed so that it reached the edges of the 'box'. When plotted in its initial position there would have been no problem. However, if you had then wished to plot it to the right by 8 pixels, to make it appear to be moving, a new helicopter would have appeared in this position – but the left hand edge of the old one would still have been on the screen. Hence the helicopter would have left a trail behind it as it moved.

To get around this problem the helicopter shape is designed so that it has a black boundary around it (black being the background colour), 8 pixels wide and 4 pixels high. This means that if you plot the sprite 8 pixels to the right of its old position the black border is drawn on top of the left hand edge of the old sprite and so wipes it out.

```
0 REM CHOPPER
5 ONERRORVDU4:END
10 MODE 2
20 VDU23,1;0;0;0;0;0;
30 *FX9,2
40 *FX10,2
50 *SLOAD S.CHOPPER
60 CLS
70 HX%=512
80 HY%=512
90 REPEAT
100   HX%=HX%+FNkeysleft_right
110   HY%=HY%+FNkeysup_down
120   PROCheli
130   UNTILFALSE
140 END
150
160 DEFFNkeysleft_right
170 IF INKEY(-98)<>0 AND HX%>8:=-8
180 IF INKEY(-67)<>0 AND HX%<1060:=8
190 =0
200
210 DEFFNkeysup_down
220 IF INKEY(-73)<>0 AND HY%<900:=4
230 IF INKEY(-105)<>0 AND HY%>8:=-4
240 =0
250
260 DEFPROCheli
270 VDU23,27,0,1,0,0,0,0,0,0
280 PLOT&ED,HX%,HY%
290 ENDPROC
```

8.2 Example 2 DOG

Simply moving a sprite to a different place on the screen isn't sufficient to make it look as though it is moving in many cases. Consider for example a dog running, to make this look realistic its paws would have to change position as well. This effect can be achieved by having two (or more) dog sprites defined which have their paws in different positions. These are then plotted alternately.

```
0 REM DOG
5 ONERRORVDU4:END
10 MODE 1
20 VDU23,1;0;0;0;0;
30 *SLOAD S.DOG
40 VDU19,2,2,0,0,0
50 GCOL0,130
60 CLG
70 C%=1
80 DX%=24
90 DY%=500
100 PROCdog
110 FORN%=24TO1000STEP8
120   DX%=N%+8
130   IFN%MOD200=0 PROCdogdown:ELSEPROCdog
140   FORM%=1TO200:NEXT
150   NEXT
160 REPEATUNTILFALSE
170
180 DEFPROCdog
190 C%=((C%-1)EOR1)+1
200 VDU23,27,0,C%,0,0,0,0,0,0
210 PLOT&ED,DX%,DY%
220 ENDPROC
230
240 DEFPROCdogdown
250 VDU23,27,0,3,0,0,0,0,0,0
260 PLOT&ED,DX%,DY%
270 FORM%=1TO5000:NEXT
280 ENDPROC
```

8.3 Example 3 SANTA

The following example demonstrates two new things; it shows how two different sprites can be used independently and it provides an alternative method of plotting and unplotting a shape. The new method of plotting uses `GCOL3,X` which Exclusive-ORs a shape on to the screen. This means that when a pixel is plotted the colour in which it appears depends on the colour of the pixel on top of which it is being plotted.

Consider the example of plotting a red pixel on top of a blue pixel in MODE 2

```
red  = 1 = 0001
blue = 4 = 0100
```

To Exclusive-OR (EOR) two numbers you combine their binary values in such a way that for any column if the two digits in it are the same the result is 0 and if they are different the result is 1. Hence in this example:

```
0001 EOR 0100 = 0101 = 5
```

Therefore, the pixel would appear to be in colour 5, ie magenta.

A result of this is that when a pixel is plotted on top of a pixel of the same colour they cancel out so a sprite can be deleted by plotting it twice in the same position.

The reason that this method is sometimes used is that when a sprite passes ‘in front of’ another shape on the screen this background shape is not deleted. One disadvantage is that if the background changes colour, the sprite likewise changes colour as it moves along, so a black and white dog would change into a green and magenta dog as it moved past a green bush. Also, it means that moving a sprite takes longer, since to change its position two plots have to be done; one on top of the old shape to cancel it out and then another in the new position to make it appear again.

In the example below the background remains the same colour so the first problem doesn't exist. In addition this new method of plotting has only been used on the parcels which are small and hence very quick to plot. If you try altering the program to plot the sleigh using this method as well you will notice that it will appear to flicker. This is because the shape is large and so takes a significant length of time to plot and unplot, consequently there are times when there is only part of a sleigh on the screen which makes it seem as though it is disappearing.

```
0 REM SANTA
5 ONERRORVDU4:END
10 MODE 2
20 VDU23,1;0;0;0;0;
30 *SLOAD S.SANTA
40 CLS
50 SX%=0
60 SY%=680
70 launch=FALSE
80 PROChouses
90 PROCsleigh
100 FORN%=0TO1400STEP8
110   SX%=N%
120   *FX19
130   SX%=N%+8
140   PROCsleigh
150   PROCparcel
160   FORM%=1TO200:NEXT
170   NEXT
180 REPEATUNTILFALSE
190
200 DEFPROCsleigh
210 VDU23,27,0,1,0,0,0,0,0,0
220 PLOT&ED,SX%,SY%
230 ENDPROC
240
250 DEFPROCparcel
260 GCOL3,0
270 VDU23,27,0,2,0,0,0,0,0,0
280 IF launch PLOT&ED,PX%,PY%:PY%=PY%-24:
PLOT&ED,PX%,PY%:IFPY%<160 PLOT&ED,PX%,PY%:launch=FALSE
290 IF SX%MOD240=184 PX%=SX%+16:PY%=680:launch=TRUE:PLOT&ED,PX%,PY%
300 GCOL0,0
```

```
310 ENDPROC
320
330 DEFPROChouses
340 VDU23,27,0,3,0,0,0,0,0,0
350 PLOT&ED,112,40
360 PLOT&ED,352,40
370 PLOT&ED,592,40
380 PLOT&ED,832,40
390 PLOT&ED,1072,40
400 ENDPROC
```

8.4 Example 4 ROCKETS

The two methods of plotting shapes which have been discussed so far can be combined so that the old shape is rubbed out and the new shape EORed on to the screen using one plot. This preserves the background and reduces the amount of flicker.

To do this a modified sprite is produced which consists of the sprite in the old position EOR the sprite in the new position. Then when this composite sprite is plotted the desired effect is achieved. The program contains the procedure `PROCGETIT` which produces the modified sprite itself by plotting the single copy of the rocket twice in the two positions and then defining the rectangular area of the screen which contains them to be the sprite which is to be used.

```
0 REM ROCKETS
5 ONERRORVDU4:END
10 MODE 5
20 VDU23,1;0;0;0;0;
30 GCOL3,7
40 *SLOAD S.ROCKETS
50 CLS
60 Y%=0
70 *SCHOOSE1
80 FORX%=200TO1000STEP200
90   PLOT&ED,X%,Y%
100   NEXT
110
120 *SCHOOSE2
130 REPEAT
140   FORX%=200TO1000STEP200
150     PLOT&ED,X%,Y%
160     NEXT
170   Y%=Y%+4
180   UNTILY%>1030
190 VDU4:END
200 DEFPROCGETIT
220 *SCHOOSE1
230 PLOT&ED,640,512
240 GCOL3,7
250 PLOT&ED,640,512+4
260 MOVE640,512
270 MOVE640+8*8-32,512+16*4-4
280 *SGET 2
290 ENDPROC
```

8.5 Example 5 LORRY

Another method of plotting a sprite and still retaining the background is used in the final example which shows a lorry moving along the road in front of hills and trees. In this case the lorry is plotted on the screen in the colours it was specified to have. The method uses the ability of the Graphics Extension ROM to take an area of the screen and convert it into a sprite. Before the lorry is plotted the area in front of it which it is about to be moved on to is defined to be a sprite. This is then stored for when the lorry has moved past this particular area and then it is re-plotted. Twenty-two sprites are used for holding the background, each of them being one byte wide. The plotting sequence is

1. Plot the sprite which contains the background which the back of the lorry is 'covering up'.

2. Redefine this sprite to contain the background of the strip in front of the lorry on to which it is about to move.

3. Plot the lorry in its new position.

A simpler way of doing this would be to just have one sprite for the background which was the same size as the lorry. Then all that would be necessary would be to plot the old background, save the background which the lorry was about to move on to and then plot the lorry. Unfortunately for a shape of the size of the lorry this would take too long since the lorry would be off the screen whilst the background sprites were being plotted and redefined and so the flicker would be very bad. In the method used the background sprites are much smaller so are plotted faster and the flicker is kept to a minimum.

```
0 REM LORRY
5 ONERRORVDU4:END
10 MODE 2
20 VDU23,1;0;0;0;0;
30 *SLOAD S.LORRY
40 C%=1
50 LX%=896:SX%=22
60 LY%=200:SY%=24
70 PROCbackground
80 FORI%=0TO21
90   PROCGET(I%,LX%+16*I%,0,LY%,SY%)
100   NEXT
110 PROClorry
120 T%=21
130 REPEAT
140   *FX19
150   VDU23,27,0,T%,0,0,0,0,0,0
```

```
160   PLOT&ED,LX%+21*16,LY%
170   MOVELX%-16,LY%
180   PLOT0,0,SY%*4
190   VDU23,27,1,T%,0,0,0,0,0,0
200   T%=T%-1:IFT%=-1 T%=21
210   LX%=LX%-16
220   VDU23,27,0,22,0,0,0,0,0,0
230   PLOT&ED,LX%,LY%
240   UNTILLX%<-SX%*16
250 REPEATUNTILFALSE
260
270 DEFPROCGET(S%,X1%,DX%,Y1%,DY%)
280 MOVEX1%,Y1%
290 MOVEX1%+DX%*16,Y1%+DY%*4
300 VDU23,27,1,S%,0,0,0,0,0,0
310 ENDPROC
320
330 DEFPROClorry
340 *SCHOOSE 22
350 PLOT&ED,LX%,LY%
360 ENDPROC
370
380 DEFPROCbackground
390 GCOL0,128+6
400 COLOUR128+6
410 CLS
420 x%=0:y%=400:dy%=1
430 MOVE0,y%
440 GCOL0,0
450 FORI%=1TO40
460   FORJ%=0TO3
470     x%=x%+8
480     y%=y%+dy%*4
490     DRAWx%,y%
500     NEXT
510   dy%=4-RND(7-y%DIV700+y%DIV400)
520   NEXT
530 GCOL0,2
540 GCOL0,128+6
550 PLOT&85,10,4
560 GCOL0,0
570 MOVE0,0
580 PLOT&65,1272,240
```

```
590 PROCtree(RND(100)+200)
600 PROCtree(RND(100)+500)
610 PROCtree(RND(100)+800)
620 ENDPROC
630
640 DEFPROCtree(tx%)
650 VDU23,12,1,0,0,1,1,0,0,1
660 GCOL16,0
670 MOVEtx%,240
680 PLOT&65,tx%+16,280
690 VDU23,13,2,1,1,2,2,1,1,2
700 GCOL32,0
710 MOVEtx%+8,320
720 PLOT&9D,tx%+16,280
730 ENDPROC
```

Appendix A

Useful *FX/OSBYTE calls

`*FX 163,242,0` sets default dot pattern and length

`*FX 163,242,1..64` set specified repeat length (1..64)

OSBYTE with `A=163`, `X=242`, `Y=65` returns GXR status in `X` and `Y`.

If the ROM is off `X=Y=0`
If the ROM is on `X=gfbbbbbb`, `Y`=number of sprite pages.
`g=1` ROM on
`g=0` ROM off
`f=1` FLOOD on
`f=0` FLOOD off
`bbbbbb=` dot-dash repeat length MOD 64

OSBYTE with `A=163`, `X=242`, `Y=66` returns size of currently selected sprite in `X` and `Y`.

If the ROM is off then `X=0`, `Y=0` on return.
If the ROM is on then `X=0`, `Y=0` if no sprite selected,
otherwise `X`=width, `Y`=height of current sprite in bytes.

Appendix B

The *HELP features

The Graphics Extension ROM provides a comprehensive set of responses to `*HELP` commands. These are divided into two parts:

The command `*HELP GRAPHICS` responds with a list of all the graphics commands and plot codes, along with the `VDU23` and `*FX163` calls, and the `GCOL` parameters. For reference, the output is as follows:

```
Graphics Extension ROM 1.20
  GXR On, Flood On

  GXR commands     (use * to precede these)
    GXR
    NOGXR
    FLOOD
    NOFLOOD

  Plot codes
    00   Line
    08   Line (LPO)          (LPO is Last Point Omitted)
    10   Dot-dash (R)        (R is pattern Restarted with new line)
    18   Dot-dash (R,LPO)    (FPO is First Point Omitted)
    20   Line (FPO)          (BEO is Both Ends Omitted)
    28   Line (BEO)          (C is pattern Continues with new line)
    30   Dot-dash (C,FPO)
    38   Dot-dash (C,BEO)

    40   Point
    48   Fill L&R to Non-bg
    50   Triangle
    58   Fill R to bg
    60   Rectangle
    68   Fill L&R to fg
    70   Parallelogram
    78   Fill R to Non-fg

    80   Flood to Non-bg
    88   Flood to fg
    90   Circle outline
    98   Circle fill
  A0   Circular arc
  A8   Circular segment
  B0   Circular sector
  B8   Block copy/move

  C0   Ellipse outline
  C8   Ellipse fill
  D0
  D8
  E0
  E8   Sprite plot
  F0
  F8

Set dot-dash repeat length
  *FX 163,242,k : k=0,1-64

Define patterns
  VDU 23,k,b,b,b,b,b,b,b,b with k=
  2,3,4,5 : full setting
  6       : dot-dash line
  11      : default setting
  12,13,14,15 : simple setting

  VDU 23,27,0,n,0;0;0;
    Choose sprite n for plotting
  VDU 23,27,1,n,0;0;0;
    Get sprite n from screen

Select colour pattern
  GCOL a,c
  a<16 solid colour c
  a=16-21 : pattern 1
  a=32-37 : pattern 2
  a=48-53 : pattern 3
  a=64-69 : pattern 4
```

The command *HELP SPRITES responds with the current sprite status and a list of the sprite commands and their parameters, as follows:

```
Sprite status
  0 page(s) sprite w/s

Sprite commands     (use * to precede these)
  SSPACE n
  SCHOOSE n
  SDELETE n
  SEDIT n
  SEDIT n,m
  SGET n
  SLOAD filename
  SMERGE filename
  SNEW
  SRENUMBER n,m
  SSAVE filename
```

Index

SBL25/B