Graphics Workshop Block Shape Animation – II

by Robert R. Devine
P.O. Box 10
Adona, Arkansas 72001

Welcome back!! I hope that sincc ou last discussion you've spent som time working with the SCAN an DRAW routines that we looked at in Part 1 This time we'll start to look at ways to animat your Block Shapes. Before we get to work let's take care of one minor piece of house keeping:

Almost all of the routines in our BLOCI SHAPE DRIVER will depend on other rou tines to complete their work.

For instance to use SCAN or DRAW, w needed to have YTABLE in memory, with it pointers set, and the next routine that we' look at will depend not only on YTABLE, bu SCAN as well. The point is simply this: as w **examine and test our new routines, always b sure that you've added any previously pre sented routines to your driver, and used the YTABLE setup, before trying out the new routines.**

THE REVDIR ROUTINE

Those of you who read the HPLOT article (NIBBLE VOL. 4 No. 2) will remember that we created a **REVDIR routine** which physically reversed the endpoints, and therefore the appearance of our shape. We'll also use a REVDIR routine in our BLOCK shape driver, but we'll not only use it with non-symmetrical shapes, but also with symmetrical shapes that change direction using SHIFT animation, which we'll get into later.

As with our other routines, we'll need to provide REVDIR with VT, VB, HR, HL, and SHAPE#, which will be POKEd into locations 252, 253, 254, 255, and 251 respectively.

HOW REVDIR WORKS

To reverse a shape, we need to reverse not only the position of the shape bytes within the table, but also the position of the bits within each byte. We also need to be sure that we're only reversing bits 0-6, and that we reset bit #7, the color bit, to its previous status.

Since the creation, or modification of a BLOCK shape takes place on the Hi-Res screen, rather than within the table, we'll actually modify the Hi-Res screen bytes, and when we're finished SCAN the modified shape into our table.

Here's how we do it. First we start out by accessing the screen bytes **backwards**. In other words, instead of starting at **VB/HR** (normal for **ALL** other routines), we begin at **VB/HL**, so that we put the shape bytes on the screen in reverse order.

Each time we get a shape byte from our table, we'll check to see if it needs to be reversed. A byte where bits 0-6 are all 0's or all 1's doesn't need reversing so we can skip over it. If you take a look at our sample shape, (SPACESHIP from NIBBLE Vol. 4. No. 3), you'll note that 16 of the 36 bytes fit one of these two categories and don't need revers-ing, so we can save some time with these tests.

Then we'll loop through lines 1250 thru 1290 seven times (in **LISTING 1**). Each time, we push a bit off our shape byte, (out of bit 0) into the carry flag, and immediately pick it up and push it into our temporary NUBYTE $F9 (into bit 0).

Finally we push out the color bit, and check to see if it was set. If it wasn't, we're finished, and if it was, we reset it in line 1306.

The modified byte is then displayed on the screen, and we move on to the next byte. When we've finally taken care of byte VT/HR, we're finished, at which time we SCAN the reversed shape into our table. There's no need to erase or reDRAW the shape because REVDIR has taken care of that for us. (If you're using page flip animation you'll probably be best advised to ERASE the pre-reversed shape on one page and reverse it on the other)

The routine is really pretty neat, especially since there's no ERASE involved. One second the shape faces one way, and without your really noticing the change, the shape now faces the other way.

To use REVDIR simply POKE VT, VB, HR, HL, and SHAPE#, and then CALL 37606. Since our sample shape is symmetrical, it isn't very good for testing REVDIR.

THE TRUCK DEMO

I've provided the following shape that might be handy for a demo. The shape is 90 bytes long (15 bytes high X 6 bytes wide) and should show how quickly you can reverse even a larger shape.

The easy way to get the shape into memory is to enter the monitor, then enter the Hex bytes listed (**LISTING 2**), the same way you've been entering the driver. The shape is a genuine Arkansas Good-Ole-Boy pickup truck, complete with 4-wheel drive C.B., and a shotgun mounted in the rear window. Would I put you on??

LET'S TEST REVDIR

Now that you've loaded **REVDIR (LISTING 1)** and our new sample **TRUCK shape**, LISTING 2) enter these Applesoft lines . . .

```
100 HGR:VT=130:VB=144:HR=25:HL=20:
    SHAPE=146

110 POKE 252, VT:POKE 253,VB:POKE
    254,HR:POKE 255,HL:POKE 251, SHAPE

120 CALL 37679
```

```
*9200.9259

9200-   00  78  00  00  1E  00  01  7C
9208-   00  00  3F  00  01  7C  00  00
9210-   3F  00  0D  7D  7F  7F  3F  3E
9218-   1E  79  7F  7F  1E  3E  1E  03
9220-   7F  7F  40  7E  1F  7F  7F  7F
9228-   7F  7E  1F  7F  7F  7F  7F  7E
9230-   1F  7F  7F  7F  7F  7E  0F  7F
9238-   7F  7F  7F  7E  00  00  60  18
9240-   00  00  00  00  70  18  00  00
9248-   00  00  30  18  00  00  00  00
9250-   38  18  00  00  00  00  1F  78
9258-   00  00
```

LISTING 2
SAMPLE SHAPE (TRUCK)
SHAPE #146

Your shape should now be on the screen. Now **CALL 37606** and watch the shape reverse.

To see just how fast it works, add this line:

```
130 CALL 37606:GOTO 130
```

About all you'll see is a blur, as the shape keeps reversing back and forth.

Animation of BLOCK shapes is very similar to those that we discussed when we looked at HPLOT shapes. There are however two very important differences that we should take a moment to look at.

WHERE IS OUR SHAPE?

When we looked at HPLOT shapes we 'ound that our shape table not only described vhat our shape looked like (in terms of end-oints), but we also found that our table lescribed WHERE our shape was. To move it, ve needed to manipulate the data within our able.

With BLOCK shapes (as with VECTOR hapes) our table only describes what our hape looks like, but includes no information s to where it IS. To move our BLOCK shape p and down we will change the values of VT nd VB. To move right and left we will change he values of HR and HL. It is imperative that he relationship of VT to VB, and HR to HL lways be the same. In other words, when you hange VT you must change VB by the same alue. If you allow the dimension of your lock shape to change, you will distort the hape.

ERASE=DRAW=ERASE=DRAW

The second (and often the most trouble-some) difference with BLOCK shapes is that **here is no true ERASE or DRAW routine. In act, our ERASE and DRAW routines are the ame routine, exactly!**

If you'll recall our discussion of the DRAW/ ERASE routine you'll also remember what appens when we **EOR** our shape byte with he **background** that we want to DRAW/ ERASE on.

If the background is empty we automati-cally DRAW our shape, and if our shape is already there, we automatically ERASE the shape.

When we developed our HPLOT FLIP rou-ines, we used the format Back 1-ERASE, Forward 2-DRAW, without any real care as to whether we really had anything to ERASE. We'll use that same format with our BLOCK shape driver. However we'll need to take care hat when we go to ERASE or DRAW, we don't wind up doing just the opposite of what we expect.

VERTICAL ANIMATION

There will be quite a bit of difference between our routines for vertical and horizon-al movement. Since our vertical routines will be the simplest, we'll begin with them.

The following listing describes our vertical movement routines (**LISTING 3**) which start at **$925E** and fit directly under the REVDIR routine. The routines are broken into 6 parts.

The first two routines are called GOUP and GODOWN. They simply add or subtract our YINCR (which is POKEd into location 227 ($E3)) from both VT and VB. To go up we subtract YINCR, and to go down we add YINCR.

The last 4 routines handle our page flipping, either up or down, and all work basically the same. As with our HPLOT flip routines, they first set the page to display, and the page to DRAW on, then they go back 1 ERASE, then FORWARD 2 DRAW. The only addition to any of these routines is in FLPUP1, where we check to make sure that VT doesn't go off the screen, (which would result in a negative value for VT, and cause your Apple to hang).

The routines are heavily REMed, so there shouldn't be any real need for us to go into detail explaining them. It is these routines that we will use for all our vertical movement. If you're CALLing these routines from an Applesoft program, use the CALL entry points listed in LISTING 3. If you're using the routines from other assembly programs, simply JSR to the appropriate hex entry point.

Each of the FLP routines has several possible entry points. **The first entry point** (lines 1380, 1510, 2380, 2510) sets the display page, draws the page, moves back 1, erases, moves forward 2, Draws, and ends.

The second entry point (lines 1400, 1530, 2400, 2530) eliminates the setting of which page to display. (We'll use some of these points to keep our DRAW/ERASE routines working properly.)

The third entry point (lines 1450, 1550, 2450, 2550) could be used with multiple shape animation if you're going to do more work on the same page setup.

Another possible entry point (lines 1480, 1580, 2480, 2580) would cause you to simply move and DRAW/ERASE (whatever's appropriate). We'll also use some of these points to keep our DRAW/ERASE routines working properly.

There are many ways that you can enter and exit these routines to make them perform different functions.

In BLOCK SHAPES part 1, we experimented with some one page animation. If you're going to go through the effort of learning how to work with BLOCK shapes, you're probably pretty serious about graphic animation, and will mostly work with page-flipping, so that's where we'll put most of our emphasis.

TWO PAGE FLIP ANIMATION

To use flip animation we'll need to learn about several **soft switches**, located in your Apple that determine which graphics page we will DISPLAY and which page we will DRAW on. It is not necessary to have a graphics page displayed on the screen to draw on it. In fact, you can change the graphics on a Hi-Res screen while in TEXT mode.

PAGE DRAWING

The first switch that we'll need to look at is located at $E6 (230 decimal). This switch tells your Apple which page to DRAW on. If the value $20 (32 decimal) is stored there, your Apple will draw on page 1 HGR. If the value $40 (64 decimal) is stored there, your Apple will draw on page 2 HGR2. You can use POKE 230,32 for page 1 and POKE 230,64 for page 2.

PAGE DISPLAY

The next switches that we need to look at are located at $C054 (decimal 49236), and $C055 (decimal 49237). These switches tell your Apple which page it is to display.

To change the page that is displayed you POKE a 0 into one of these switches, and the switch most recently POKEd will determine which page is displayed.

POKE 49236,0 displays page 1
POKE 49237,0 displays page 2

In page flip animation we first display page 1, and draw on page 2. When we enter a flip routine, the values of VT, VB, HR, and HL are always those of the page which is being displayed.

We first move back (on the hidden page) one YINCRement and ERASE our shape, then move forward 2 YINCRements and DRAW the shape, at which time we exchange pages and repeat the process.

SAMPLE PAGE FLIP ANIMATION

Let's start out with an example of page-flip which moves our shape up and down on the screen. To run this test you'll need the SPACE SHIP shape in **LISTING 4** (which we created in Part 1), and our driver in memory.

LISTING 1
REVDIR ROUTINE SOURCE-CODE

```
:ASM

                   1000  .OR $92E6
                   1010  .TA $800
00FC-              1030 VT  .EQ $FC          ** DECIMAL 252
00FD-              1040 VB  .EQ $FD          ** DECIMAL 253
00FE-              1050 HR  .EQ $FE          ** DECIMAL 254
00FF-              1060 HL  .EQ $FF          ** DECIMAL 255
0026-              1070 HBASL  .EQ $26       ** DECIMAL 38  (SCREEN BASE
0027-              1080 HBASH  .EQ $27       ** DECIMAL 39  ADDRESS)
0006-              1090 YO  .EQ $6           ** DECIMAL 6
00F9-              1100 NUBYTE  .EQ $F9      ** DECIMAL 249
00FA-              1110 BASL  .EQ $FA        ** DECIMAL 250 (TABLE BASE
00FB-              1120 BASH  .EQ $FB        ** DECIMAL 251 ADDRESS)
9391-              1130 YADDR  .EQ $9391     ** DECIMAL 37777 (READ YTABLE)
9361-              1140 SCAN  .EQ $9361      ** DECIMAL 37729
92E6- A9 00        1150 REVDIR LDA #0        ** CALL 37606 TO ENTER
92E8- 85 FA        1160  STA BASL            ** POINT TO START OF TABLE
92EA- A5 FD        1170  LDA VB              ** GET BOTTOM Y-COORDINATE
92EC- 85 06        1180  STA YO              ** STORE IN $6 FOR USE BY YADDR
92EE- 20 91 93     1190 L1A JSR YADDR        ** RETURNS-LO=HBASL/HI=HBASH
92F1- A4 FF        1200  LDY HL              ** SET Y-REG TO LEFTMOST BYTE
92F3- A2 00        1210 L2A LDX #0           ** SET TABLE OFFSET=0
92F5- A1 FA        1220  LDA (BASL,X)        ** GET SHAPE BYTE FROM TABLE
92F7- C9 7F        1221  CMP #127            ** IS BYTE 01111111 ? ($7F)
92F9- F0 15        1222  BEQ J1              ** YES-NO NEED TO REVERSE
92FB- C9 01        1224  CMP #1              ** IS BYTE 00000000 ? ($00)
92FD- 90 11        1226  BCC J1              ** YES-NO NEED TO REVERSE
92FF- 86 F9        1240  STX NUBYTE          ** SET ALL BITS TO ZERO
9301- 4A           1250 NXTBIT LSR           ** PUSH BIT OFF SHAPE BYTE -->
9302- 26 F9        1260  ROL NUBYTE          ** PUT BIT IN REVERSED BYTE <--
9304- E8           1270  INX                 ** BUMP BIT COUNTER
9305- E0 07        1280  CPX #7              ** HAVE WE DONE BITS 0-6 ?
9307- 90 F8        1290  BCC NXTBIT          ** NO-GOTO NEXT BIT
9309- 4A           1295  LSR                 ** PUSH HI BIT INTO THE CARRY
930A- A5 F9        1300  LDA NUBYTE          ** GET THE REVERSED BYTE
930C- 90 02        1305  BCC J1              ** IF HI BIT WAS 0-JUMP
930E- 09 80        1306  ORA #$80            ** SET THE HI BIT
9310- 91 26        1310 J1 STA (HBASL),Y     ** LOAD REVERSED BYTE ON SCREEN
9312- C8           1320  INY                 ** POINT TO NEXT BYTE -->
9313- E6 FA        1340  INC BASL            ** POINT TO NEXT TABLE ELEMENT
9315- D0 02        1350  BNE NC2             ** IF <256 BYTES JUMP
9317- E6 FB        1360  INC BASH            ** PAGE OVERFLOW-GOTO NEXT PAGE
9319- C4 FE        1370 NC2 CPY HR           ** HAVE WE PASSED HR YET ?
931B- 90 D6        1380  BCC L2A             ** NO-GET THE NE:T BYTE
931D- F0 D4        1390  BEQ L2A             ** NO-BUT THIS IS LAST BYTE
931F- C6 06        1410  DEC YO              ** MOVE UP TO NEXT LINE
9321- A5 06        1420  LDA YO              ** GET NEW Y-COORDINATE
9323- C9 FF        1430  CMP #$FF            ** HAS Y-COORDINATE REACHED 0 ?
9325- F0 04        1440  BEQ RTN2            ** YES-WE'RE FINISHED
9327- C5 FC        1450  CMP VT              ** HAVE WE PASSED VT ?
9329- B0 C3        1460  BCS L1A             ** NO-START THE NEXT LINE
932B- 20 61 93     1470 RTN2 JSR SCAN        ** DONE-REVISE BLOCK TABLE
932E- 60           1480  RTS                 ** EXIT ROUTINE
```

LISTING 4
BLOCK SPACE SHIP

```
9000-  00  0C  00  00  7F  40  03  7F
9008-  70  07  7F  78  0F  7F  7C  1F
9010-  7F  7E  39  4C  67  3F  7F  7F
9018-  0F  7F  7C  03  7F  70  00  7F
9020-  40  00  1E  00  02  01
```

LISTING 5
PAGE FLIP TEST

```
10  HGR : HGR2 : CALL 37799
20  POKE 251,144:YINCR = 2: POKE 2
    27,YINCR
30  HR = 2:HL = 0: POKE 254,HR: POKE
    255,HL: REM START SHAPE ON LEFT
    SIDE OF SCREEN
35  POKE 252,2: POKE 253,13: REM SET
    INITIAL VALUES FOR VT/VB
36  POKE 230,64: CALL 37524: REM
    DRAW AT VT-YINCR/PAGE 2
37  POKE 230,32: CALL 37574: REM
    DRAW AT VT/PAGE 1
45  FOR X = 1 TO 40
50  CALL 37556: CALL 37581: REM USE
    FLPDN ROUTINES
60  NEXT
65  CALL 37556: CALL 37470: REM DRAW
    AT VT-YINCR AND VT+YINCR/PAGE
    2-RETURN TO VT
70  FOR X = 1 TO 40
80  CALL 37500: CALL 37531: REM USE
    FLPUP ROUTINES
90  NEXT
100 CALL 37561: CALL 37470: REM DRAW
    AT VT-YINCR AND VT+YINC
    R/PAGE 2-RETURN TO VT
110 GOTO 45
```

Uses

1) Block Shape #144

2) Block Routines

3) Listing 1

Once this program is in memory simply RUN it, and you'll see your shape move quickly and smoothly up and down on the screen.

To see just how it works, we'll refer to **listing 5** as well as **figures 1 and 2** which pictorially represent what happens. Note: While figures 1 and 2 only show 5 loops, our program actually makes 40 loops.

Before we can execute any vertical move routine with the driver, we must tell the driver just how many dots we want to move each time. This is done by POKEing our YINCRement into location $E3 (decimal 227)

POKE 227,YINCR

In listing 5 we will start our shape at Y-coordinate 2, and move our shape 2 dots per move. When we say that we're starting at Y-coordinate 2, we're really referring to the value of VT. VB will always be changing, right along with VT, and in this case will always equal VT+11.

Line 10 clears the Hi-Res screens, and executes the YTABLE setup.

Line 20 sets our shape#, and tells the driver that we want YINCR=2.

Line 30 establishes the locations of HR and HL on the left edge of the screen.

Line 35 sets up the starting location of our shape at VT=2, VB=13.

At this point we come upon our first problem namely making sure that our very first loop works properly. Looking at figure 1, you'll note that the first trip through our flip routines will ERASE at 0 and DRAW at 4 on page 2, then ERASE at 2 and DRAW at 6 on page 1.

The only problem with this is that since there really wasn't anything to ERASE at 0 page 2, or 2 page 1, what has happened is that we've DRAWn our shape 4 times — twice on each page. Loop 2 will properly ERASE the shapes at 4 and 6. However, the shapes at 0 and 2 will remain, flashing on and off as we exchange pages. To correct this problem we'll first need to DRAW at 0 page 2, and 2 page 1, so that our very first loop has something to ERASE.

Line 36 sets up to DRAW page 2, then CALLs the driver to GOUP and DRAW.

Line 37 sets up to DRAW page 1, then CALLs the driver to GODOWN and DRAW.

Now the first loop will work properly, as will the rest of our loops until we're ready to stop and turn around.

FLIP DOWN TO UP

The next problem that we need to deal with is changing from the FLPDOWN to the FLPUP routines. You'll see from figure 1 that we ended our last loop with the shape DRAWn at 20 page 2, and 22 page 1. Now looking at figure 2 you'll see that the very first loop going up ERASEs at 24 and DRAWs at 20 page 2, however since there is no shape at 24 the ERASE is really a DRAW, and since there is a shape at 20 our DRAW is really an ERASE.

LISTING 3
VERTICAL MOVEMENT ROUTINES

```
:ASM

                 1000  .OR $925E
                 1010  .TA $800
00FC-            1030 VT .EQ $FC          ** DECIMAL 252
00FD-            1040 VB .EQ $FD          ** DECIMAL 253
00FE-            1050 HR .EQ $FE          ** DECIMAL 254
00FF-            1060 HL .EQ $FF          ** DECIMAL 255
0026-            1070 HBASL .EQ $26       ** DECIMAL 38  (SCREEN BASE
0027-            1080 HBASH .EQ $27       ** DECIMAL 39  ADDRESS)
0006-            1090 YO .EQ $6           ** DECIMAL 6
00F9-            1100 NUBYTE .EQ $F9      ** DECIMAL 249
00FA-            1110 BASL .EQ $FA        ** DECIMAL 250 (TABLE BASE
00FB-            1120 BASH .EQ $FB        ** DECIMAL 251 ADDRESS)
9391-            1130 YADDR .EQ $9391     ** DECIMAL 37777 (READ YTABLE)
9361-            1140 SCAN .EQ $9361      ** DECIMAL 37729
932F-            1150 DRAW .EQ $932F      ** DECIMAL 37679
932F-            1160 ERASE .EQ $932F     ** DECIMAL 37679
00E3-            1170 YINCR .EQ $E3       ** DECIMAL 227
925E- 38         1200 GOUP SEC            ** CALL 37470 TO ENTER
925F- A5 FC      1210  LDA VT             ** GET VT
9261- E5 E3      1220  SBC YINCR          ** VT=VT-YINCR
9263- 85 FC      1230  STA VT             ** RESET VT
9265- 38         1240  SEC
9266- A5 FD      1250  LDA VB             ** GET VB
9268- E5 E3      1260  SBC YINCR          ** VB=VB-YINCR
926A- 85 FD      1270  STA VB             ** RESET VB
926C- 60         1280  RTS                ** DONE
926D- 18         1290 GODOWN CLC          ** CALL 37485 TO ENTER
926E- A5 FC      1300  LDA VT             ** GET VT
9270- 65 E3      1310  ADC YINCR          ** VT=VT+YINCR
9272- 85 FC      1320  STA VT             ** RESET VT
9274- 18         1330  CLC
9275- A5 FD      1340  LDA VB             ** GET VB
9277- 65 E3      1350  ADC YINCR          ** VB=VB+YINCR
9279- 85 FD      1360  STA VB             ** RESET VB
927B- 60         1370  RTS                ** DONE
927C- A9 00      1380 FLPUP1 LDA #0       ** CALL 37500 TO ENTER
927E- 8D 54 C0   1390  STA $C054          ** DISPLAY PAGE 1
9281- A9 40      1400  LDA #$40           ** CALL 37505 TO ENTER
9283- 85 E6      1410  STA $E6            ** DRAW PAGE 2
9285- A5 FC      1420  LDA VT             ** GET VT
9287- C5 E3      1430  CMP YINCR          ** WILL WE GO OFF SCREEN?
9289- 90 0F      1440  BCC EXIT           ** YES-CANCEL ROUTINE
928B- 20 6D 92   1450  JSR GODOWN         ** MOVE BACK
928E- 20 2F 93   1460  JSR ERASE          ** ERASE SHAPE
9291- 20 5E 92   1470  JSR GOUP           ** RETURN TO START
9294- 20 5E 92   1480  JSR GOUP           ** MOVE AHEAD 1 YINCR
9297- 20 2F 93   1490  JSR DRAW           ** DRAW SHAPE
929A- 60         1500 EXIT RTS            ** DONE-EXIT ROUTINE
929B- A9 00      1510 FLPUP2 LDA #0       ** CALL 37531 TO ENTER
929D- 8D 55 C0   1520  STA $C055          ** DISPLAY PAGE 2
92A0- A9 20      1530  LDA #$20           ** CALL 37536 TO ENTER
92A2- 85 E6      1540  STA $E6            ** DRAW PAGE 1
92A4- 20 6D 92   1550  JSR GODOWN         ** MOVE BACK
92A7- 20 2F 93   1560  JSR ERASE          ** ERASE SHAPE
92AA- 20 5E 92   1570  JSR GOUP           ** RETURN TO START
92AD- 20 5E 92   1580  JSR GOUP           ** MOVE AHEAD 1 YINCR
92B0- 20 2F 93   1590  JSR DRAW           ** DRAW SHAPE
92B3- 60         1600  RTS                ** DONE-EXIT ROUTINE
92B4- A9 00      2380 FLPDN1 LDA #0       ** CALL 37556 TO ENTER
92B6- 8D 54 C0   2390  STA $C054          ** DISPLAY PAGE 1
92B9- A9 40      2400  LDA #$40           ** CALL 37561 TO ENTER
92BB- 85 E6      2410  STA $E6            ** DRAW PAGE 2
92BD- 20 5E 92   2450  JSR GOUP           ** MOVE BACK
92C0- 20 2F 93   2460  JSR ERASE          ** ERASE SHAPE
92C3- 20 6D 92   2470  JSR GODOWN         ** RETURN TO START
92C6- 20 6D 92   2480  JSR GODOWN         ** MOVE AHEAD 1 YINCR
92C9- 20 2F 93   2490  JSR DRAW           ** DRAW SHAPE
92CC- 60         2500  RTS                ** DONE-EXIT ROUTINE
92CD- A9 00      2510 FLPDN2 LDA #0       ** CALL 37581 TO ENTER
92CF- 8D 55 C0   2520  STA $C055          ** DISPLAY PAGE 2
92D2- A9 20      2530  LDA #$20           ** CALL 37586 TO ENTER
92D4- 85 E6      2540  STA $E6            ** DRAW PAGE 1
92D6- 20 5E 92   2550  JSR GOUP           ** MOVE BACK
92D9- 20 2F 93   2560  JSR ERASE          ** ERASE SHAPE
92DC- 20 6D 92   2570  JSR GODOWN         ** RETURN TO START
92DF- 20 6D 92   2580  JSR GODOWN         ** MOVE AHEAD 1 YINCR
92E2- 20 2F 93   2590  JSR DRAW           ** DRAW SHAPE
92E5- 60         2600  RTS                ** DONE-EXIT ROUTINE
```

Obviously we need to make a patch here as well.

Line 65 doesn't need to set the draw page, since we're still on page 2, so we simply CALL the driver to GOUP and ERASE, then GODOWN 2 and DRAW, after which we CALL the driver to GOUP (move only) and return to the starting place. Now the first loop up will work properly, as well as the other loops, until we need to turn around again.

TURNING AT THE TOP

This time when we're at the top, our problem is a little different than it was when we started the loop. Again we have the same turn-around problem that we had at the lower part of the loop, i.e. that page 2's first ERASE and DRAW won't work correctly in the first loop. So again we need to setup for the return trip.

Line 100 again doesn't need to set the draw page, so we simply CALL the driver to GOUP and DRAW, then GODOWN 2 and ERASE, after which we again GOUP (move only) which returns us to the starting place.

SAMPLE PAGE FLIP ANIMATION

Now let's try the same program, except that when we return to the top of our loop we'll move our shape 1 byte to the right (7 dots) before starting the next trip down.

LISTING 6

```
 10 HGR : HGR2 : CALL 37799
 20 POKE 251,144:YINCR = 2: POKE
    227,YINCR: REM SET SHAPE#/YINCR
 30 HR = 2:HL = 0: POKE 254,HR: POKE
    255,HL: REM START SHAPE ON LEFT
    SIDE OF SCREEN
 35 POKE 252,2: POKE 253,13: REM SET
    INITIAL VALUES FOR VT/VB
 36 POKE 230,64: CALL 37524: REM
    DRAW AT VT-YINCR/PAGE 2
 37 POKE 230,32: CALL 37574: REM
    DRAW AT VT/PAGE 1
 45 FOR X = 1 TO 40
 50 CALL 37556: CALL 37581: REM USE
    FLPDN ROUTINES
 60 NEXT
 65 CALL 37556: CALL 37470: REM DRAW
    AT VT-YINCR AND VT+YINC R/PAGE
    2-RETURN TO VT
 70 FOR X = 1 TO 40
 80 CALL 37500: CALL 37531: REM USE
    FLPUP ROUTINES
 90 NEXT
100 POKE 230,64: CALL 37574: POKE
    230,32: CALL 37524: REM ERASE VT-
    YINCR/PAGE 2-VT/PAGE 1
105 HR = HR + 1 :HL = HL + 1: POKE
    254,HR: POKE 255,HL: REM MOVE
    RIGHT
110 IF HR < 40 THEN 36: REM IF WE'RE
    NOT OFF SCREEN CONTINUE RIGHT
120 GOTO 30: REM START OVER
```

Listing 6 is really exactly the same as listing 5, except that when we return to the top of the screen we have to be careful not to leave any untidy mess behind us before moving our shape right to its new starting position. When completing loop 5, on the way up we have to ERASE the shapes that we left at 4 page 2, and 2 page 1.

Line 100 CALLs the driver to GODOWN and ERASE, then set up to draw page 1, and finally CALLs the driver to GOUP and ERASE page 1.

Line 105 then moves the shape right 1 byte.

Line 110 checks to see if we're still on the screen, or if it's time to return to the left side and start all over.

FIGURE 1

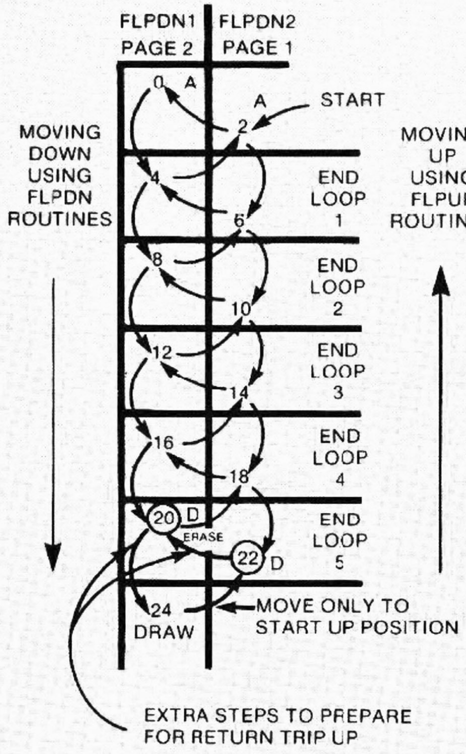
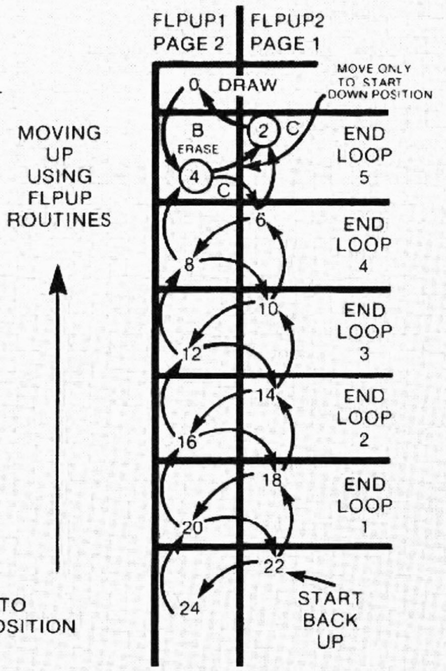

FLPDN1
PAGE 2
FLPDN2
PAGE 1
START
MOVING DOWN USING FLPDN ROUTINES
END LOOP 1
END LOOP 2
END LOOP 3
END LOOP 4
END LOOP 5
ERASE
DRAW
MOVE ONLY TO START UP POSITION
EXTRA STEPS TO PREPARE FOR RETURN TRIP UP

FIGURE 2



A BEFORE STARTING, DRAW SHAPE IN THESE PLACES RETURNING TO START SO THAT LOOP 1 WILL PROPERLY ERASE

B EXTRA STEPS TO PREPARE FOR RETURN TRIP DOWN

THE NUMBERS SHOWN ON EACH PAGE REPRESENT THE PRESENT VALUE OF **VT**. VB WILL ALSO CHANGE IN RELATION TO VT.

C TO ELIMINATE SHAPE AT THIS HL/HR, ERASE AT THESE 2 LOCATIONS,
D OR IN THE CASE OF A **DOWN** MOVING SHAPE YOU COULD ALSO RUN YOUR SHAPE OFF THE BOTTOM OF THE SCREEN.

TABLE 1
SUMMARY OF NEW DRIVER ENTRY POINTS

Routine Name	CALL Address	Hex Address	Routine Function
REVDIR	37606	$92E6	Reverse physical appearance of shape left-right.
GOUP	37470	$925E	Subtract YINCR from VT and VB.
GODOWN	37485	$926D	Add YINCR to VT and VB.
FLPUP1	37500	$927C	Display 1-Draw 2, move down ERASE, up DRAW.
	37505	$9281	Same as above without setting display page.
	37515	$928B	Same as above without setting DRAW page.
	37524	$9294	Move up and DRAW only.
FLPUP2	37531	$929B	Display 2-Draw 1, move down ERASE, up DRAW.
	37536	$92A0	Same as above without setting display page.
	37540	$92A4	Same as above without setting DRAW page.
	37549	$92AD	Move up and DRAW only.
FLPDN1	37556	$92B4	Display 1-Draw 2, move up ERASE, down DRAW.
	37561	$92B9	Same as above without setting display page.
	37565	$92BD	Same as above without setting DRAW page.
	37574	$92C6	Move down and DRAW only.
FLPDN2	37581	$92CD	Display 2-Draw 1, move up ERASE, down DRAW.
	37586	$92D2	Same as above without setting display page.
	37590	$92D6	Same as above without setting DRAW page.
	37599	$92DF	Move down and DRAW only.

POKE 227,YINCRement tells the driver how many vertical dots to move each time.

SPEED COMPARISONS

One of the main reasons that BLOCK SHAPES are so popular in Hi-Res graphics is because of their superior speed. However, no graphics routines can run any faster than the main program allows them to. To see just how much our Applesoft test was slowing us down, I conducted the following test:

First I ran, and timed how long it took listing 6 to move completely across the screen. This test took approximately 70 seconds.

Then I exactly duplicated listing 6, only this time wrote it in machine language. The first run was actually so fast that if you sat in the normal place in front of the screen, you got the impression that you were seeing the shape in several positions at the same time.

Standing back about 10 feet from the screen was better, but obviously something was needed to slow things down.

The next test that I ran was to change YINCR from 2 to 1, and lengthened the loops from 40 to 80, which meant that our test would need to do twice the work to go the same distance. When I ran this test it took 53 seconds to complete the trek across the screen.

In the end this showed that the machine language CALLing program ran **24% faster**, even though it needed to do **twice the work**!

Obviously any speed deficiencies in our tests will not be the fault of the driver. Applesoft CALLing programs will work fine for many uses, and when we're all done, we'll also explore how to use the Ampersand vector to replace CALL statements (which will result in a 15%-20% speed improvement). But for the ultimate in speed you should use the driver with a machine language CALLing program.

Just in case you'd like to try out the speed differences yourself, the following is a hex dump (listing 7) machine language version of listing 6. **To try it out yourself, simply enter it into memory through the monitor, load the driver and your shape, then enter HGR:HGR2 and finally CALL 2048.**

To change it to YINCR=1 and loops=80, enter POKE 2056,1:POKE 2090,80: and POKE 2110,80 before you CALL 2048.

STORING YOUR SHAPES IN MEMORY

There won't be enough space in this issue to get into horizontal movement, so let's take a few seconds to look at how to be more efficient in storing your shapes in memory.

Earlier we said that you must always start your shape at the very first byte of the selected memory page, and that we'll always number our shape with the decimal number of that page. Therefore our sample shape spaceship, although it is only 36 bytes long, took up an entire 256 byte page of memory. If you only have a few shapes, and can spare the wasted memory, that will still be the easiest method.

However, if memory starts running short, you may want to try packing more than one shape on a page. That's what we'll look at here.

If you'll go back to Part 1 (NIBBLE Vol. 4 No. 3), and look at the top part of the SCAN routine source-code, you'll see that **the shape table Base Address is stored in memory locations 250 and 251 ($FA and $FB).** BASH= BASe address Hi-byte, and BASL=BASe address Lo-byte.

What you're doing with your POKE 251, SHAPE# is putting the HI-byte portion of the start address in the BASH pointer. If the shape started at $9300, you'd POKE 251,147 because 147 decimal is the same as $93.

Now if you look at the first two instructions in SCAN, DRAW, and REVDIR you'll see that the very first thing any of these routines do is put a 0 in BASL, at which time BASH=$93 and BASL=00 . . . the start of your table!!

The problem here is that if you wanted to begin your table at some byte other than the first byte of a page, BASL would be destroyed the first time through one of these routines as we INCrement BASL to point to each table element. To fix this, we'll need to store the starting value of BASL somewhere else so that we can reset it to the start of the table each time we enter a routine.

Let's find a place to put BASL. At this point it might be a good idea to turn to page 74 of the new Apple II Reference manua which shows the available Zero page addresses. So far, of the available addresses, we've used 30, 31, 206, 207, 6, 252, 253, 254, 255, 250, 251, 249, and 227. Now we'll add 239 ($EF) to the list, and we'll use it to store the initial value of BASL.

The first thing you'll need to do is change the very first instruction in SCAN, DRAW, and REVDIR from LDA #0 to LDA $EF, which will change the hex bytes from A9 00 to A5 EF. Now the driver routines will be able to handle shapes that start on any byte within the memory page.

CHANGES WITHIN YOUR PROGRAM

Let's assume that the shape you want to work with now starts at **$93A5**.

You would still need to POKE 251,147 because your shape's Hi-Byte is still $93. However you'll also need to POKE 239, 165 because the decimal equivalent of the low byte $A5 is 165. **Using this format you'll always need to enter BOTH POKEs, even if you start a shape at the first byte of a memory page.**

POKE 251,SHAPE# (Hi-Byte)
POKE 239,Lo-Byte

For the balance of our discussion we'll stick with our start-of-page, POKE 251,SHAPE# method, and let you make these changes if you want to.

There are a few things that you'll need to watch out for if you use this multiple shape per page method. Earlier we said that it was okay for your shape to overflow onto the next memory page, meaning that your shape was longer than 256 bytes. It's still okay to let that happen. However, YOUR SHAPE MUST **NEVER** OVERFLOW ONTO THE NEXT MEMORY PAGE, UNLESS IT STARTED AT THE FIRST BYTE OF A PREVIOUS PAGE. Never let any shape that didn't start on the first byte of a page overflow onto the next page.

That about wraps it up for this part of our discussion. Next time around we'll get into horizontal movement using SHIFT animation.

See you then!!

HEX DUMP LISTING 2
MACHINE LANGUAGE VERSION

```
*800.865

0800-   20  A7  93  A9  90  85  FB  A9
0808-   02  85  E3  A9  02  85  FE  A9
0810-   00  85  FF  A9  02  85  FC  A9
0818-   0D  85  FD  A9  40  85  E6  20
0820-   94  92  A9  20  85  E6  20  C6
0828-   92  A9  28  85  F9  20  B4  92
0830-   20  CD  92  C6  F9  D0  F6  20
0838-   B4  92  20  5E  92  A9  28  85
0840-   F9  20  7C  92  20  9B  92  C6
0848-   F9  D0  F6  A9  40  85  E6  20
0850-   C6  92  A9  20  85  E6  20  94
0858-   92  E6  FE  E6  FF  A5  FE  C9
0860-   28  90  B8  4C  0B  08
```