13

GRAPHICS

This chapter is all about fast shape drawing. Like everything else, shapes such as space invaders are stored as a series of numbers in the computer's memory This chapter consists of two main sections. The first contains a routine which is designed to be used independent of any other program. It allows a shape to be designed, and produces the relevant numbers representing that shape. The second contains two routines which are meant to be included as part of an assembler program. While the program is running, these routines convert the numbers in the memory back into the original shape and plot this shape on the screen at a stated position. These routines have been highly optimised for fast animation.

Some knowledge of graphics is assumed, up to User Guide level. Those of you without this knowledge are recommended to read the Acornsoft book, Creative Graphics, which gives a clear illustration of the graphics facilities available on the BBC Microcomputer and Acorn Electron.

13.1 Shape designer - DESIGN

The BASIC program below can be used to design hapes which can be plotted on the screen using the

routines described later in this chapter. Remember that smaller shapes are plotted faster so if vou wish to move several shapes on the screen at once it is better not to make them too large. The program is only suitable for a BBC Model B, or Electron.

The keys used by this program are:

0,1,..,E,F	Colours 0 to 15
V and H	Vertical and horizontal reflections
L and S	Load and Save shapes
Cursor keys	For moving the cursor around
SY%	Height of 'pixels' in large shape.
shape	Byte vector to hold shape.
cursor X	X co-ordinate of cursor in 'pixels'.
cursor Y	Y co-ordinate of cursor in 'pixels'.
command$	String containing commands.
	String containing keys for colours.
colour$	Colours are specified by presssing key corresponding to Hexadecimal digit. Note state of CAPS LOCK or SHIFT keys is irrelevant.
key$	Holds key pressed.
length	Holds length of shape data.
C%	Zero if command, else holds colour key plus one.

```
   0 REM Design Program
  10
+-----------------------------------------------------+
|Top Level. Note that FNinitialise returns the mode   |
|as MODEs cannot be defined in FNs or PROCs           |
+-----------------------------------------------------+
  20 MODE FNinitialise
  30 PROCmainloop
  40
+-----------------------------------------------------+
|Set up global variables, especially those that are   |
|relevant to the screen mode selected.                |
+-----------------------------------------------------+
  50 DEF FNinitialise
  60 lenshape 300
  70 DIM shape lenshape
  80 REPEAT
  90   INPUT "Mode ( 0 - 2 ) ?" mode
 100 UNTIL mode >=0 AND mode <=2
 110 RESTORE 230
```

```
 120   FOR I% = 0 TO mode
 130 READ pixbits
 140 NEXT I%
 150 RESTORE 240
 160 FOR I% = 0 TO mode
 170   READ W%
 180 NEXT I%
 190 RESTORE 250
 200 FOR I% = 0 TO mode
 210   READ pixelperbyte
 220 NEXT I%
 230 DATA 1, 2, 4
 240 DATA 2, 4, 8
 250 DATA 8, 4, 2
 260 INPUT "Width in X-direction:" NX%
 270 INPUT "Width in Y-direction:" NY%
 280 byteNX% = NX%DlVpixelperbyte
 290 byteNY% = NY%-1
 300 SX% = (1024 DIV NX%) AND &FFF0
 310 SY% = (1024 DIV NY%) AND &FFF8
 320 PROCclear
 330 cursorX = 0 : cursorY = 0
 340 *FX 4 1
 350 command$ = "VvHhLlSs" + CHR$&88 + CHR$&89 +
                                  CHR$&8A + CHR$&8B
 360 colour$ = "001!2""3#4$5%6&7'8(9)AaBbCcDdEeFf"
 370 length = NX% * NY% * pixbits DIV 8
 380 = mode
 390
+-------------------------------------------------+
|Main loop. Get a key, and then test to see if it |
|is legal. If the key is legal, then pass it on to|
|the rest of the routine, which then calls the    |
|relevant routine(s).                             |
+-------------------------------------------------+
 400 DEF PROCmainloop
 410 PROCcursor
 420 REPEAT
 430   REPEAT
 440     keyS = GETS
 450   UNTILINSTR(colour$,key$) OR INSTR(command$,key$)
 460   C% = (INSTR(colour$, key$) + 1) DIV 2
 470   IF C% THEN PROCcolour(C%-1) ELSE ON
   INSTR(commandS, key$) GOSUB 670, 670, 780, 780,
```

```
   630, 630, 650, 650, 510, 540, 570, 600
 480 UNTIL FALSE
 490 ENDPROC
 500
+-----------------------------------------------------+
|Handle cursor control keys, making sure that the     |
|cursor does not go off the side of the screen.       |
+-----------------------------------------------------+
 510 IF cursorX > 0 THEN PROCdocursor(-1,0)
 520 RETURN
 530
 540 IF cursorX < NX% - 1 THEN PROCdocursor(1,0)
 550 RETURN
 560
 570 IF cursorY > 0 THEN PR0Cdocursor(0,-1)
 580 RETURN
 590
 600 IF cursorY < NY% - 1 THEN PR0Cdocursor(0,1)
 610 RETURN
 620
+-----------------------------------------------------+
| Handle saving/loading of shapes                     |
+-----------------------------------------------------+
 630 PROCload : VDU 22, mode : PR0Cshape(FALSE,
          byteNX%,byteNY%) : PR0Cdisplay : RETURN
 640
 650 PROCsave : VDU 22, mode : PROCshape(FALSE,
          byteNX%,byteNY%) : PROCdisplay : RETURN
 660
+-----------------------------------------------------+
| Reflect the shape in Y=maximum Y / 2                |
+-----------------------------------------------------+
 670 FORI%=0 TO NX%-1
 680   FORJ%=0 TO (NY%-1)DIV2
 690     temp=FNpoint (I%,J%)
 700     PROCplot(I%,J%,FNpoint(I%,NY%-1-J%))
 710     PROCplot(I%,NY%-1-J%,temp)
 720   NEXTJ%
 73C NEXTI%
 740 PROCshape(TRUE,byteNX%,byteNY%)
 750 PROCdisplay
 760 RETURN
 770
```

```
+-----------------------------------------------------+
| Reflect the shape in X maximum X / 2                |
+-----------------------------------------------------+
 780 FORJ%=0 TO NY%-1
 790   F0RI%=0 TO (NX%-1)DIV2
 800     temp=FNpoint(I%,J%)
 810     PROCpIot(I%,J%,FNpoint(NX%-1-I%,J%))
 820     PROCplot (NX%-1-I%,J%,temp)
 830   NEXTI%
 840 NEXTJ%
 850 PROCshape(TRUE,byteNX%,byteNY%)
 860 PRoCdisplay
 870 RETURN
 880
 890 DEF PROCplot(X,Y,col)
 900 GCOL 0,col
 910 PLOT 69, X * W% + 1024, Y * 4 + 640
 920 ENDPROC
 930
 940 DEF FNpoint(X,Y)
 950 = POINT(X * W% + 1024, Y * 4 + 640)
 960
+-----------------------------------------------------+
| Plot a large square at the cursor position, looking |
| at the final size version at the side of the screen |
| to find the colour.                                 |
+-----------------------------------------------------+
 970 DEF PROCsq(X, Y)
 980 GCOL 0, FNpoint(X,Y)
 990 MOVE X * SX%, Y * SY%
1000 PLOT 0, SX% - 4, 0
1010 PLOT 81, 4 - SX%, SY% - 4
1020 PLOT 81, SX% - 4, 0
1030 ENDPROC
1040
+-----------------------------------------------------+
| Display whole shape                                 |
+-----------------------------------------------------+
1050 DEF PROCdisplay
1060 LOCAL X, Y
1070 FOR X = 0 TO NX% - 1
1080   FOR Y = 0 TO NY% - 1
1090     PROCsq(X,Y)
1100   NEXT Y
```

```
1110 NEXT X
1120 PROC cursor
1130 ENDPROC
1140

+-----------------------------------------------------+
| Plot the box cursor at the cursor position          |
+-----------------------------------------------------+
1150 DEF PROCcursor
1160 VDU 5
1170 GCOL 3,3

1180 MOVE SX% * (cursorX + .25), SY% * (cursorY + .25)
1190 PLOT 1, SX% DIV 2 - 4, 0
1200 PLOT 1, 0, SY% DIV 2 - 4
1210 PLOT 1, 4 - SX% DIV 2, 0
1220 PLOT 1, 0, 4 - SY% DIV 2
1230 ENDPROC
1240
+-----------------------------------------------------+
| Get a shape from the filing system                  |
+-----------------------------------------------------+
1250 DEF PROCload
1260 LOCAL I%, channel
1270 PROCshape(TRUE,byteNX%,byteNY%)
1280 channel = FNopen(TRUE)
1290 IF channel = FALSE ENDPROC
1300 PROCclear
1310 FOR I% = 0 TO (byteNX% * NY%)-1
1320    shape? I% = BGET#channel
1330 NEXT I%
1340 CLOSE #channel
1350 ENDPROC
1360
+-----------------------------------------------------+
| Write a shape to the filing system                  |
+-----------------------------------------------------+
1370 DEF PROCsave
1380 LOCAL I% , channel
1390 PROCshape (TRUE, byteNX% ,byte NY%)
1400 channel = FNopen(FALSE)
1410 IF channel = FALSE ENDPROC
1420 FOR I% = 0 TO (byteNX% * NY%) -1
1430   BPUT#channel,shape? I%
1440 NEXT I%
1450 CLOSE #channel
```

```
1460 ENDPROC
1470
+-----------------------------------------------------+
| Utility used by PROCload and PROCsave to open a file|
+-----------------------------------------------------+
1480 DEF FNopen(in)
1490 LOCAL W$
1500 VDU 22,7
1510 INPUT "File name ?"W$
1520 IF W$ = "" = FALSE
1530 IF in THEN = OPENIN(W$) ELSE OPENOUT(W$)
1540
+-----------------------------------------------------+
| Plot point on final size shape and also plot        |
| square on main screen.                              |
+-----------------------------------------------------+
1550 DEF PROCcolour(colour)
1560 PROCcursor
1570 PROCplot(cursorX,cursorY,colour)
1580 PROCsq(cursorX,cursorY)
1590 PROCcursor
1600 ENDPROC
1610
+-----------------------------------------------------+
| Wipe previous cursor from screen update cursor's    |
| X and Y coordinates, and then plot the cursor at    |
| new coordinates                                     |
+-----------------------------------------------------+
1620 DEF PROCdocursor(X,Y)
1630 PROCcursor
1640 cursorX = cursorX + X
1650 cursorY = cursorY + Y
1660 PROCcursor
1670 ENDPROC
1680
+-----------------------------------------------------+
| Clear the array used to hold the shape              |
+-----------------------------------------------------+
1690 DEF PROCclear
1700 LOCAL clear
1710 FOR clear = 0 TO lenshape-4 STEP 4
1720   clear!shape=0
1730 NEXT clear
1740 ENDPROC
```

```
1750
+-------------------------------------------------------+
| Either write final size shape from shape array or     |
| put final size shape into shape array                 |
+-------------------------------------------------------+
1760 DEFPROCshape(flag, X, Y)
1770 LOCAL I%, J%, tempx, tempy
1780 FOR I% = 0 TO X
1790   FOR J% = 0 TO Y
1800     tempx=I%+1024 DIV 16
1810     tempy=((J%+640 DIV 4)EOR&FF)DIV8
1820     PROCaccess(flag, (tempy*&280+tempx*8+((J%
                  AND 7)EOR 7)+&3000), Y+1, J%, I%)
1830   NEXT J%
1840 NEXT I%
1850 ENDPROC
1860
+-------------------------------------------------------+
| Get/put byte from/to final shape.                     |
+-------------------------------------------------------+
1870 DEF POCaccess(flag,addr , Y , J% , I%)
1880 IF flag THEN shape/(I%*Y+(Y-J%-1))=?addr ELSE
                  ?addr=shape?(I%*Y+(Y-J%-1))
1890 ENDPROC
```

Variables Used:

lenshape	Maximum length of shape (in bytes).
mode	Holds graphics modeseected.
I%	General loop control variable.
J%	General loop control variable.
pixbits	Number of bits per pixel.
pixelperbyte	Holds number of pixels per byte.
W%	Width of pixels in graphics co-ordinates
NX%	Width of shape
NY%	Height of shape
byteNX%	Width of shape (in bytes)
byteNY%	Height of shape (in bytes)
SX%	Width of pixels In larga shape
SY%	Height of pixels in large shape.
shape	Byte vector to hold shape.
cursorX	X co-ordinate of cursor in pixels.
cursorY	Y co-ordinate of cursor in ' pixels' .
command$	String containing commands.
colour$	String containing keys for colours
	Colours are specified by pressing key

	corresponding to Hexadecimal digit. Note state of CAPS LOCK or SHIFT keys is irrelevant.
key$	Holds key' pressed.
length	Holds length of shape data.
C%	Zero if command, else holds colour key plus one.

13.2 Plotting a shape on the screen

To plot a shape on the screen at a specified position it is necessary to have two routines; one to convert the X and Y coordinates to a memory location on the screen, and another routine to plot a shape at that address.

Two routines are given below which perform these tasks. The method chosen can only be used for plotting shapes to a resolution of 80 by 256 in MODEs 0, 1 and 2. The routines work by Exclusive ORing the shape onto the screen. This has two main advantages over other methods (such as writing the shape on the screen or ORing the shape with the screen memory). The first advantage is that the detail under the shape is not lost when the shape is unplotted, the second is that the same routine can be used for both plotting and unpotting.

Convert X, Y coordinate to screen address -- getaddr

This routine doesn' twrite anything to the screen, all that it does is generate an address where a shape might then be written to the screen, or read from the screen. It will generate an address for MODEs 0, 1 and 2, allowing for hardware scroll. The algorithm used is given at the end of the listing so that the code can be adapted for MODEs 4 and 5.

On entry, X holds the X coordinate (0 79), Y holds the Y coordinate (0 - 255), and A is irrelevant.

A typical call might be:

```
LDX xcoord
LDY ycoord
JSR getaddr
```

On exit, X will be preserved, Y and A will have been corrupted. The resultant address is left in ' addr'and ' addr+1(low byte, high byte) which must be in zero page. Other locations used are ' temp'(1 byte) and

' top'(2 bytes). For speed, these locations should also
be in zero page.

```
.getaddr
   LDA #&00              Set hi byte of address
   STA addr+1            to zero
   TYA                   Invert Y coordinate
   EOR #&FF              and save to stack
   PHA
   OPT FN rotateacc(3)   Divide Y cordinate by 8
   TAY                   and leave in Y
   LSR A                 Adjust carry for * &280
   STA temp              Save Y/16 in temp
   LDA #&00              Set bottom byte of address to
0
   ROR A                 Put carry into top bit
   ADC top               and add in top of screen
   PHP                   Save carry flag
   STA addr              Store result in addr
   TYA                   Get Y/8
   ASL A                 Double it for top byte
   ADC temp              of addr. Add in Y/16
   PLP                   Restore carry
   ADC top+1             and add in top of screen
   STA addr+1
   LDA #&00              Set temp to zero
   STA temp
   TXA                   Get X coordinate
   ASL A                 Perform two byte multiply
   ROL temp              by 8 because of memory
   ASL A                 map
   ROL temp
   ASL A
   ROL temp
   ADC addr              Add in rest of result so
   STA addr              far, and store it
   LDA temp
   ADC addr+1
   BPL ok                Check for hardwar scroll
   SEC                   If over 3000-8000 boundary
   SBC #&50              then correct address
.ok
   STA addr+1            And store it
   PLA                   Restore inverted Y coord
   AND #&07              Get row number in computed
   ORA addr              column, and add it in
```

```
STA addr
RTS                    Return
```

Some words of explanation:

The algorithm used to calculate the screen memory address is:

```
addr = X * 8 + ((Y EOR &FF)
AND 7) + &280 * ((Y EOR &FF
)DIV 8)+ top
```

where X and Y are the coordinates. The reason Y is inverted is so that the bottom of the screen is treated as 0, even though the memory map is the other way round. There are 640 bytes per character line on the display, which is &280 in hex. The variable ' top'is normally set to &3000, except when hardware scroll is taking place. In most applications this will be irrelevant, and so ' topmay be set up at the beginning of the program and then forgotten about. The Y AND 7 and Y DIV 8 operations are performed because of the memory map of the screen, DIV 8 is performed to get to the start of the current character cell, and AND 7 to get to the current byte in the character cell:

```
top   +0      +8      +&10    +&18    +&20    +&28
+0    +-------+-------+-------+-------+-------+.....
+1    +-------+-------+-------+-------+-------+.....
+2    +-------+-------+-------+-------+-------+.....
+3    +-------+-------+-------+-------+-------+.....
+4    +-------+-------+-------+-------+-------+.....
+5    +-------+-------+-------+-------+-------+.....
+6    +-------+-------+-------+-------+-------+.....
+7    +-------+-------+-------+-------+-------+.....
+&280 +-------+-------+-------+-------+-------+.....
+&281 +-------+-------+-------+-------+-------+.....
      .       .       .       .       .       .
      .       .       .       .       .       .
```

Plotting a shape at a given screen coordinate - doplot
This routine works in conjunction with the above routine for plotting a shape at a given X, Y screen coordinate.

On entry, all registers are irrelevant. The parameters passed are:

counter -- holds number of bytes in the shape
addr -- holds screen address to put shape
depth -- holds height of shape
shape -- start address of shape in memory

(' shape' must be in zero page.)

On exit, all registers may have been corrupted, but all parameters will have been preserved.

```
.doplot
   LDY #&00          Set shape offset to zero
   LDA addr+1        Push screen sddress onto
   PHA               stack for later use
   LDA addr
   PHA
   LDA depth         Get depth of shape
   STA rowcounter
   LDA addr          Put offset in character
   AND #&07          cell into Y
   STA offset
   LDA addr          Adjust address accordingly
   AND #&F8          Go to top of character cell
   STA addr
   STY temp
.innerloop
   LDY temp
   LDA (shape),Y     Get byte from shape
   INY
   STY temp
   LDY offset
   EOR (addr),Y
   STA (addr),Y
   INY               Y holds offset on screen
   CPY #&08          bottom of character cell
   BEQ block         if so then go down a line
.nobIoc
   STY offset
   DEC rowcounter
   BNE innerloop
.nextblock
   LDA shape

   CLC
   ADC depth
   STA shape
```

```
   BCC nohi
   INC shape+1
.nohi
    CLC                Go to the top of the next
column
   PLA                 by resetting address to top
   ADC #&08            of current column, and
   STA addr            moving to nect character
   PLA                 cell
   ADC #&00
   BPL nobound1

   SEC
   SBC #&50
.nobound1
   STA addr+1
   DEC counter         Easier to DEC counter in
   BNE doplot          two place and test
   RTS
.block
   LDY #&00            Go down a line
   LDA addr            addr = (addr + &280)
   CLC
   ADC #&80
   STA addr
   LDA adr + 1
   ADC #&02
   BPL noboundary      If the contents of addr are
   SEC                 greater than &8000 then
   SBC #&50            subtract &5000
.noboundary
   STA addr + 1
   BNE nolock          Always jump
```

Interfacing getaddr and doplot - plotshape

The number of parameters may be cut down by having a ' frontend' attached to the start of the doplot routine which would make interfacing easier.

On entry to ' plotshape'A would hold the number of the shape to be plotted and X and Y would hold the X and Y coordinates at which the shape is to be plotted.

A typical call to ' plotshape' would be:

```
LDY #0              Get first shape
```

```
TYA
PHA                 Preserve shape number
LDA xcoord,Y        Get x coordinate
TAX
LDA ycoord,Y        Get y coordinate
TAY
PLA                 Get shape number back
JSR plotshape       Plot shape
```

On exit from ' plotshape'all registers may have been corrupted, although all internal parameters (' depth' , ' counter' , etc.) will have been preserved.

```
.plotshape
   PHA                      Save shape number
   JSR                      Getaddr Get address
   PLA                      Restore shape
   TAY
   LDA shapeloaddr,Y        Set up parameters
   STA shape                This assumes that
   LDA shapehiaddr,Y        the User has set
   STA shape + 1            up the relevent
   LDA shapesize,Y          tables
   STA counter              (shape loaddr,
   LDA shapedepth,Y         shapesize, etc)
   STA depth
```