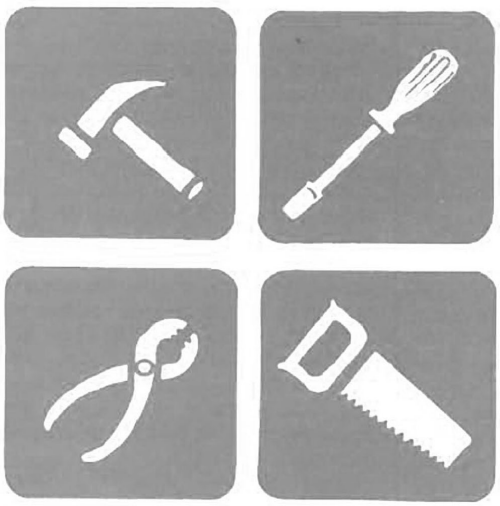
DOS 3.3



DOUBLE HI-RES GRAPHICS III

The Graphics Workshop continues its exploration of block shape animation in Part III of the Double Hi-Res series. Use the routines developed last month on your 128K //e or //c for simple block shape animation.

by Robert R. Devine
1415 West 19th St.
El Dorado, AR 71730

Last month we described processes and programs for drawing block shapes on the Double Hi-Res screen. This month we'll learn the fine art of DHR animation. You'll need the programs we developed last month, and it'll help to refer back to the article and figures.

Animating the Shape

Now that we've got some shapes to work with, let's try our first animation. We're going to demonstrate not only how we can use the DRAW and DRAWDN routines for animation (without the need to erase), but we'll also see how we can make our shapes **appear** to move behind other objects on the screen.

First we'll draw a horizontal orange barrier through the middle of the screen. Then we'll move the spaceship up and down the screen with the shape always moving behind the barrier. Before we actually run the program, let's look at **Figure 1**, which shows how we go about disappearing and reappearing from behind the barrier.

The first thing you need to understand is that **you don't need to draw the entire shape**. There is nothing included in your Shape Table which defines where it ends. You can never change the HR-HL dimensions of your shape without destroying the appearance of the shape; however, you can change the VT-VB dimensions of the shape. Once a driver routine has completed its work within the VT-VB dimensions you have specified, it will end. If you set VT-VB dimensions less than the actual size of the shape, you will only draw part of the shape.

This is where the difference in approach between DRAW and DRAWDN comes into play. If you only want to draw the top part of your shape, you can use DRAWDN, and the only bytes that will appear will be those between VT and whatever VB you have specified. On the other hand, if you only want the bottom parts, you can use DRAW, and the only parts of the shape that will be drawn are those between VB and whatever VT you have specified.

You could also do the same thing with REVDIR and reverse only the lower parts of your shape; however, you should be aware that since REVDIR automatically reSCANs the shape, your table will, as always, match exactly what's on the screen...but maybe that's just what you want.

If you look at **Figure 1** you will note that our barrier is set from Y-coordinates 90 through 105. Once the shape gets down to VB=89, we will leave it alone and keep INCrementing VT until it is also 89, at which time our shape will appear to have moved behind the barrier.

What has really happened is that we've kept moving it downward, erasing it line by line at the barrier until it is no longer on the screen. To bring the shape out from behind the barrier, we will begin with BOTH VT and VB set at 106. We will then leave VT alone while we keep INCrementing VB and reDRAWing until the entire shape has been brought, line by line, back onto the screen. We will use, on the top side of the barrier, DRAWDN (because we only want the top parts of our shape #143), and we'll use DRAW and shape #144 below the barrier (because we only want the bottom parts of our shape).

Our demonstration program is contained in **Listing 1**. Let's see how the program works.

How ANIMATION.1 Works

Lines 80-150 should be familiar and easily understandable. The only thing new here is that we've added **CALL 37517** to **line 130**, which turns off the EOR functions of DRAW and DRAWDN. If we left the EOR function on, our shape would leave a trail behind it because it would be EORing with its own data bytes.

Line 160 draws the orange barrier across the screen using the **line 140** subroutine to translate our 0-559 coordinates into 0-279 HPLOT coordinates.

Line 170 POKEs the starting shape parameters into memory.

Line 190 moves the shape downward until it reaches the edge of the barrier. First it DRAWDNs the shape, then it uses GODOWN to INCrement VT and VB in readiness for the next draw.

Line 200 — At this point VB=89, so we continue to INCrement VT, then DRAWDN until VT also equals 89, at which time the shape has disappeared from the screen.

Line 210 switches to the proper shape for use with DRAW.

Line 220 sets VT and VB equal to 106, then INCrements VT and DRAWs until we have brought the shape fully back onto the screen, just below the barrier.

Line 230 then moves the shape to the bottom of the screen by DRAWing it and then using GODOWN to INCrement VT and VB for the next drawing operation.

Line 250 is exactly the opposite of **line 230**. Here we DRAW the shape, then use GOUP to DECrement VT and VB until we again reach the barrier.

Line 260 — At this point VT=106, so we

leave VT alone and continue to DECrement VB and redraw the shape until it disappears behind the barrier.

Line 270 changes back to shape #143 for use above the barrier with DRAWDN.

Line 280 — Next we set both VT and VB to 89 and keep DECrementing VT until the shape has been brought from behind the barrier.

Line 290 — Finally we use DRAWDN and GOUP until the shape again reaches the top of the screen.

Line 300 — At this point we haven't developed any smooth moving horizontal animation routines, so this line first enables the EOR function with EORON, then erases the shape, and again disables the EOR with EOROFF.

Line 310 uses the MOVERT routine to INCrement HR and HL in readiness for the next trip down.

Notice that we have not added trap to keep us from going off the right edge of the screen, as this protection is built into the MOVERT routine. Once the shape gets to the right edge, it will simply keep moving up and down in the same place.

This little program, while rather simple and not terribly sophisticated, should give you a good idea of how to work with DRAW and DRAWDN.

Speeding Up the Animation

One way that you could make the animation a bit faster would be to use the YINCRU and YINCRD routines instead of GOUP and GODOWN, which would allow you to move your shapes more than one Y-coordinate per move. If you were to do this, and avoid the need for any erase activities, you would need to allow a number of blank lines above and below your shape, equal to the YINCR you selected. While our animation is quite smooth and reasonably fast, the best way to speed things up would be to bypass the Applesoft interpreter and use straight machine language.

To get an idea of just how much machine code would speed things up, enter the hex code in **Listing 2**, which is an exact machine code translation of **lines 170-320**. Save it to disk with the command:

```
BSAVE FAST.ANIMATION,A$4000,L$B7
```

To try out the machine code version, delete **lines 170-320** from **Listing 1** and add this new **Line 170:**

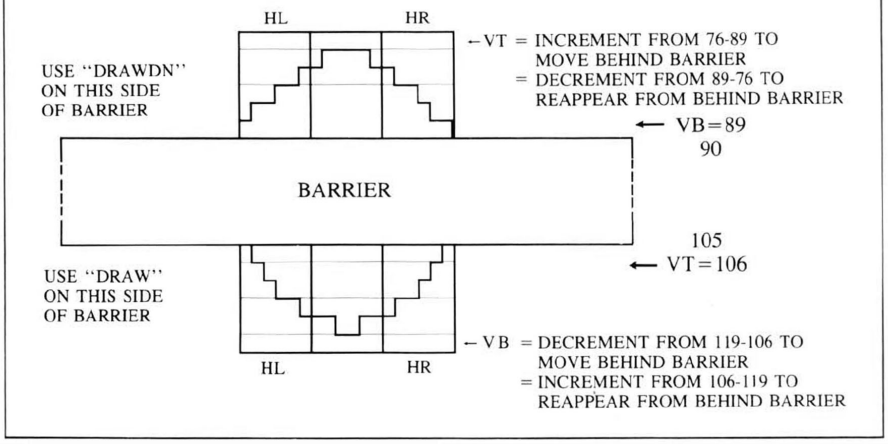
```
170 PRINT CHR$(4)"BLOAD
    FAST.ANIMATION": CALL 16384
```

You'll quickly find that our animation is now too fast, but that's okay. If you were to build a program around it, with all its other various activities, the speed would probably be just about right.

Before we call it quits for this month, let's take a look at one more animation demonstration that performs some horizontal animation, and makes use of the REVDIR routine. At this point we still don't have any smooth-moving horizontal shifting routines. (We'll get into that next month.) However, by moving 14 horizontal dots per move, we can get some reasonable animation.

In our test we'll draw an arrow with HPLOTs, then SCAN it into a Shape Table, and finally move it back and forth across the screen. At each side of the screen we'll REVDIR the shape to point it in the proper direction, then move it down 10 lines after each round trip across the screen. This test is included in **Listing 3**.

FIGURE 1: OVERLAPPING SHAPES



Lines 80-170 initialize Double Hi-Res and draw our arrow shape on the screen. Since we don't yet have any nice self-erasing shift routines, the shape includes 14 blank dots (one extra address) behind it to take care of all the erasing chores. This means that the actual width of the shape is three addresses; however, we've made it four addresses wide to allow for erasing.

Line 180 sets all the shape parameters and SCANs the shape into a table.

Line 190 sets YINCR so that we can move the shape down 10 lines after each trip across the screen.

Line 200 turns off the EOR functions of DRAW.

Line 210 moves the shape rightward across the screen. After each DRAW, the MOVERT routine is used to INCrement HR and HL.

Line 220 reverses the shape to point leftward. You'll note that since REVDIR is a DRAWing routine, it took care of its own erasing of the old arrow as it drew the new one.

Line 230 moves the shape leftward across the screen. Here we've used the MOVELF routine to DECrement HR and HL.

Line 240 — Now that we're back where we began, we erase the shape from the screen.

Line 250 is used to test VB to see if we're at the bottom of the screen. If we are, then VT and VB are reset for 0 and 13. Next, the shape is redrawn and reversed (in one operation) at the top of the screen using REVDIR.

Line 260 — If we're not at the bottom of the screen, then the shape is moved down 10 lines by using YINCRD to add 10 to both VT and VB, and the shape is reversed and redrawn on the screen.

The handy thing about REVDIR is that in some cases it can save you the effort of needing right- and left-facing Shape Tables. In this test we managed to display two different arrows on the screen, even though we only created one Shape Table.

Conclusion

Now that you have an assortment of different Double Hi-Res drawing routines, you should be able to tackle some animation of your own — enjoying some of the advantages of machine code speed, and Applesoft flexibility, without the need to be concerned with all the complexities of the Double Hi-Res screen. For a map of the entire DHR driver, see **Figure 2**. See you next month!

FIGURE 2: MAP OF THE DHR DRIVER

SCREEN ADDRESS TABLE	$9480	
SETUP	$946F	37999
YADDR	$9464	37988
KILL	$944E	37966
INIT	$9441	37958
HGR	$9428	37928
HOME	$941C	37916
SCAN	$93DA	37850
DRAW	$9394	37780
DRAWDN	$934C	37708
REVDIR	$92F8	37624
YINCRD	$92E5	37605
YINCRU	$92D4	37588
GODOWN	$92C9	37577
GOUP	$92C0	37568
MOVELF	$92B7	37559
MOVERT	$92AC	37548
EOROFF	$928D	37517
EORON	$9283	37507

Total length = 893 bytes

LISTING 1: ANIMATION.1

```
10  REM   ***********************
20  REM   *     ANIMATION.1     *
30  REM   * BY ROBERT R. DEVINE *
40  REM   * COPYRIGHT (C)  1984 *
50  REM   * BY MICROSPARC, INC. *
60  REM   * LINCOLN, MA.  01773 *
70  REM   ***********************
80  PRINT  CHR$ (4)"BLOAD DHR.DRIVER": CALL 3
    7999: HIMEM: 37507: REM  LOAD/SETUP/PROT
    ECT
90  PRINT  CHR$ (4)"BLOAD SHAPE-D #143": PRINT
     CHR$ (4)"BLOAD SHAPE-U #144"
100  CALL 37953: REM  INIT
110  HGR : CALL 37928: REM  CLEAR DHR SCREEN
120  POKE 49153,0: POKE 49234,0: REM  80STORE
    /FULL SCREEN
130  HCOLOR= 3: CALL 37517: GOTO 160: REM  HP
    LOT MODE ON/EOR FUNCTION OFF
140  POKE 49236,0:C =  INT (X / 7): IF C / 2 =
     INT (C / 2) THEN  POKE 49237,0: REM  FL
    IP PAGE2
150 XC =  INT (C / 2) + X / 7 - C:XC =  INT (
    XC * 7 + .5): RETURN
160  FOR X1 = 3 TO 559 STEP 4:X = X1 - 1: GOSUB
    140: HPLOT XC,90 TO XC,105:X = X1: GOSUB
    140: HPLOT XC,90 TO XC,105: NEXT : REM
    DRAW HORIZONTAL ORANGE BARRIER
170  POKE 251,143: POKE 252,0: POKE 253,13: POKE
    254,2: POKE 255,0: REM  STARTING SHNUM/V
    T/VB/HR/HL
180  REM  *** GOING DOWN ***
190  FOR VB = 13 TO 88: CALL 37708: CALL 3757
    7: NEXT VB: REM  DRAWDN/GODOWN TO BARRIE
    R
200  FOR VT = 76 TO 89: POKE 252,VT: CALL 377
    08: NEXT VT: REM  MOVE SHIP BEHIND BARRI
    ER
210  POKE 251,144: REM  CHANGE TO NORMAL 'DRA
    W' SHAPE
220  POKE 252,106: FOR VB = 106 TO 119: POKE
    253,VB: CALL 37780: NEXT VB: REM  BRING
    SHIP FROM BEHIND BARRIER
230  FOR VB = 119 TO 191: CALL 37780: CALL 37
    577: NEXT VB: REM  DRAW/GODOWN TO BOTTOM
     OF SCREEN
240  REM  *** GOING UP ***
250  FOR VT = 178 TO 107 STEP  - 1: CALL 3778
    0: CALL 37568: NEXT VT: REM  DRAW/GOUP T
    O BARRIER
260  FOR VB = 119 TO 106 STEP  - 1: POKE 253,
    VB: CALL 37780: NEXT VB: REM  MOVE SHIP
    BEHIND BARRIER
270  POKE 251,143: REM  CHANGE TO 'DRAWDN' SH
    APE
280  POKE 253,89: FOR VT = 89 TO 76 STEP  - 1
    : POKE 252,VT: CALL 37708: NEXT VT: REM
     BRING SHIP FROM BEHIND BARRIER
290  FOR VT = 88 TO 0 STEP  - 1: CALL 37708: CALL
    37568: NEXT VT: REM  DRAWDN/GOUP TO TOP
    OF SCREEN
300  CALL 37507: CALL 37708: CALL 37517: REM
     EORON/ERASE/EOROFF
310  CALL 37548: REM  MOVERIGHT
320  GOTO 190: REM  START DOWN AGAIN
```

LISTING 3: ANIMATION.2

```
10  REM   ***********************
20  REM   *     ANIMATION.2     *
30  REM   * BY ROBERT R. DEVINE *
40  REM   * COPYRIGHT (C)  1984 *
50  REM   * BY MICROSPARC, INC. *
60  REM   * LINCOLN, MA.  01773 *
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 49153,0: POKE 49234,0: REM  80STORE
    /FULL SCREEN
120  HCOLOR= 3: GOTO 150
130  POKE 49236,0:C =  INT (X / 7): IF C / 2 =
     INT (C / 2) THEN  POKE 49237,0: REM  FL
    IP PAGE2
140 XC =  INT (C / 2) + X / 7 - C:XC =  INT (
    XC * 7 + .5): RETURN
150  FOR X = 14 TO 37: GOSUB 130: HPLOT XC,6:
     NEXT X
```

LISTING 2: FAST.ANIMATION

```
4000- A9 8F 85 FB A9 00 85 FC
4008- 85 FF A9 0D 85 FD A9 02
4010- 85 FE A9 0D 85 E3 20 4C
4018- 93 20 C9 92 E6 E3 A5 E3
4020- C9 59 90 F2 A9 4C 85 E3
4028- 85 FC 20 4C 93 E6 E3 A5
4030- E3 C9 5A 90 F3 A9 90 85
4038- FB A9 6A 85 FC 85 E3 85
4040- FD 20 94 93 E6 E3 A5 E3
4048- C9 78 90 F3 A9 77 85 E3
4050- 20 94 93 20 C9 92 E6 E3
4058- A5 E3 C9 C0 90 F2 A9 B2
4060- 85 E3 20 94 93 20 C0 92
4068- C6 E3 A5 E3 C9 6B B0 F2
4070- A9 77 85 E3 85 FD 20 94
4078- 93 C6 E3 A5 E3 C9 6A B0
4080- F3 A9 8F 85 FB A9 59 85
4088- FD 85 E3 85 FC 20 4C 93
4090- C6 E3 A5 E3 C9 4C B0 F3
4098- A9 58 85 E3 20 4C 93 20
40A0- C0 92 C6 E3 A5 E3 D0 F4
40A8- 20 83 92 20 4C 93 20 8D
40B0- 92 20 AC 92 4C 12 40
```