Graphics Workshop

DOS 3.3
○
0

DOUBLE HI-RES GRAPHICS II

The Graphics Workshop explores block shape animation on the Double Hi-Res screen in Part II of the Double Hi-Res series. A machine language driver and several demonstration programs show how it's done.

by Robert R. Devine
Small Computer Services
P.O. Box 10
Adona, AR 72001

n the May issue we laid all the needed groundwork for Double Hi-Res. This month we'll develop more routines for our DHR DRIVER, and begin to get into some animation techniques.

If you've been conducting your own experiments, you've probably found that there is a lot of potential in Double Hi-Res; however, graphics animation from Applesoft is rather cumbersome and slow. To avoid the peculiarities of Double Hi-Res, the routines that we'll develop will automatically handle all the soft switch flipping for us, therefore making it unnecessary for us to worry about columns, duplicate addresses, and the like.

Let's get to work.

Double Hi-Res Block Shapes

Figure 1 is a representation of the first shape that we'll work with. This is the same alien spaceship that we used earlier in the Graphics Workshop series; however, this shape is defined in such a way that it will work on the Double Hi-Res screen.

Note that the shape is six bytes/columns wide, but only three addresses wide. In regular 280-dot Hi-Res, the width of a block shape was defined by the number of horizontal bytes it occupied. In Double Hi-Res block shapes, the width is defined by the number of addresses that it occupies; therefore, all Double Hi-Res block shapes will be an even number of columns wide. The shape in **Figure** 1 has a width of 3 and a height of 14. The total size of the shape is 3 * 14 * 2 = 84 bytes.

I have approached the shape width in terms of addresses (rather than bytes) to avoid the need to constantly check every byte of data to see which way the page 2 soft switch needs to be set. As it is now, our Double Hi-Res shapes will require twice as many data bytes as would be required by the same size 280-mode shapes. This already means that our drawing routines will need to do twice the work, so we want to avoid as much checking activity as possible to maintain maximum speed. By establishing a consistent shape definition and data format, no checking activities are needed.

"...graphics animation from Applesoft is rather cumbersome and slow."

Defining a Block Shape

Each of our shapes will be defined with five values.

SHape NUMber (POKE 251,SHNUM)
Each of our shapes will have a number which is stored in memory location 251 ($FB). This value will tell our drawing routines where to find the data that defines the shape.

The normal way of storing shapes in memory is to begin at the top of available memory (just below the driver) and build downward with each additional shape. Each of the shapes will begin at the very first byte of a memory page; i.e., $7500, $8A00, $9000, etc. There are also ways that you can store multiple shapes on a memory page...we'll get into that later. **If your shape begins at the first byte of a memory page**, you may let it overflow onto the next page; therefore, there is no maximum shape length that you need to worry about.

To determine the proper shape number, take the first two digits of the hex starting address and convert those digits to their decimal value. For example, let's use a shape that you are going to store in memory beginning at $9000. The first two digits of $9000 are $90, and since 144 is the decimal equivalent of $90, your shape number will be 144.

Vertical Top (POKE 252,VT)
The value of VT will be the topmost Y-coordinate that your shape occupies (0-191).

Vertical Bottom (POKE 253,VB)
The value of VB will be the lowermost Y-coordinate that your shape occupies (0-191).

Horizontal Right (POKE 254,HR)
The value of HR will be the rightmost address offset that your shape occupies (0-39).

Horizontal Left (POKE 255,HL)
The value of HL will represent the leftmost address offset that your shape occupies (0-39).

Every time we manipulate our shape on the screen, we will specify the VT, VB, HR, and HL of the shape to define the portion of the screen in which our animation routines are to perform their activities. The value of SHNUM will be used with any routines that use the Shape Definition Table which is stored in memory.

As you look at **Figure 1**, you will see that the current VT, VB, HR, and HL values for our shape are 0, 13, 2, and 0, respectively.

What Is a Block Shape?

A **block shape** is a rectangular "block" of Hi-Res screen bytes which is bounded on the top and bottom by VT and VB, and bounded on the sides by HR and HL. A **Block Shape Table** is a sequential string of data bytes (in our example there are 84) which contains the bit patterns for each byte within the rectangle.

Our animation and drawing routines step through the table, element by element, and place the proper bit pattern into the proper Hi-Res bytes within the defined rectangle. Our Shape Table contains no information indicating where it begins or ends; therefore, our animation routines will continue to manipulate screen bytes until they have dealt with all bytes within the bounds of VT, VB, HR, and HL.

If the dimensions that you have set to define the rectangle do not conform to the Shape Table data (the way the shape was created), then the shape will be incomplete or distorted. You will note that many of the bytes within the rectangle have nothing at all to do with the shape itself, and are in fact part of the background; however, since they fall within the shape's area of influence, they are necessary parts of the block shape.

For more information on block shapes, see "Graphics Workshop: Block Shapes, Part 1," *Nibble* Vol. 4/No. 3.

How to Create a Block Shape

Block shapes are probably the easiest of all shapes to create. Rather than having to figure out a lot of data values or vector moves that go into each byte, all you need to do is draw your shape on the screen, using any method you like, and then use a routine that's built into the driver which will translate your drawing into the needed Block Shape Table.

If you've been following the Graphics Workshop series, then you can also use the BLOCK SHAPE MAKER program which ap-

FIGURE 1: THE ALIEN SPACESHIP

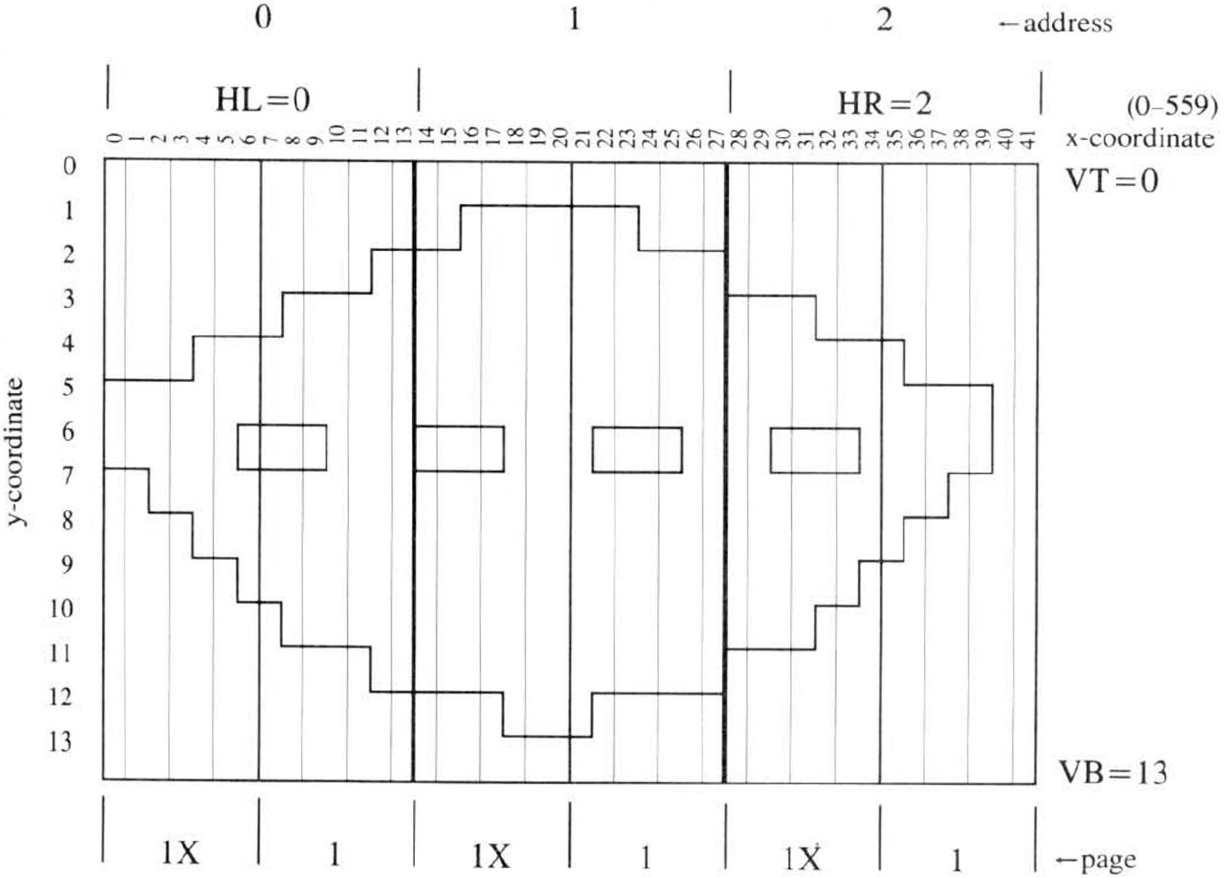
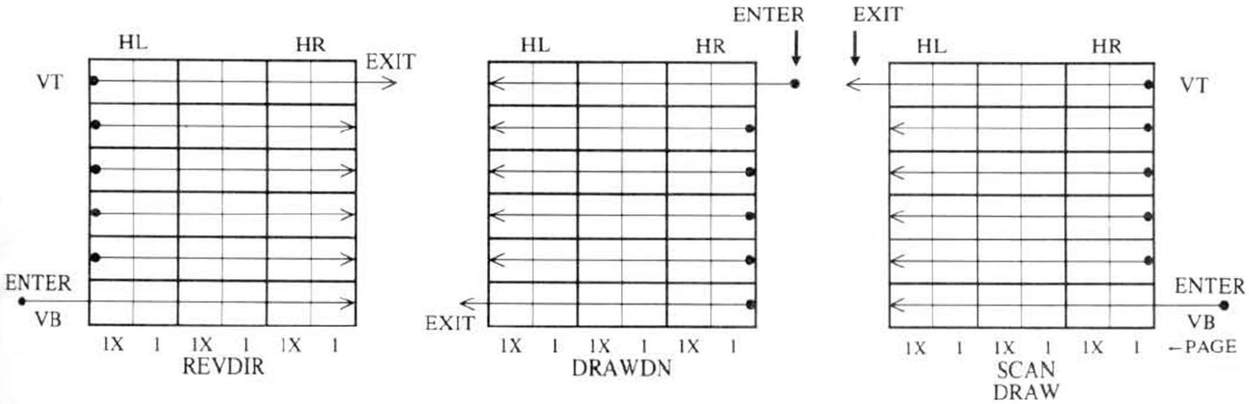


FIGURE 2: SHAPE PROCESSING METHODS



peared in *Nibble* Vo. 4/No. 5 for the creation of your shapes. Bear in mind that BLOCK SHAPE MAKER creates shapes on the regular Hi-Res screen, so you'll need to be sure and make your shapes an even number of bytes wide (the HR-HL dimension).

If you are planning to use color in your shapes, be sure to use the proper sets of four dots for color. Your shape will need to appear twice as wide on the regular Hi-Res screen as it will when you use it later on the Double Hi-Res screen.

More Routines for the DHR Driver

At this point it's going to be a bit difficult to try any animation tests until you have some new routines to work with, so let's continue building our DHR driver. The first thing to do is **BLOAD DHR.DRIVER** which we developed in the May issue. (See **Listing 0.5** for the hex dump of DHR.DRIVE if you're joining us this month. Key it in before adding SCAN, etc.) Then enter the Monitor and we'll add some new routines.

SCAN $93DA

The SCAN routine (**Listing 1**) is one of the most important routines, as it is the part of the driver that creates the shapes that you will use. Its function is to look at the shape which you have drawn on the screen (using HPLOTs or whatever), take the data patterns from the screen, and create a Block Shape Table. To use SCAN you must specify VT, VB, HR, and HL to tell SCAN which area of the screen it is to use in creating the Shape Table. You must also **POKE 251,SHNUM** to tell the routine where in memory you want the Shape Table to be assembled and stored.

DRAW $9394

The DRAW routine (**Listing 2**) is exactly the opposite of SCAN. It takes the data from the Shape Table in memory and places the proper values directly on the Double Hi-Res screen. Each time the routine changes to a new Hi-Res screen address, it places the first data byte in the odd column (page 1) and then places the next data byte in the even column (page 1X).

LISTING 1: THE SCAN ROUTINE

```
                 Ø1ØØ • SCAN ROUTINE
                 Ø11Ø • COPYRIGHT 1984 BY MICROSPARC, INC.
                 Ø12Ø •
                 Ø13Ø • S-C ASSEMBLER
                 Ø14Ø •
                 1ØØØ  .OR $93DA
                 1Ø1Ø  .TF SCAN $93DA.OBJ
ØØFC-            1Ø2Ø VT .EQ $FC           •• DECIMAL 252
ØØFD-            1Ø3Ø VB .EQ $FD           •• DECIMAL 253
ØØFE-            1Ø4Ø HR .EQ $FE           •• DECIMAL 254
ØØFF-            1Ø5Ø HL .EQ $FF           •• DECIMAL 255
ØØ26-            1Ø6Ø HBASL .EQ $26        •• DECIMAL 38  (SCREEN BASE
ØØ27-            1Ø7Ø HBASH .EQ $27        •• DECIMAL 39  ADDRESS)
ØØØ6-            1Ø8Ø YO .EQ $6            •• DECIMAL 6
ØØFA-            1Ø9Ø BASL .EQ $FA         •• DECIMAL 25Ø (TABLE BASE
ØØFB-            11ØØ BASH .EQ $FB         •• DECIMAL 251 ADDRESS)
9464-            111Ø YADDR .EQ $9464      •• DECIMAL 37988 (READ YTABLE)
CØ54-            1114 PAGE1 .EQ $CØ54
CØ55-            1116 PAGE1X .EQ $CØ55
93DA- A9 ØØ      112Ø SCAN LDA #Ø          •• SCANNER CALL 3785Ø TO ENTER
93DC- 85 FA      113Ø  STA BASL            •• POINT TO START OF TABLE
93DE- A5 FD      114Ø  LDA VB              •• GET BOTTOM Y COORDINATE
93EØ- 85 Ø6      115Ø  STA YO              •• STORE IN $6 FOR USE BY YADDR
93E2- 2Ø 64 94   116Ø L1 JSR YADDR         •• RETURNS-LO=HBASL/HI=HBASH
93E5- A4 FE      117Ø  LDY HR              •• SET Y-REG TO RIGHTMOST BYTE
```

```
93E7- A2 ØØ      118Ø  LDX #Ø              •• SET TABLE OFFSET=Ø
93E9- 8D 54 CØ   119Ø L2 STA PAGE1         •• READ MAIN MEMORY
93EC- B1 26      1195  LDA (HBASL),Y       •• GET SHAPE BYTE FROM SCREEN
93EE- 81 FA      12ØØ  STA (BASL,X)        •• PUT IN SHAPE TABLE
93FØ- E6 FA      123Ø  INC BASL            •• POINT TO NEXT TABLE ELEMENT
93F2- DØ Ø2      124Ø  BNE J1              •• IF ×256 BYTES-JUMP
93F4- E6 FB      125Ø  INC BASH            •• PAGE OVERFLOW-GOTO NEXT PAGE
93F6- 8D 55 CØ   1252 J1 STA PAGE1X        •• READ AUXILIARY MEMORY
93F9- B1 26      1253  LDA (HBASL),Y       •• GET SHAPE BYTE FROM SCREEN
93FB- 81 FA      1254  STA (BASL,X)        •• PUT IN SHAPE TABLE
93FD- E6 FA      1255  INC BASL            •• POINT TO NEXT TABLE ELEMENT
93FF- DØ Ø2      1256  BNE NC1             •• IF ×256 BYTES-JUMP
94Ø1- E6 FB      1257  INC BASH            •• PAGE OVERFLOW-GOTO NEXT PAGE
94Ø3- 88         1258 NC1 DEY              •• POINT TO NEXT BYTE ×---
94Ø4- CØ FF      126Ø  CPY #$FF            •• HAS Y-REGISTER REACHED Ø ?
94Ø6- FØ Ø4      127Ø  BEQ NXTLN           •• YES-GOTO NEXT LINE
94Ø8- C4 FF      128Ø  CPY HL              •• IS Y-REGISTER >=HL ?
94ØA- BØ DD      129Ø  BCS L2              •• YES-GET THE NEXT BYTE
94ØC- C6 Ø6      13ØØ NXTLN DEC YO         •• MOVE UP TO NEXT LINE
94ØE- A5 Ø6      131Ø  LDA YO              •• GET NEW Y COORDINATE
941Ø- C9 FF      132Ø  CMP #$FF            •• HAS Y-COORDINATE REACHED Ø ?
9412- FØ Ø4      133Ø  BEQ RTN             •• YES-WE'RE FINISHED
9414- C5 FC      134Ø  CMP VT              •• HAVE WE REACHED VT YET ?
9416- BØ CA      135Ø  BCS L1              •• NO-START THE NEXT LINE
9418- 6Ø         136Ø RTN RTS              •• DONE-EXIT ROUTINE
```

LISTING 2: THE DRAW ROUTINE

```
                 Ø1ØØ • DRAW ROUTINE
                 Ø11Ø • COPYRIGHT 1984 BY MICROSPARC INC.
                 Ø12Ø • S-C ASSEMBLER
                 Ø13Ø •
                 1ØØØ  .OR $9394
                 1Ø1Ø  .TF DRAW $9394.OBJ
ØØFC-            1Ø2Ø VT .EQ $FC           •• DECIMAL 252
ØØFD-            1Ø3Ø VB .EQ $FD           •• DECIMAL 253
ØØFE-            1Ø4Ø HR .EQ $FE           •• DECIMAL 254
ØØFF-            1Ø5Ø HL .EQ $FF           •• DECIMAL 255
ØØ26-            1Ø6Ø HBASL .EQ $26        •• DECIMAL 38  (SCREEN BASE
ØØ27-            1Ø7Ø HBASH .EQ $27        •• DECIMAL 39  ADDRESS)
ØØØ6-            1Ø8Ø YO .EQ $6            •• DECIMAL 6
ØØFA-            1Ø9Ø BASL .EQ $FA         •• DECIMAL 25Ø (TABLE BASE
ØØFB-            11ØØ BASH .EQ $FB         •• DECIMAL 252 ADDRESS)
9464-            111Ø YADDR .EQ $9464      •• DECIMAL 37988 (READ YTABLE)
CØ54-            112Ø PAGE1 .EQ $CØ54
CØ55-            113Ø PAGE1X .EQ $CØ55
9394- A9 ØØ      115Ø DRAW LDA #Ø          •• CALL 3778Ø TO ENTER
9396- 85 FA      117Ø  STA BASL            •• POINT TO START OF TABLE
9398- A5 FD      118Ø  LDA VB              •• GET BOTTOM Y-COORDINATE
939A- 85 Ø6      119Ø  STA YO              •• STORE IN $6 FOR USE BY YADDR
939C- 2Ø 64 94   12ØØ L1A JSR YADDR        •• RETURNS-LO=HBASL/HI=HBASH
939F- A4 FE      121Ø  LDY HR              •• SET Y-REG TO RIGHTMOST BYTE
93A1- A2 ØØ      122Ø  LDX #Ø              •• SET TABLE OFFSET=Ø
93A3- A1 FA      123Ø L2A LDA (BASL,X)     •• GET SHAPE BYTE FROM TABLE
93A5- 8D 54 CØ   124Ø  STA PAGE1           •• DRAW MAIN MEMORY
93A8- 51 26      125Ø  EOR (HBASL),Y       •• MODIFY TO BACKGROUND
93AA- 91 26      126Ø  STA (HBASL),Y       •• LOAD SHAPE BYTE ON SCREEN
93AC- E6 FA      127Ø  INC BASL            •• POINT TO NEXT TABLE ELEMENT
93AE- DØ Ø2      128Ø  BNE J1              •• IF x256 BYTES JUMP
93BØ- E6 FB      129Ø  INC BASH            •• PAGE OVERFLOW-GOTO NEXT PAGE
93B2- A1 FA      13ØØ J1 LDA (BASL,X)      •• GET SHAPE BYTE FROM TABLE
93B4- 8D 55 CØ   131Ø  STA PAGE1X          •• DRAW AUXILIARY MEMORY
93B7- 51 26      1315  EOR (HBASL),Y       •• MODIFY TO BACKGROUND
93B9- 91 26      132Ø  STA (HBASL),Y       •• LOAD SHAPE BYTE ON SCREEN
93BB- E6 FA      133Ø  INC BASL            •• POINT TO NEXT TABLE ELEMENT
93BD- DØ Ø2      134Ø  BNE NC2             •• IF x256 BYTES JUMP
93BF- E6 FB      135Ø  INC BASH            •• PAGE OVERFLOW-GOTO NEXT PAGE
93C1- 88         136Ø NC2 DEY              •• POINT TO NEXT SCREEN ADDRESS
93C2- CØ FF      137Ø  CPY #$FF            •• HAS Y-REGISTER REACHED Ø ?
93C4- FØ Ø4      138Ø  BEQ NXTLN2          •• YES-GOTO NEXT LINE
93C6- C4 FF      139Ø  CPY HL              •• IS Y-REGISTER >=HL ?
93C8- BØ D9      14ØØ  BCS L2A             •• YES-JUMP TO LOOP2A
93CA- C6 Ø6      141Ø NXTLN2 DEC YO        •• MOVE UP YO NEXT LINE
93CC- A5 Ø6      142Ø  LDA YO              •• GET NEW Y-COORDINATE
93CE- C9 FF      143Ø  CMP #$FF            •• HAS Y-COORDINATE REACHED Ø ?
93DØ- FØ Ø4      144Ø  BEQ RTN2            •• YES-WE'RE FINISHED
93D2- C5 FC      145Ø  CMP VT              •• HAVE WE REACHED VT YET ?
93D4- BØ C6      1455  BCS L1A             •• NO-START THE NEXT LINE
93D6- 6Ø         147Ø RTN2 RTS             •• DONE-EXIT ROUTINE
```

LISTING 0.5: DHR.DRIVER FROM THE MAY ISSUE

```
9283- A9 51 2Ø 92 92
9288- A9 26 4C 9F 92 A9 EA 2Ø
929Ø- 9F 92 8D 63 93 8D 72 93
9298- 8D AB 93 8D BA 93 6Ø 8D
92AØ- 64 93 8D 73 93 8D AC 93
92A8- 8D BB 93 6Ø A5 FE C9 27
92BØ- BØ Ø4 E6 FE E6 FF 6Ø A5
92B8- FF FØ Ø4 C6 FE C6 FF 6Ø
92CØ- A5 FC FØ Ø4 C6 FC C6 FD
92C8- 6Ø A5 FD C9 BF BØ Ø4 E6
92DØ- FC E6 FD 6Ø A5 FC 38 E5
92D8- E3 3Ø Ø9 85 FC A5 FD 38
92EØ- E5 E3 85 FD 6Ø A5 FD 18
92E8- 65 E3 C9 CØ BØ Ø9 85 FD
92FØ- A5 FC 18 65 E3 85 FC 6Ø
92F8- A9 ØØ 8D Ø1 CØ 85 FA A5
93ØØ- FD 85 Ø6 2Ø 64 94 A4 FF
93Ø8- 8D 55 CØ 2Ø 2B 93 8D 54
931Ø- CØ 2Ø 2B 93 C8 C4 FE 9Ø
9318- EF FØ ED C6 Ø6 A5 Ø6 C9
932Ø- FF FØ Ø4 C5 FC BØ DC 2Ø
9328- DA 93 6Ø A2 ØØ A1 FA C9
933Ø- 7F FØ 1Ø C9 Ø1 9Ø ØC 86
9338- F9 4A 26 F9 E8 EØ Ø7 9Ø
934Ø- F8 A5 F9 91 26 E6 FA DØ
9348- Ø2 E6 FB 6Ø A9 ØØ 8D Ø1
935Ø- CØ 85 FA A5 FC 85 Ø6 2Ø
9358- 64 94 A4 FE A2 ØØ A1 FA
936Ø- 8D 54 CØ 51 26 91 26 E6
9368- FA DØ Ø2 E6 FB A1 FA 8D
937Ø- 55 CØ 51 26 91 26 E6 FA
9378- DØ Ø2 E6 FB 88 CØ FF FØ
938Ø- Ø4 C4 FF BØ D9 E6 Ø6 A5
9388- Ø6 C9 FF FØ Ø6 C5 FD 9Ø
939Ø- C6 FØ C4 6Ø A9 ØØ 8D Ø1
9398- CØ 85 FA A5 FD 85 Ø6 2Ø
93AØ- 64 94 A4 FE A2 ØØ A1 FA
93A8- 8D 54 CØ 51 26 91 26 E6
93BØ- FA DØ Ø2 E6 FB A1 FA 8D
93B8- 55 CØ 51 26 91 26 E6 FA
93CØ- DØ Ø2 E6 FB 88 CØ FF FØ
93C8- Ø4 C4 FF BØ D9 C6 Ø6 A5
93DØ- Ø6 C9 FF FØ Ø4 C5 FC BØ
93D8- C6 6Ø
```

This is the same approach used by SCAN, DRAW, and DRAWDN. To use DRAW, you must first specify VT, VB, HR, and HL to define where on the screen the shape is to be drawn. You must also **POKE 251, SHNUM** to tell the routine where in memory it is to find the Shape Table.

DRAWDN $934C

LISTING 3: THE DRAWDN ROUTINE

```
                 Ø1ØØ • DRAWDN ROUTINE
                 Ø11Ø •
                 Ø12Ø • COPYRIGHT 1984 BY MICROSPARC, INC.
                 Ø13Ø • S-C ASSEMBLER
                 Ø14Ø •
                 1ØØØ  .OR $934C
                 1Ø1Ø  .TF DRAWDN $934C.OBJ
ØØFC-            1Ø2Ø VT .EQ $FC           •• DECIMAL 252
ØØFD-            1Ø3Ø VB .EQ $FD           •• DECIMAL 253
```

The DRAWDN routine (**Listing 3**) works just the same as DRAW except that it places the data bytes on the screen in a slightly different order. This routine is handy for flipping shapes upside-down, or moving shapes behind (or from behind) other shapes or background. We'll demonstrate this routine shortly. To use DRAWDN you will again need to specify SHNUM, VT, VB, HR, and HL.

REVDIR $91F8

The purpose of REVDIR (**Listing 4**) is to physically reverse the appearance of a shape from left to right by placing the Shape Table bytes on the screen in reverse of the order in which they were SCANned.

Before each byte is placed on the screen, the bit pattern of bits 0-6 is reversed, so the routine not only changes the order, but also the values that are stored on the Hi-Res screen. Bit 7 is ignored and automatically set to zero.

Before the bits are reversed, the byte is first checked for the values 0 (00000000) and 127 (01111111). You'll notice that our sample shape has 59 of the 84 bytes with one of these two patterns, so you can save time by not reversing unneeded bytes. This is also a good reason why you should use **HCOLOR=3** when creating your shapes, as this will keep bit 7 set to zero.

Each time this routine changes to a new screen address, it places the first data byte on page 1X (the even column), and then places the next data byte on page 1 (the odd column). After the shape is reversed and placed on the screen, the reversed shape is then reSCANned into the Shape Table so that the table always conforms to the appearance of the shape on the screen. REVDIR is also defined with SHNUM, VT, VB, HR,and HL.

The shape processing methods shown in **Figure 2** illustrate how each of the above routines processes the data bytes within the defined area of the block shape. We will not discuss the internal workings of each routine here, as each routine is heavily documented within each listing.

Notice in **Figure 2** that both SCAN and DRAW enter the shape at VB/HR and work through the bytes, ending at VT/HL; therefore, DRAW will display the shape exactly the same way that it was SCANned.

DRAWDN begins placing shape bytes on the screen at VT/HR, working through to VB/HL. The result is that DRAWDN will draw the shape upside-down. This routine can be used to flip shapes over, or as we'll soon see, both DRAW and DRAWDN can be used to bring shapes from behind other shapes. Which routine you select will depend on whether you're coming from behind another sh^pe at the top or the bottom of the shape.

The REVDIR routine begins processing the shape at VB/HL, and finishes up at VT/HR. The effect here is one of flipping the shape over from left to right.

```
00FE-          1040 HR .EQ $FE          .. DECIMAL 254
00FF-          1050 HL .EQ $FF          .. DECIMAL 255
0026-          1060 HBASL .EQ $26       .. DECIMAL 38  (SCREEN BASE
0027-          1070 HBASH .EQ $27       .. DECIMAL 39  ADDRESS)
0006-          1080 YO .EQ $6           .. DECIMAL 6
00FA-          1090 BASL .EQ $FA        .. DECIMAL 250 (TABLE BASE
00FB-          1100 BASH .EQ $FB        .. DECIMAL 252 ADDRESS)
9464-          1110 YADDR .EQ $9464     .. DECIMAL 37988 (READ YTABLE)
C054-          1120 PAGE1 .EQ $C054
C055-          1130 PAGE1X .EQ $C055
934C- A9 00    1150 DRAWDN LDA #0       .. CALL 37708 TO ENTER
934E- 85 FA    1170  STA BASL           .. POINT TO START OF TABLE
9350- A5 FC    1180  LDA VT             .. GET TOP Y-COORDINATE
9352- 85 06    1190  STA YO             .. STORE IN $6 FOR USE BY YADDR
9354- 20 64 94 1200 L1A JSR YADDR       .. RETURNS-LO=HBASL/HI=HBASH
9357- A4 FE    1210  LDY HR             .. SET Y-REG TO RIGHTMOST BYTE
9359- A2 00    1220  LDX #0             .. SET TABLE OFFSET=0
935B- A1 FA    1230 L2A LDA (BASL,X)    .. GET SHAPE BYTE FROM TABLE
935D- 8D 54 C0 1240  STA PAGE1          .. DRAW MAIN MEMORY
9360- 51 26    1250  EOR (HBASL),Y      .. MODIFY TO BACKGROUND
9362- 91 26    1260  STA (HBASL),Y      .. LOAD SHAPE BYTE ON SCREEN
9364- E6 FA    1270  INC BASL           .. POINT TO NEXT TABLE ELEMENT
9366- D0 02    1280  BNE J1             .. IF ×256 BYTES JUMP
9368- E6 FB    1290  INC BASH           .. PAGE OVERFLOW-GOTO NEXT PAGE
936A- A1 FA    1300 J1 LDA (BASL,X)     .. GET SHAPE BYTE FROM TABLE
936C- 8D 55 C0 1310  STA PAGE1X         .. DRAW AUXILIARY MEMORY
936F- 51 26    1315  EOR (HBASL),Y      .. MODIFY TO BACKGROUND
9371- 91 26    1320  STA (HBASL),Y      .. LOAD SHAPE BYTE ON SCREEN
9373- E6 FA    1330  INC BASL           .. POINT TO NEXT TABLE ELEMENT
9375- D0 02    1340  BNE NC2            .. IF ×256 BYTES JUMP
9377- E6 FB    1350  INC BASH           .. PAGE OVERFLOW-GOTO NEXT PAGE
9379- 88       1360 NC2 DEY             .. POINT TO NEXT SCREEN ADDRESS
937A- C0 FF    1370  CPY #$FF           .. HAS Y-REGISTER REACHED 0 ?
937C- F0 04    1380  BEQ NXTLN2         .. YES-GOTO NEXT LINE
937E- C4 FF    1390  CPY HL             .. IS Y-REGISTER >=HL ?
9380- B0 D9    1400  BCS L2A            .. YES-JUMP TO LOOP2A
9382- E6 06    1410 NXTLN2 INC YO       .. MOVE DOWN TO NEXT LINE
9384- A5 06    1420  LDA YO             .. GET NEW Y-COORDINATE
9386- C9 FF    1430  CMP #$FF           .. HAS Y-COORDINATE REACHED 0 ?
9388- F0 06    1440  BEQ RTN2           .. YES-WE'RE FINISHED
938A- C5 FD    1450  CMP VB             .. HAVE WE REACHED VB YET ?
938C- 90 C6    1455  BCC L1A            .. NO-START THE NEXT LINE
938E- F0 C4    1460  BEQ L1A            .. NO-THIS IS LAST LINE
9390- 60       1470 RTN2 RTS            .. DONE-EXIT ROUTINE
```

LISTING 4: REVDIR

```
               0100 . REVDIR ROUTINE
               0110 .
               0120 . COPYRIGHT 1984 BY MICROSPARC, INC.
               0130 . S-C ASSEMBLER
               0140 .
               1000  .OR $92F8
               1010  .TF REVDIR $92F8.OBJ
00FC-          1020 VT .EQ $FC     .    .. DECIMAL 252
00FD-          1030 VB .EQ $FD          .. DECIMAL 253
00FE-          1040 HR .EQ $FE          .. DECIMAL 254
00FF-          1050 HL .EQ $FF          .. DECIMAL 255
0026-          1060 HBASL .EQ $26       .. DECIMAL 38  (SCREEN BASE
0027-          1070 HBASH .EQ $27       .. DECIMAL 39  ADDRESS)
0006-          1080 YO .EQ $6           .. DECIMAL 6
00F9-          1085 NUBYTE .EQ $F9      .. DECIMAL 249
00FA-          1090 BASL .EQ $FA        .. DECIMAL 250 (TABLE BASE
00FB-          1100 BASH .EQ $FB        .. DECIMAL 252 ADDRESS)
9464-          1110 YADDR .EQ $9464     .. DECIMAL 37988 (READ YTABLE)
C054-          1120 PAGE1 .EQ $C054
C055-          1130 PAGE1X .EQ $C055
93DA-          1145 SCAN .EQ $93DA
92F8- A9 00    1150 REVDIR LDA #0       .. CALL 37624 TO ENTER
92FA- 85 FA    1160  STA BASL           .. POINT TO START OF TABLE
92FC- A5 FD    1170  LDA VB             .. GET BOTTOM Y-COORDINATE
92FE- 85 06    1180  STA YO             .. STORE IN $6 FOR USE BY YADDR
9300- 20 64 94 1190 L1A JSR YADDR       .. RETURNS LO=HBASL/HI=HBASH
9303- A4 FF    1200  LDY HL             .. SET Y-REG TO LEFTMOST BYTE
9305- 8D 55 C0 1212 L2A STA PAGE1X      .. DRAW AUXILIARY MEMORY
9308- 20 28 93 1214  JSR R              .. ROTATE/DRAW DATA BYTE
930B- 8D 54 C0 1216  STA PAGE1          .. DRAW MAIN MEMORY
930E- 20 28 93 1218  JSR R              .. ROTATE/DRAW DATA BYTE
9311- C8       1220  INY                .. POINT TO NEXT AFFRESS -->
9312- C4 FE    1370 NC2 CPY HR          .. HAVE WE PASSED HR YET?
9314- 90 EF    1380  BCC L2A            .. NO-GET THE NEXT ADDRESS
9316- F0 ED    1390  BEQ L2A            .. NO-WE'RE DOING HR NOW
9318- C6 06    1410  DEC YO             .. MOVE UP TO NEXT LINE
931A- A5 06    1420  LDA YO             .. GET NEW Y-COORDINATE
931C- C9 FF    1430  CMP #$FF           .. HAS Y-COORDINATE REACHED 0?
931E- F0 04    1440  BEQ RTN2           .. YES-WE'RE FINISHED
9320- C5 FC    1450  CMP VT             .. HAVE WE PASSED VT?
9322- B0 DC    1460  BCS L1A            .. NO-START THE NEXT LINE
9324- 20 DA 93 1470 RTN2 JSR SCAN       .. DONE-REVISE BLOCK TABLE
9327- 60       1480  RTS                .. EXIT ROUTINE
9328- A2 00    1490 R LDX #0            .. SET OFFSET POINTER=0
932A- A1 FA    1500  LDA (BASL,X)       .. GET SHAPE BYTE FROM TABLE
932C- C9 7F    1510  CMP #127           .. IS BYTE 01111111 ? ($7F)
```

MOVE Routines $9283

This collection of routines (**Listing 5**) will be very handy for use in our animation. Here's what each routine does.

EORON $9283 — If you look at **lines 1250 and 1315** of the DRAW and DRAWDN routines, you will find the instructions EOR (HBASL),Y. These instructions modify the shape data byte to the present screen background before drawing to the screen. This is very useful for making the DRAW and DRAWDN routines erase shapes from the screen. The EOR function is also useful in moving shapes over the background or over other shapes, and restoring the bit patterns on the screen as the shape moves away. The EORON routine places the EOR (HBASL),Y instructions in **lines 1250 and 1315** of both DRAW and DRAWDN just as they appear in **Listings 2** and **3**.

EOROFF $928D — This routine removes the EOR (HBASL),Y instructions from DRAW and DRAWDN, replacing them with NOP (No OPeration) instructions. In much of your animation you will not want the EOR instructions functioning.

MOVERT $92AC — This routine is used with rightward moving shapes to INCrement the values of HR and HL.

MOVELF $92B7 — This routine simply DECrements the values of HR and HL and is used on leftward moving shapes.

GOUP $92C0 — The GOUP routine DECrements the values of VT and VB for upward moving shapes.

GODOWN $92C9 — This routine will INCrement the values of VT and VB for downward moving shapes.

YINCRU $92D4 — To use this routine you must first POKE into location 227 ($E3) the number of vertical dots which you want the shape to move. The routine will then subtract that value (YINCR) from both VT and VB, causing the shape to move upward YINCR screen coordinates.

YINCRD $92E5 — This routine is similar to YINCRU, except that it adds the value of YINCR to both VT and VB for downward moving shapes.

This collection of routines will make it very easy for you to manipulate the values of VT, VB, HR, and HL for moving shapes about the screen. Note that each of the move routines includes protectors which will not allow the values of VT, VB, HR, or HL to exceed the legal limits of 0-39.

Once you've added all of these new routines to your driver, save them to disk with the command:

BSAVE DHR.DRIVER,A$9283,L$37D

Creating a Double Hi-Res Shape

Now that we've got the boring stuff out of the way, let's put your Apple to work. **Listing 6** is a short program that will create spaceship shapes and automatically save them to disk.

At this point you should enter the program and RUN it; then we'll discuss what it does.

```
932E- FØ 1Ø     152Ø  BEQ J2              •• YES-NO NEED TO REVERSE
933Ø- C9 Ø1     153Ø  CMP #1              •• IS BYTE ØØØØØØØØ ? ($ØØ)
9332- 9Ø ØC     154Ø  BCC J2              •• YES-NO NEED TO REVERSE
9334- 86 F9     155Ø  STX NUBYTE          •• SET ALL BITS TO ZERO
9336- 4A        156Ø NXTBIT LSR           •• PUSH BIT OFF SHAPE BYTE -->
9337- 26 F9     157Ø  ROL NUBYTE          •• PUT BIT IN REVERSED BYTE x--
9339- E8        158Ø  INX                 •• BUMP BIT COUNTER
933A- EØ Ø7     159Ø  CPX #7              •• HAVE WE DONE BITS Ø-6?
933C- 9Ø F8     16ØØ  BCC NXTBIT          •• NO-GO DO NEXT BIT
933E- A5 F9     16Ø5  LDA NUBYTE          •• LOAD REVERSED BYTE
934Ø- 91 26     161Ø J2 STA (HBASL),Y     •• LOAD REVERSED BYTE ON SCREEN
9342- E6 FA     162Ø  INC BASL            •• POINT TO NEXT TABLE ELEMENT
9344- DØ Ø2     163Ø  BNE J3              •• IF x256 BYTES-JUMP
9346- E6 FB     164Ø  INC BASH            •• PAGE OVERFLOW-GOTO NEXT PAGE
9348- 6Ø        165Ø J3 RTS               •• FINISHED BYTE ROTATION
```

LISTING 5: MOVE ROUTINES

```
                Ø1ØØ • MOVE ROUTINES
                Ø11Ø •
                Ø12Ø • COPYRIGHT 1984 BY MICROSPARC, INC.
                Ø13Ø • S-C ASSEMBLER
                Ø14Ø •
                1ØØØ  .OR $9283
                1Ø1Ø  .TF MOVE ROUTINES $9283.OBJ
ØØFC-           1Ø3Ø VT .EQ $FC           •• DECIMAL 252
ØØFD-           1Ø4Ø VB .EQ $FD           •• DECIMAL 253
ØØFE-           1Ø5Ø HR .EQ $FE           •• DECIMAL 254
ØØFF-           1Ø6Ø HL .EQ $FF           •• DECIMAL 255
ØØE3-           1Ø7Ø YINCR .EQ $E3        •• DECIMAL 227
9283- A9 51     12ØØ EORON LDA #$51       •• CALL 375Ø7 TO ENTER
9285- 2Ø 92 92  121Ø  JSR STORE1          •• INSERT EOR (HBASL),Y
9288- A9 26     122Ø  LDA #$26            •• IN DRAW AND DRAWDN
928A- 4C 9F 92  123Ø  JMP STORE2
928D- A9 EA     125Ø EOROFF LDA #$EA      •• CALL 37517 TO ENTER
928F- 2Ø 9F 92  126Ø  JSR STORE2          •• REMOVE EOR (HBASL),Y FROM DRAW/DRAWDN
9292- 8D 6Ø 93  128Ø STORE1 STA $936Ø
9295- 8D 6F 93  129Ø  STA $936F
9298- 8D A8 93  13ØØ  STA $93A8
929B- 8D B7 93  131Ø  STA $93B7
929E- 6Ø        132Ø  RTS
929F- 8D 61 93  133Ø STORE2 STA $9361
92A2- 8D 7Ø 93  134Ø  STA $937Ø
92A5- 8D A9 93  135Ø  STA $93A9
92A8- 8D B8 93  136Ø  STA $93B8
92AB- 6Ø        137Ø  RTS
92AC- A5 FE     138Ø MOVERT LDA HR         •• CALL 37548 TO ENTER
92AE- C9 27     139Ø  CMP #39
92BØ- BØ Ø4     14ØØ  BCS J1               •• INCREMENT HR AND HL
92B2- E6 FE     141Ø  INC HR               •• DON'T ALLOW HR>39
92B4- E6 FF     142Ø  INC HL
92B6- 6Ø        143Ø J1 RTS
92B7- A5 FF     144Ø MOVELF LDA HL         •• CALL 37559 TO ENTER
92B9- FØ Ø4     145Ø  BEQ J2
92BB- C6 FE     146Ø  DEC HR               •• DECREMENT HR AND HL
92BD- C6 FF     147Ø  DEC HL               •• DON'T ALLOW HLxØ
92BF- 6Ø        148Ø J2 RTS
92CØ- A5 FC     149Ø GOUP LDA VT           •• CALL 37568 TO ENTER
92C2- FØ Ø4     15ØØ  BEQ J3
92C4- C6 FC     151Ø  DEC VT               •• DECREMENT VT AND VB
92C6- C6 FD     152Ø  DEC VB               •• DON'T ALLOW VTxØ
92C8- 6Ø        153Ø J3 RTS
92C9- A5 FD     154Ø GODOWN LDA VB         •• CALL 37577 TO ENTER
92CB- C9 BF     155Ø  CMP #191
92CD- BØ Ø4     156Ø  BCS J4               •• INCREMENT VT AND VB
92CF- E6 FC     157Ø  INC VT               •• DON'T ALLOW VB>191
92D1- E6 FD     158Ø  INC VB
92D3- 6Ø        159Ø J4 RTS
92D4- A5 FC     16ØØ YINCRU LDA VT         •• CALL 37588 TO ENTER
92D6- 38        161Ø  SEC
92D7- E5 E3     162Ø  SBC YINCR
92D9- 3Ø Ø9     163Ø  BMI J5               •• SUBTRACT YINCR
92DB- 85 FC     164Ø  STA VT               •• FROM VT AND VB
92DD- A5 FD     165Ø  LDA VB               •• DON'T ALLOW VTxØ
92DF- 38        166Ø  SEC
92EØ- E5 E3     167Ø  SBC YINCR
92E2- 85 FD     168Ø  STA VB
92E4- 6Ø        169Ø J5 RTS
92E5- A5 FD     17ØØ YINCRD LDA VB         •• CALL 376Ø5 TO ENTER
92E7- 18        171Ø  CLC
92E8- 65 E3     172Ø  ADC YINCR
92EA- C9 CØ     173Ø  CMP #192             •• ADD YINCR
92EC- BØ Ø9     174Ø  BCS J6               •• TO VT AND VB
92EE- 85 FD     175Ø  STA VB               •• DON'T ALLOW VB>191
92FØ- A5 FC     176Ø  LDA VT
92F2- 18        177Ø  CLC
92F3- 65 E3     178Ø  ADC YINCR
92F5- 85 FC     179Ø  STA VT
92F7- 6Ø        18ØØ J6 RTS
```

How SHAPE.MAKER Works

Lines 80-140 should be rather easily understood, as we worked with the same instructions last month.

Lines 150-190 drudgingly go about the process of drawing our spaceship on the screen using a series of HPLOT end points that are defined in the DATA statements. The shape is drawn exactly as it is defined in **Figure 1**. The extra line of empty bytes above and below the shape are there so that the shape will erase itself as we move it about the screen.

Line 200 POKEs the value of SHNUM. (We're going to store the shape at $9000.) Then it sets the values of VT, VB, HR, and HL. Finally, it SCANs the shape into memory. At this point our Block Shape Table has been created in memory and is available for us to use with our drawing routines.

Line 210 saves the Shape Table to disk.

Line 220 changes the values of HR and HL to another part of the screen and test DRAWs the shape from the table. If you don't have two spaceships on the screen now, there is a problem with either your SCAN or your DRAW routine.

Line 230 changes HR and HL again to yet another part of the screen and DRAWDNs the shape from the table. The third spaceship which appears on the screen should be drawn upside-down.

Line 240 changes the shape number (SHNUM) to 143 ($8F00) and SCANs the upside-down shape into another Shape Table. You should note here that since DRAWDN always draws the shape upside-down from the way it was SCANned, now that we SCANned shape #143 upside-down, DRAWDN will now draw shape #143 in its proper upright position.

Line 250 saves this second Shape Table to disk.

At this point there are two shapes saved to disk. Shape #144 will be drawn in its proper upright position with DRAW, and shape #143 will be drawn in its proper upright position with DRAWDN.

Line 260 reselects shape #144, moves HR and HL again, and draws a reversed version of shape #144. You probably won't notice any difference in this REVDIRed shape since it's symmetrical; however, if the shape looks correct on the screen, you can be reasonably sure that REVDIR is working properly.

You should be aware that Shape Table #144 has been modified (in memory only, not on disk) by the REVDIR routine which reSCANned the shape. If you look at **Figure 1**, you'll notice that there are two empty dots to the right of our shape; when REVDIR did its thing, it moved those two empty dots to the left of the shape.

As you can see from this short little program, the hardest part was drawing the original shape on the screen using HPLOTs. Once it was on the screen, the SCAN routine made it quite easy to translate what we'd drawn into a Block Shape Table.

A Moving Conclusion

Next month we'll show you how to animate the shapes you've created. You'll learn how to produce Double Hi-Res movement from both Applesoft and machine language. Your //e or //c will love it.

TABLE 1: SUMMARY OF ADDITIONAL DHR.DRIVER ROUTINES

Routine Name	Call Address	Hex Address	Routine Function
SCAN	37850	$93DA	Create a Block Shape Table from the screen.
DRAW	37780	$9394	Draw a shape from the bottom to the top.
DRAWDN	37708	$934C	Draw a shape from the top to the bottom.
REVDIR	37624	$92F8	Reverse the shape and create a new table.
YINCRD	37605	$92E5	Add YINCR to VT and VB.
YINCRU	37588	$92D4	Subtract YINCR from VT and VB.
GODOWN	37577	$92C9	Add one to VT and VB.
GOUP	37568	$92C0	Subtract one from VT and VB.
MOVELF	37559	$92B7	Subtract one from HR and HL.
MOVERT	37548	$92AC	Add one to HR and HL.
EOROFF	37517	$928D	Cancel DRAW and DRAWDN EOR functions.
EORON	37507	$9283	Install DRAW and DRAWDN EOR functions.

Special POKEs to use with the driver:

POKE 227,YINCR	Establishes the value to be used by YINCRU and YINCRD for modifying VT and VB.
POKE 251,SHNUM	Tell SCAN, DRAW, DRAWDN, and REVDIR where to find the Shape Table.
POKE 252,VT	Set the topmost Y-coordinate of the shape.
POKE 253,VB	Set the bottommost Y-coordinate of the shape.
POKE 254,HR	Set the rightmost address offset of the shape.
POKE 255,HL	Set the leftmost address offset of the shape.

Note that there are many other points at which you might choose to enter a driver routine to perform special fuctions. If you need to take some action that is not described in the documentation, look through each listing to see if some other entry point might do the job. There are also many ways that you could change the functions of a routine with a few simple POKEs. For instance, the GODOWN and YINCRD routines could be changed to keep you above VB = 159 if you were using the mixed text and graphics mode, or you could enter a few POKEs to cancel the automatic SCAN function of REVDIR.

LISTING 6: SHAPE.MAKER

```
10  REM   ***********************
20  REM   *      SHAPE.MAKER     *
30  REM   * BY ROBERT R. DEVINE *
40  REM   * COPYRIGHT (C)  1984 *
50  REM   * BY MICROSPARC, INC. *
60  REM   * LINCOLN, MA.  Ø1773 *
70  REM   ***********************
80  PRINT  CHR$ (4)"BLOAD DHR.DRIVER": CALL 3
     7999: HIMEM: 37507: REM  LOAD/SETUP/PROT
     ECT
90  CALL 37953: REM  INIT
100  HGR : CALL 37928: REM  CLEAR DHR SCREEN
110  POKE 49153,Ø: POKE 49234,Ø: REM  8ØSTORE
     /FULL SCREEN
120  HCOLOR= 3: GOTO 15Ø
130  POKE 49236,Ø:C =  INT (X / 7): IF C / 2 =
      INT (C / 2) THEN  POKE 49237,Ø: REM  FL
     IP PAGE2
140 XC =  INT (C / 2) + X / 7 - C:XC =  INT (
     XC * 7 + .5): RETURN
150  FOR X = Ø TO 19: READ Y: READ Y1: GOSUB
     13Ø: HPLOT XC,Y TO XC,Y1: NEXT : RESTORE
160  FOR X = 39 TO 2Ø STEP  - 1: READ Y: READ
     Y1: GOSUB 13Ø: HPLOT XC,Y TO XC,Y1: NEXT
170  FOR M = 6 TO 3Ø STEP 8: FOR X = M TO M +
     3: READ Y: GOSUB 13Ø: HPLOT XC,Y TO XC,5
     : NEXT X,M
180  DATA   5,6,5,6,5,7,5,7,4,8,4,8,7,9,7,9,7
     ,1Ø,7,1Ø,3,1Ø,3,1Ø,2,11,2,11,7,11,7,11,7
     ,11,7,11,1,12,1,12
190  DATA  4,4,3,3,2,2,1,1,1,1,2,2,3,3,4,4
200  POKE 251,144: POKE 252,Ø: POKE 253,13: POKE
     254,2: POKE 255,Ø: CALL 3785Ø: REM  SCAN
      THE SHIP
210  PRINT  CHR$ (4)"BSAVE SHAPE-U #144,A$9ØØ
     Ø,L84": REM  SAVE 'DRAW' SHAPE
220  POKE 254,12: POKE 255,1Ø: CALL 3778Ø: REM
      DRAW IT
230  POKE 254,22: POKE 255,2Ø: CALL 377Ø8: REM
      DRAWDN IT
240  POKE 251,143: CALL 3785Ø: REM  SCAN DRAW
     DN SHAPE
250  PRINT  CHR$ (4)"BSAVE SHAPE-D #143,A$8FØ
     Ø,L84": REM  SAVE 'DRAWDN' SHAPE
260  POKE 251,144: POKE 254,32: POKE 255,3Ø: CALL
     37624: REM  REVDIR IT
```