GRAPHICS WORKSHOP - I
HPLOT ANIMATION

HPLOT Animation

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

INTRODUCTION

The age of the personal computer is here, and thousands of proud new Microcomputer owners are being added to the list every day. With the dawn of this new age has come an abundance of software products to feed these hungry devils. For many of us, once the glamour of those prepackaged disks has worn off, we ask ourselves, can I write a program using graphics like that?

If you have reached that point, you've probably already experimented with shape tables, and Applesoft's **DRAW** and **XDRAW** routines. They do very well on small shapes, especially when a series of small (but different) shapes are needed. The main problem with DRAW becomes apparent when you begin to draw and manipulate larger shapes. DRAW is simply too slow. A large shape table can take lots of memory. A vector shape table (the kind described in the Applesoft manual) is difficult and time consuming to create, and is very difficult to modify if you make an error, or want to change it. While they are hard to create, the routines to manipulate them are built into your Apple, so frequently they will still be the best way to go.

There are two other methods of shape creation that are commonly used: the **HPLOT shape**, and the **BLOCK** or **BYTE shape**. In this article we will deal with the HPLOT shape, and in future articles we'll take a look at BLOCK shapes.

HPLOT shapes work best when a **large area** needs to be covered, or when a **large shape** consisting of many straight lines is needed. For want of a better name they're called HPLOT shapes, because they make use of the Applesoft HPLOT and HPLOT TO (HLINE) routines in the creation of shapes.

In this discussion we will first show **how to create a HPLOT shape**. Then we will develop a **machine language driver to manipulate our shapes**. This driver will be simply a tool, and **you can use it in your own programs**. Our finished driver will have many entry points which can be used in the **creation**, **testing**, and **manipulation** of our shapes. Several points within the driver can also be modified by our CALLing program to change its functions.

Finally, we'll try to demonstrate the driver's logic in terms of Applesoft, and see what approaches work, and some that won't.

CREATING SHAPES

Let's start out by looking at different methods for creating a shape.

There are two basic ways that you can define an HPLOT shape. The first is to define a **solid shape (Figure 1)** where all points within the shape are lighted. The second method is to define an **open shape (Figure 2)** where only the outline of the shape is illuminated.

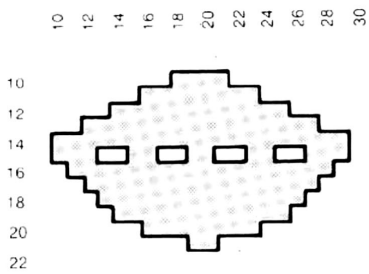


FIGURE 1
SOLID SHAPE

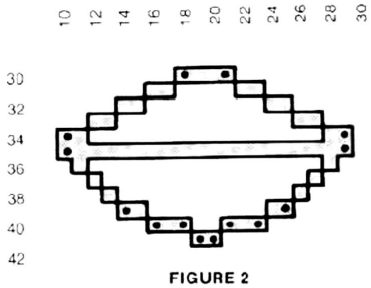


FIGURE 2
OPEN SHAPE

The solid shape in figure 1 is only **20 points wide by 12 points high**, but it would require an **80** to **100** byte **vector shape table** (depending on how you defined it), using the **DRAW** command. It's small enough that **DRAW** might still be the best way to go, but if you make it much larger, the amount of memory required, not to mention the efforts involved, will quickly double or triple. And of course, if you make an error...you'd better begin again.

This is where you will begin to find that **HPLOT** shapes are a better way to go. In our discussion of **HPLOT** shapes, we will be working with the shape defined in **figure 1**. An alien spaceship no less!!

OPEN SHAPES

Drawing an OPEN shape figure is probably the easiest way to proceed. It may run faster than a solid shape since there are often fewer points to define. However, an open shape may lack the desired level of detail. In an **OPEN** shape, a starting point (**X,Y coordinate**) is defined, (in our example it is **10,35**) and **a point is HPLOTed**. Then **another point is selected (such as 29,35)**, and **a line is drawn between the two points**. Lines are then drawn around the figure, always **using the end of the last line as the starting point of the next line**, until the figure is complete. This really amounts to nothing more than a connect-the-dots picture from a childs fun book.

SOLID SHAPES

Creating a SOLID shape is much the same as with an open shape, except that rather than starting the next line where the last line left off, **both the starting and ending points for each line are defined**. You'll note that since we are able to lift our electronic pencil off the paper, it was possible to add windows to figure 1, while figure 2 was simply dissected by a straight line.

CREATING HPLOT SHAPES

For the balance of this discussion we will be dealing in terms of **SOLID shapes**, and using **figure 1** in our examples. Our assembly driver will be able to handle OPEN or SOLID shapes, but to keep things simple, we will stick with SOLID shapes for now. You may in fact use both types of shapes in the same program and still CALL or JSR the same drawing routines. More details later.

SOME BASIC RULES

Let's take a moment to establish some basic rules that we'll use when defining HPLOT shapes and HPLOT shape tables.

1. **Never allow your shape to have more than 85 points.**
2. **Never define your original shape so that any horizontal coordinate is greater than 255.**
3. **Never define your original shape closer to the left or right edges of the screen than you expect your maximum movement increment to be**. In other words, if you expect to move **10** points per move (a rather large move), don't define any **X**-coordinate less than **10** (on a right moving object), or greater than **269** (on a left moving shape).
4. **All HPLOT shape tables are 256 bytes long**, regardless of the number of points in the shape.
5. **Try to make a habit of always placing the leftmost and rightmost endpoints of your shape in the same place within each of your tables**, so that you can always find the present location of your shape.

HOW AN HPLOT TABLE IS ORGANIZED

Okay, let's create the HPLOT shape table for figure 1. I've found that using the Applesoft **DATA statement** is very handy here. The **FIRST** element of your table must be the **number of points** to be defined in the shape. If you defined figure 1 the same way I did, you'll come up with **16 horizontal lines** and **32 endpoints**. When working with **SOLID** shapes there will always be an **even number** of endpoints. When working with **OPEN** shapes there may be an **odd or even number** of endpoints, and you must specify a point every time you see it, even if you're redrawing from or to a previously plotted point.

After the number of points is entered, the first **X (horizontal)** coordinate is specified, **followed by a 0**. Next the **Y (vertical)** coordinate is entered, then back to the next **X** coordinate, another **0**, the next **Y** coordinate, and so on until the table is complete.

The reason for the **0** after each **X** coordinate has to do with a different way of **entering numbers** when they're **greater than 255** (i.e. **Rule #2**), and will make your table easier to enter for use with our assembly driver.

KEEPING TRACK OF YOUR SHAPE

There is one major difference between **HPLOT** and **VECTOR** shapes that you'll need to get used to in order to work with HPLOT shapes. Consider this question...Where **is** the shape that's in the shape table?

With a **VECTOR** shape, the shape **IS anywhere you say that it is** in the **DRAW** statement. (Relative to the **X,Y** coordinates given.)

With an HPLOT shape the shape **IS** exactly **wherever the data within the table says** that it is, and can't be drawn anywhere else.

To **move a VECTOR shape** and keep it on the screen, we manipulate it and test the values for our **X/Y** coordinates. However to do the same thing with an **HPLOT shape** we manipulate and test the **data** (line endpoints) within our shape table. We'll need to place the right and/or leftmost endpoints somewhere that we can find them in our table. Perhaps the first line of your shape extends from one endpoint to the other. At any rate, you should know, and keep a note as to where this information is located in your shape data. You can then **PEEK** at your table to find these endpoints for your tests.

CREATING FIGURE 1

Now examine **Listing 1** and we'll see how to begin putting it together. The DATA statements in **lines 2000-2010** represent the complete shape for **Figure 1**. That's certainly a lot easier that trying to put the shape together by the vector method from the Applesoft Manual.

Line 1900 reserves memory for the shape table and then READs the DATA statements and POKEs them into memory.

Now type **lines 1000-1145** into your Apple and SAVE the partial program on disk. Don't type in **lines 10-20** or **lines 1210-1285** just yet. Having typed in these lines, now type in the following **immediate execution** command:

HGR:HCOLOR=3:GOSUB 1900:GOTO 1000 <**Return**>

If you did everything right your shape should now be on the screen. If the program so far doesn't look very efficient, that's okay. The line numbers, variable names, and logic used are meant to simulate those of the assembly driver which we'll use later.

A special note at this time. You'll see that this routine is designed to draw solid shapes by going back to a new starting point after drawing each line. If we changed the jump in line 1140 from 1015 to 1075, the routine would instead draw an open shape which has the effect of filling in the windows of the space ship. This is the same approach that we'll use to change our assembly routine for drawing either type of shape, a task which can be performed with one simple POKE.

Now type in the remainder of the program, **lines 10-20** and **1210-1285**. **Lines 1210-1285** will move your shape 1 increment to the right. Again, these lines are compatible with our Assembly Driver (later in this article) and they correspond to the MOVER1 routine.

Lines 10-20 physically move the shape across the screen.

When you RUN this program your shape will move from left to right across the screen.

Yes, I know it's slow, but remember we're still working in Applesoft. At this point it would be a good idea to closely examine the Applesoft program lines to be sure you understand what's happening and how the routine works.

From now on we'll be working with the assembly driver which is based on the principles used in the routine which we now have in memory. To save yourself a little work later, I'd suggest that you **SAVE** this Applesoft routine to disk.

RULES 1 AND 4

Inside your Apple, there are memory "PAGES". Every page is **256 bytes** long, and **each of our shapes will take up one complete page**. For instance, the memory area from **$800 — $8FF** (hex) is one page, and **$E00 — $EFF** is another page. **We will always begin our shape tables at the first byte of each page.** For convenience **we will number our shapes with the decimal number of the memory page that they appear on**. For instance the first available page is **$800** and the last available page is **$9300 (shape #147)**. **Note: 147 is the decimal equivalent of $93**. Also note that **shape #147** can only have **83 points** so that it doesn't overrun our machine language driver which starts at **$93FA**.

The reason for the 1 page limit is that after **BASL** or the **Y-register** (our element pointers) exceeds **255** it will roll over to **0**, which points us back at the first byte of our table. If we kept trying to plot points, we'd end up with a real mess.

PLACING SHAPES IN MEMORY

Where is the best place in memory to store your shapes? Good question. Obviously you can't store them on your Hi-Res pages. The assembly driver sits just under DOS. Building shapes down from our driver is the best way to go, protecting them by setting HIMEM under the lowest shape.

WHAT IF YOU NEED MORE THAN 85 POINTS?

If **85** points won't do the job, here are a few suggestions....

1. **Think smaller!** (Smaller will also run faster)
2. **Break your shape into 2 separate shapes.**
3. Sometimes, **drawing vertical rather than horizontal lines** will help, or vice versa.
4. **Never allow your shape to have an X-coordinate greater than 255.** This way you can eliminate all the **0's** in your table, and draw **127** points. You would of course need to modify the driver.
5. Modify the driver to **increment BASH when the element pointer reaches 255.**

Now try these tricks after saving Listing 1 on your disk. First enter **HGR:HCOLOR=3**, then type this line. **FOR X=1 TO 50:HPLOT X,0 TO X,50:NEXT** <**Return**>. You got a big white square, right? But then you already knew that. You covered an area 50 points high and 50 vertical lines wide, requiring 100 endpoints.

Now type **HGR** again and try this: **FOR X=1 TO 50 STEP 2:HPLOT X,0 TO X,50:NEXT**. This time you covered the same area with only 25 lines, requiring only 50 endpoints; however, the color changed to **green**.

Finally enter **HGR** again, and try this: **FOR X=2 TO 50 STEP 2:HPLOT X,0 TO X,50:NEXT**. Again you covered the same area with only 50 endpoints, but now the color is **blue**. The idea is this, if you can live with green or blue for your shape, draw **every other** vertical line and you can reduce the number of points needed. When the lines are on **odd numbered coordinates** the color will be **green**; when they are on **even numbered coordinates** the color will be **blue**.

THE HPLOT DRIVER

At this point I think it would be a good idea if we stopped long enough to enter and save the driver. The assembly source-code listing shown was created with the S-C assembler. If you don't have an assembler, you can simply enter the Hex bytes as listed. To enter the Hex bytes, you'll first need to enter the Monitor with **CALL-151**. Then enter **93FA:FA A9 00 8D 54 C0** etc. until you've filled about **4 lines** on the screen. Then press **RETURN**, enter another colon (:), and fill up another 4 lines. Repeat the process until the entire listing is entered.

To save the completed driver to disk, enter **BSAVE H/P DRIVER $93FA,A$93FA,L$207.** Before we get into working with the driver, you might take a few minutes to look over the summary that follows. The names of all the routines, as well as their functions and entry points, are listed in **Table 1**. You'll also find a list of all the **POKE**s that you may need to use with the driver in **Table 2**.

For more information on entering machine language code directly into memory, see the Letters section of this issue.

HOW THE DRIVER WORKS

The driver listing is heavily documented, so even those of you who don't write machine language should be able to follow the listing with little trouble. All the routines work with the same approach as the Applesoft program that we've already tried out. Basically each routine sets up at the **first byte** of the page where the shape is found (your shape# told it which page), gets the number of points, and puts that value in the counter.

The balance of the routines are used to increment the pointers which, one-by-one, step through all the elements in the table, each time checking the counter to see if all the points have been processed. The move routines add to or subtract from the increment, depending on the direction of travel. The **DRAW/ERASE** routines first **HPLOT** a point, then drop through and **HPLOT TO** the next point, at which time a check is made (**line 1140**) to see if we're drawing **SOLID** or **OPEN** shapes, and the appropriate jump is made; either to **HPLOT** where a new line is started, or **HPLOT TO** where we continue drawing from our present point.

THE DRIVER IN MEMORY

We'll be making use of our **driver** from now on, so **you should have it in memory**. Because of its location, just below DOS, you can safely enter any Applesoft program without damage to the driver. The one exception has to do with strings. **DO NOT execute any string or CHR$( ) type statements unless you've first protected the driver, and your shapes, by setting HIMEM.** Strings are stored at the top of memory, right where the driver is. To protect the driver and subsequent shapes, type **HIMEM:37120** <Return>.

ENTERING SHAPES

Now let's create an **HPLOT TABLE** of **figure 1**. To save some work, let's reload the Applesoft program we were playing with earlier, and delete all the lines **except** the **DATA** statements in **lines 2000-2010**. This is the same table our machine language routines will use. Let's store our shape on memory **page $147**, the same page where our driver starts. We'll start the shape at **$9300** hex and call it **shape #147.** To place the table in memory enter this line:

```
100 FOR X=37632 TO 37728:READ A:POKE
  X,A:NEXT
```

37632 is the decimal equivalent of **$9300**, and **37728=37632 + (# of points)*3**. When you RUN this line, your table will be stored in memory as **shape #147.** At this point you should save it to disk with **BSAVE SHAPE #147,A$9300,L$61.** Believe it or not, you've just saved your first HPLOT table, and are ready to use the routines available in the assembly driver!

DRAWING SHAPES ON THE SCREEN

For the balance of our discussion it will be necessary that you keep the driver in memory, as well as a shape to work with. In order to use any of the driver routines **you must always POKE the shape number into location 251:**

To view your shape enter **HGR:POKE 251,147:CALL 38102**, and magically your shape should appear. It wasn't even necessary to set HCOLOR. **POKE 251,147** entered our shape# and **CALL 38102** is our **DRAW** routine. Now enter **CALL 38095**. You say the shape disappeared? Of course it did, **CALL 38095** is our **ERASE** routine.

DRAWING AT X-COORDINATES GREATER THAN 255

What if you really wanted your original shape to appear on the right side of the screen, but had only defined it on the left side (to be consistent with Rule #2)? Believe me, consistency was a good choice.

Remember all this stuff you keep reading about the number **255**. It seems that this is some magical number with powers that make it an immovable object!!! That's because **you can't store any number greater than 255 in 1 byte**. A number larger than 255 (up to 65535) needs 2 bytes, and since your screen is 279 points wide, we must allow for X-coordinates being 2 byte numbers. That's the reason for all those extra 0's in your table. If for instance you had selected an X-coordinate of **267**, you would have needed to enter the number **$010B**, the hex equivalent of 267. The **0** in your table would have become a **1**, and the **X byte** would need to be **11**, because **11** is the decimal equivalent of **$0B**. For simplicity, we'll keep our **X-coordinate** under 255.

USING A MOVE ROUTINE

Our shape presently is positioned from horizontal **X-coordinates 10-29**. Now that it's in memory, let's MOVE it to coordinates **248-267**. We want to MOVERight **238 points**, since **267-29=238**. To use any of the movement or animation routines in the driver **you must POKE the MOVEment INCRement into memory location 207:**

POKE 207,INCRement

Enter **POKE 207,238**, and then **CALL 38167** which is the **MOVERight 1** increment routine. This time, when you **CALL 38102** your shape will appear on the right side of the screen.

Wasn't it much easier to let your Apple do all the figuring?

SETTING SHAPE TYPE

The driver is presently set for **SOLID** shapes. To change to **OPEN** shapes **POKE ($9515) 38165,226**. To change back to **SOLID** shapes **POKE 38165,204**. This POKE is only needed when changing from one shape type to another.

SHAPE ANIMATION

By now you should be familiar with our DRAW and ERASE routines, which are used by simply POKEing the shape # into memory location **251**, and **CALLing** the desired routine.

TABLE 1
HPLOT DRIVER SUMMARY

Routine Name	Call Address	Hex Address	Routine function
FLPDN1	37882	$93FA	Display page 1-Move shape down on page 2
	37891	$9403	Secondary entry point
FLPDN2	37907	$9413	Display page 2-Move shape down on page 1
	37916	$941C	Secondary entry point
FLPUP1	37932	$942C	Display page 1-move shape up on page 2
	37941	$9435	Secondary entry point
FLPUP2	37957	$9445	Display page 2-move shape up on page 1
	37966	$944E	Secondary entry point
REVDIR	37982	$945E	Reverse physical appearance of shape left-right
GODOWN	38049	$94A1	Move shape down 1 Y-INCRement
GOUP	38072	$94B8	Move shape up 1 Y-INCRement
ERASE	38095	$94CF	Erase shape
DRAW	38102	$94D6	Draw shape
MOVER1	38167	$9517	Move shape right 1 INCRement
MOVEL1	38196	$9534	Move shape left 1 INCRement
MOVEL2	38225	$9551	Move shape left 2 INCRements
MOVER2	38268	$957C	Move shape right 2 INCRements
FLIPR1	38312	$95A8	Display page 1-Move shape right on page 2
	38321	$95B1	Second entry point
FLIPR2	38334	$95BE	Display page 2-Move shape right on page 1
	38343	$95C7	Second entry point
FLIPL1	38356	$95D4	Display page 1-Move shape left on page 2
	38365	$95DD	Second entry point
FLIPL2	38378	$95EA	Display page 2-Move shape left on page 1
	38387	$95F3	Second entry point

TABLE 2
SPECIAL POKES TO MODIFY THE FUNCTIONS OF THE DRIVER

POKE 38096,COLOR	Set background (ERASE) color.
POKE 38103,COLOR	Set shape color
Use these values for color	
0 ($0)=BLACK 127 ($7F)=WHITE 42 ($2A)=GREEN 170 ($AA)=RED 85 ($55)=BLUE	
213 ($D5)=BLUE2	
POKE 251,SHAPE#	Set shape #, required for *ALL* routines
POKE 207,INCRement	Set # of points to move left/right, required for *ALL* MOVE and FLIP routines.
POKE 253,YINCRement	Set # of points to move up/down, required for *ALL* GO and FLP routines.
POKE 38165,204	Set DRAW/ERASE routines to handle SOLID type shapes.
POKE 38165,226	Set DRAW/ERASE routines to handle OPEN type shapes.
POKE 252,0	Tell REVDIR that endpoints are in X-coordinates 1 & 2.
POKE 252,1	Tell REVDIR that endpoints are in X-coordinates 1 & 3.
POKE 38333,96 POKE 38377,96	Break the horizontal FLIP routines into 4 separate segments for multiple shape use.
POKE 38333,234 POKE 39377,234	Set the horizontal FLIP routines for single shape flip animation
POKE 37906,96 POKE 37956,96	Break the vertical FLIP routines into 4 separate segments for multiple shape use.
POKE 37906,234 POKE 37956,234	Set the vertical FLIP routines for single shape flip animation.

To animate our shapes, we will use the **4 MOVE** routines which will MOVE our shape **1** or **2** INCRements left or right. Bear in mind that **the MOVE routines have no effect on your graphics display; they simply modify the contents of the shape table in preparation for the next DRAW or ERASE command**. Before you use any MOVE routine you must first specify the INCRement by POKEing memory location **207**.

To MOVE our shape to the **RIGHT** we'll use the **MOVER1** and **MOVER2** routines, and to MOVE **LEFT** we'll use **MOVEL1** and **MOVEL2**.

SIMPLE ONE PAGE ANIMATION

The same basic principles that you've probably used with VECTOR shapes will be the same ones you'll use with HPLOT shapes, except that instead of incrementing or decrementing the **X value** used in the **DRAW ..AT X,Y,** you will instead increment or decrement your shape location by using the **MOVE** routines. Let's try out a simple test where we move our shape back and forth across the screen. First **BLOAD** the **H/P DRIVER** and your shape into memory, and then enter and RUN the program in **Listing 2.**

In **line 100** we set the necessary POKEs for the driver. The balance of the program simply ERASEs the shape and tests to see if we're at the edge of the screen. If we're not at the edge, we **MOVE** the shape and **DRAW** it. If we are at the edge, we simply **MOVE** in the **other** direction and then **DRAW**. Spend a few minutes playing with the routine. You'll note that **lines 120** and **170** protect you from going off the screen regardless of the INCRement that you set.

The one major problem with our test is that there is still quite a bit of flicker, as the shape is erased and redrawn. Part of this is due to the fact that the shape actually spends more time in the ERASE state than it does in the DRAW state. We could help things quite a bit by including a slight delay between DRAW and ERASE, but that would slow things down, and we're trying to keep our routines as fast as possible.

The solution to the problem is to **only display your shape in the DRAW mode**, and to never let anyone see our ERASE actions. Here's where we really start getting into the good stuff!!

HI-RES PAGE FLIPPING

As you probably know by now, your Apple has **2 Hi-Res screens**. The first is called **Page 1 (HGR)**, and the second is called **Page 2 (HGR2)**.

When you select one of these pages with **HGR** or **HGR2**, you are telling your Apple that you wish to graphically display the contents of a specific area of Hi-Res memory. You also tell it that any graphics commands that it encounters are to be executed upon the selected Hi-Res page.

The most important concept that you'll need to understand is this: **You don't need to have a Hi-Res screen visibly displayed on the screen to draw on it**. There is absolutely no restriction that prevents you from **drawing or erasing a shape on one Hi-Res screen** while you **display the other Hi-Res screen**.

This is all accomplished with what are referred to as **soft switches**. There are several soft switches; however there are only **3** that we will be concerned with.

WHICH PAGE WILL WE DRAW ON?

The first switch we need to be aware of is located at **memory location $E6 (decimal 230)**. The value stored in this location tells your Apple **which Hi-Res page to DRAW on**. If the value stored there is **$20 (decimal 32)** then your Apple will DRAW on **page 1**. However if the value is **$40 (decimal 64)** it will DRAW on **Hi-Res Page 2**.

For those of you who like to contemplate new ideas, here's an idea to play with: If you were to store a **$60 (decimal 96)** in this location you would DRAW on **PAGE 3 ($6000-$7FFF)**. While you can't display page 3, you could DRAW there, and then using the Monitor MOVE command, move page 3 onto one of the other 2 pages for display. Again you should remember that the page we're DRAWing on, can be independent of the page we're displaying at any particular moment.

WHICH PAGE WILL WE DISPLAY?

The next two switches that we'll need to learn about are located at **$C054 (decimal 49236)** and **$C055 (decimal 49237)**. These switches tell your Apple which page to **DISPLAY**. If we want to look at a page we would put a **0** into one of these locations. The command **POKE 49236,0 displays Page 1**, and **POKE 49237,0 displays page 2**. Your Apple will display whichever page whose switch was accessed last.

With the information in hand as to which page to DISPLAY, and which page to DRAW on, it is now possible for us to **display one page, while we draw on the other.**

LET'S PLAY WITH SOME PAGE FLIPPING

First let's draw something on each Hi-Res page and see what it looks like with page flip.

```
100 HCOLOR=3: HGR: HPLOT 50,50 TO
    50,60: REM DRAW A 1 ON PAGE 1
110 HGR2: HPLOT 100,50 TO 110,50 TO
    110,55 TO 100,57 TO 100,60 TO 110,60:
    REM DRAW A 2 ON PAGE 2
120 POKE 49236,0: REM DISPLAY PAGE 1
130 POKE 49237,0: REM DISPLAY PAGE 2
140 GOTO 120
```

When you RUN this little program we first clear each Hi-Res page and draw a number on the page. After that we simply loop through **lines 120-140**, over and over again. While it **appears** that we keep drawing a **1** and a **2** on the screen you'll note that there are no graphics drawing commands in either of these lines. All we're doing is popping back and forth from one page to the other and looking at what's drawn on that page.

You should also note one other very important **difference** between using these "soft switches" to change pages, and using the **HGR** or **HGR2** commands. The standard Applesoft graphics commands set all switches to DRAW and DISPLAY the selected page, but they also **ERASE** whatever was drawn on the page when the **HGR** or **HGR2** command is given. Using the soft switches **does not erase the page**.

When using PAGE FLIP animation we always do the ERASEing, MOVEing, and reDRAWing on the **hidden** page before we flip it into view. In this way **the viewer always sees the graphics in the DRAW state, and never in the ERASE state**.

If your graphics display contains background graphics, you will need to draw them on BOTH pages so that the user will be unaware that page-flip is being used.

Now let's take a look at how we might use our FLIP routines to move our shape across the screen.

Let's see how we would move our shape to the right using page-flip animation. For the moment we'll also assume that we're using **FLIPR1**, which simply means that the EXPOSED PAGE is Page 1 and the HIDDEN PAGE is Page 2. **FLIPR2** does exactly the same thing except that pages 1 and 2 are reversed.

A. This is what we first see (find) when we enter a FLIP routine. Here we **SEE** the shape at **X=20 on the displayed page;** however the shape is also **drawn at X=10 on the hidden page. NOTE:** The shape table coordinates are those of the **EXPOSED** page.

B. Here we need to move **backward 1 increment** (since our direction of travel is right, backwards would be left). Now that we've moved back, we're at the same **X** coordinate as the shape on the hidden page, at which point we ERASE the shape.

C. Now we need to move **forward 2 increments** so that our shape is **1** increment ahead of the shape which we're still looking at on the exposed page. Now that our shape is at **X=30**, we DRAW it on the HIDDEN PAGE.

D. Step D is really the same as **STEP A**, except that **we've exchanged pages**. The Hidden page is now exposed, and the exposed page is now hidden. The shape that we **SEE** on the screen is now at **X=30**, and the shape on the hidden page is at **X=20**. To continue moving across the screen GOTO STEP B.

Above we see a pictorial representation of PAGE-FLIP animation. Now let's see how we would go about writing a short program, using our driver, that will accomplish the task.

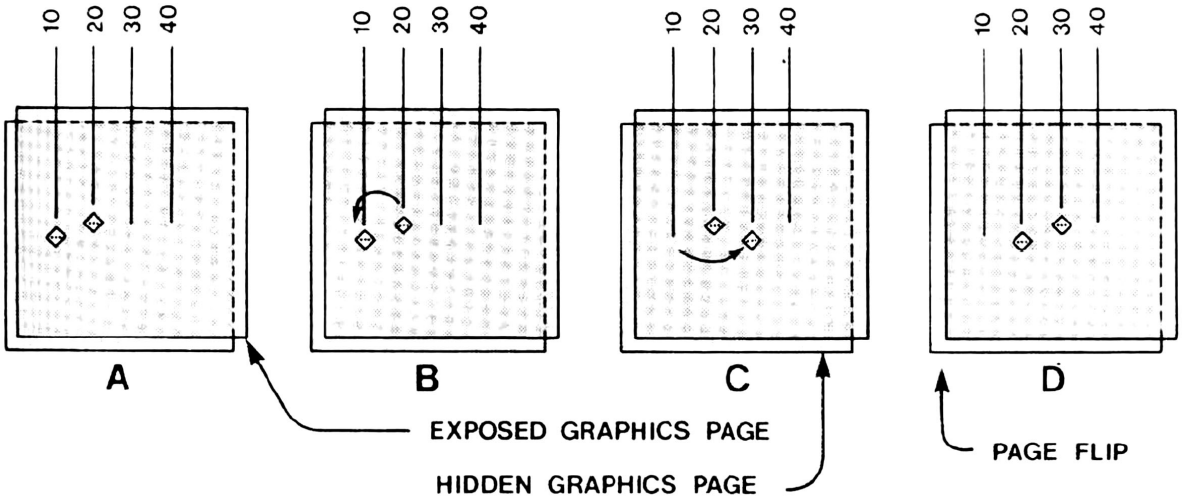
After you have the driver and your shape in memory, type in and RUN the program in **Listing 3** and you'll see your shape move smoothly (without so much as a flicker) back and forth across the screen. This routine is very similar to our previous test, except that instead of CALLing ERASE, MOVE, and DRAW, we're letting the FLIP routines do that for use. The **HPLOT 50,50 to 50,60** in **line 100** simply draws a **1** on **page 1** so that you can see that we really are page flipping.

KEEPING ON THE SCREEN

As you ran the last test, your shape moved from left to right, and then came back the other way. However, when it got to the left edge of the screen the program hung up, right? Without knowing it, we just broke rule #3, which said you should never try to DRAW the shape closer to the edge of the screen than 1 INCRement. The only way to recover is to hit **RESET** and then enter **3D0G**. You'll also need to reload your shape since your table has been destroyed.

Here's what happened. Our **leftmost X-coordinate** started at **10**, and as we moved right it went to **14, 18, 22** and so on. On the return trip it dropped from **18, 14, 10, 6 to 2**. When it was at **6**, our test in **line 140** let it make **1 more move** (we're testing for 0+4=4) and it dropped to **2**. Now when it came to **line 140** again we were sent to **line 120**, which is our **FLIPR1** routine. If you remember **step B** in our example, you'll note that the first thing our FLIP routines do is move backwards to erase our shape. In this case **moving backwards from 2 meant moving off the screen to -2.**

EXAMPLE: Moving our spaceship from left to right, moving 10 points per move.



The fix is easy. Reload your shape. Type **POKE 207,6**, and **CALL 38196**. What we've done is to move the starting position of our shape from **10** to **4** to that our **line 140 test** will properly reverse us. Now RUN the program again and everything should work perfectly.

You should notice that page flipping is simple with the driver. In fact it took fewer commands to write our page-flip test than it did with our one-page test. The movement of our shape is much smoother as well.

To save yourself problems in the following tests, I'd suggest that you reload the shape. Then type **POKE 207,6:CALL 38196** again, and reSAVE your shape at the new coordinates.

MULTIPLE SHAPE ANIMATION

At this point we looked at how to animate shapes on **one page** and on **two pages** using page-flip. Now let's see how we might move more than one shape on the screen.

The first thing that you'll need to do is **create another shape**. I'd suggest that you create another version of figure 1. This time, however, you should run the new shape from **vertical** coordinates **30-41** and **horizontal** coordinates **253-272**. (With these coordinates our future tests won't crash.)

Let's call this shape **#146**, which means you'll start your table at **$9200 (37376 decimal)**.

Here's the easy way to do it. First be sure that the **driver** and **SHAPE #147** are in memory, and then Type **HGR: POKE 251,147:CALL 38102**. This should put our present shape on the screen at **Y** coordinates **10-21**, and **X** coordinates **4-23**. (You did move and reSAVE it as we suggested, didn't you?)

Now type **POKE 253,20:CALL 38049** which will move the shape DOWN to coordinates **30-41**. We haven't discussed vertical movement yet, but if you look at the driver summary you'll find that **POKE 253,YINCR** handles vertical movement, and **CALL 38049 moves our shapes DOWNWARD** on the screen.

If you want to make sure the shape made the trip safely, enter **CALL 38102** again to see the shape in its new location.

Now enter **POKE 207,249:CALL 38167**. By now you should recognize that this will move the shape right **249** points. One last time, enter **CALL 38102** to be sure the shape made the trip okay.

Now that we have the shape table where we want it, the problem is that it's on the wrong page! No big deal, let's move it to the correct page with your Apple's Monitor **MOVE** command. To do so, first enter **CALL-151** to get into the Monitor. Then enter **9200<9300.9360M <RETURN>** and your shape will move where we want it.

Finally enter **3D0G** to return to Applesoft and save the shape with **BSAVE SHAPE #146,AS9200,LS61.** This may seem like a round-about method, but it made use of new parts of the driver and a Monitor command. To be sure you did everything correctly, type **HGR** to clear the screen, and then **POKE 251,146: CALL 38102.** You should have been able to CALL your shape as SHAPE #146.

DIFFERENCES WITH MULTIPLE SHAPE ANIMATION

With multiple shape animation you'll need to **ERASE, MOVE,** and **DRAW all the shapes** being animated, on the hidden page, **before** flipping it into view. If you don't do this, the viewer will see all our back 1, forward 2, maneuvers on both screens for each shape.

MODIFYING THE FLIP ROUTINES

As the FLIP routines are presently written, we first **display page 1 and draw page 2**, at which time we **FLIP the pages and display page 2 while drawing page 1**. What we need to do is **break out of the FLIP routine** after our move on page 2 so that we can repeat the process with our other shape(s), before exchanging pages.

To do this we will replace the **NOP (NO oPeration)** codes in **lines 1656** and **1856** with **RTS (ReTurn)** codes, effectively **breaking the 2 FLIP routines into 4**. This is accomplished with the following two POKEs: **POKE 38333,96** and **POKE 38377,96**. To replace the **NOP**s for **single shape** use, Type: **POKE 38333,234** and **POKE 38377,234.**

A SAMPLE TWO SHAPE ANIMATION ROUTINE

To run the next test, you'll need to be sure that you have the **driver** as well as **both shapes** in memory (SHAPE #146 and SHAPE #147). Then simply enter the program in **Listing 4** and RUN it.

You can see from this program how a multiple shape routine CALLs the first and second parts of the FLIP routines separately to get the needed effect.

ALTERNATE ENTRY POINTS

As you looked at the driver summary, you probably noticed that all of our **FLIP** and **FLP** (for vertical flipping) routines showed a

secondary entry point. We didn't use them in our test. However now is a good time to see what they do.

If you'll take a moment to look at the driver source-code, you'll see that the first thing that is done in each part of a FLIP routine is to set the numbers of page to **display** and the page on which to **draw**. The first time we set up to work on a page we need to execute those instructions (i.e. **lines 220,260,330,** and **370**). However if we're coming to the **same** page set-up again we really don't need to waste execution time repeating the same instructions. So in our example **we could have substituted the secondary entry points in our CALLs in lines 240, 280, 350, and 390.**

THE REVDIR ROUTINE

Now let's take a look at a specialized routine that we haven't talked about before. So far we've only been dealing with nice symmetrical shapes, but what about when you need a **non-symmetrical shape** that requires a **left-facing and right-facing version of the same shape?** This is where **REVDIR** comes in very handy. You could create separate shape tables for each version, but that takes memory, and **REVDIR** can help.

HOW REVDIR WORKS

REVDIR physically reverses the shape from left to right, creating a backwards version of your shape using the following formula: **(Leftmost point+Rightmost point)-Present point= New location of point.**

First it adds the leftmost and rightmost points in your shape. Then it steps through the shape, subtracting every X-coordinate from this value to establish a new X-coordinate.

RESTRICTIONS ON THE USE OF REVDIR

In order to use REVDIR we need to return to the topic of defining exactly **where** in your shape the routine will look to find the needed points. Searching for the information would take too long, so we have established **RULE #5**.

If you're going to use REVDIR to reverse your shape then you **MUST** define your shape in the following manner (otherwise it will simply destroy your shape):

5A. The **FIRST line** in your shape table must extend from the rightmost coordinate to the leftmost coordinate of the shape. This line may be horizontal or slanted.

OR...

5B. The **FIRST X coordinates in each** of the first two lines must be the extreme endpoints of the shape. In other words, if the **first X coordinate** in the table (the **first line** being drawn) is on the **leftmost** side, then the **first X coordinate** (of the **second line**) must be on the **rightmost** side. These lines may be horizontal, vertical, or diagonal. Bear in mind that in a SOLID shape these will be the first points of the **first** and **second** lines. However with an OPEN shape these will actually be the starting points of the **first** and **third** lines. At any rate they must be the **FIRST** and **THIRD X coordinates** in the table.

Before using the REVDIR routine you must tell the routine how you defined your shape so it'll know where to get the endpoints.

If you defined you shape according to **option A**, enter **POKE 252,0**.

If you defined your shape according to **option B**, enter **POKE 252,1.**

PUT IT ALL TOGETHER

By now you should be familiar with just about all of the routines in the driver. While we haven't directly looked at the routines for vertical movement, they work the same way as the horizontal movement routines, **except** that you **POKE** your increment into memory location **253** instead of **207**.

To get an overall view of the driver abilities, let's try to create a routine that directly or indirectly makes use of every driver routine. This will also help detect any bugs or typos in your driver.

Let's make a program that does the following: **First**, we'll manipulate **3 shapes** on the screen. The first shape will be our shapeship (**shape #147**) which we'll move back and forth across the screen. Next we'll make a simple **arrow** that will always travel the opposite direction from our spaceship, and which we'll **REVERSE** at each side of the screen so that it's always pointed in the proper direction. We'll call the arrow **shape #146**. Finally, create a simple **rectangle** which will travel up and down on the screen. This way **we'll always have 3 shapes moving, all in different directions**. The **rectangle** will be called **shape #145**. In addition we'll define our **shapes #145** and **#146** as **OPEN shapes**, which means that we'll also need to keep track of what type of shape we're dealing with. Since our two new shapes will be rather simple, we'll create them within our program.

THE THREE SHAPE PROGRAM

Once you've typed the program **Listing 5** into memory, you'll need to **load the driver and shape #147**, which should be at **X-coordinates 4-23**.Then simply **RUN** the program.

The program is heavily REMed so you should be able to follow it. You'll note that we've used the secondary entry points for **shapes #145** and **#146**. Basically what we're doing is POKEing the **shape#**, setting our

LISTING 1

```
10  HGR : GOSUB 1900:INCR = 10
15  IF BASH(4) >  = 279 THEN  END
20  HCOLOR= 0: GOSUB 1001: GOSUB 1210: HCOLOR= 3: GOSUB
    1001: GOTO 15
1000  REM  *** DRAW ***
1001 BASL = 0: REM ..........SET POINTER TO START OF T
    ABLE
1010 CTR = BASH(BASL): REM ..GET # OF POINTS
1015 BASL = BASL + 1: REM .....POINT TO X COORDINATE
1020 X = BASH(BASL): REM ....GET X COORDINATE
1045 BASL = BASL + 2: REM .....POINT TO Y COORDINATE
1050 Y = BASH(BASL): REM ....GET Y COORDINATE
1060  HPLOT X,Y: REM ........HPLOT STARTING POINT
1065 CTR = CTR - 1: REM  .......NOTE THAT A POINT WAS
    USED
1075 BASL = BASL + 1: REM .....POINT TO NEXT X COORDIN
    ATE
1080 X = BASH(BASL): REM ....GET X COORDINATE
1105 BASL = BASL + 2: REM .....POINT TO Y COORDINATE
1110 Y = BASH(BASL): REM ....GET Y COORDINATE
1130  HPLOT  TO X,Y: REM ......DRAW A LINE
1135 CTR = CTR - 1: REM .......ANOTHER POINT USED
1140  IF CTR <  > 0 THEN 1015
1145  RETURN
1210 Y = 0: REM ..............SET POINTER TO START
1220 CTR = BASH(Y): REM .....GET # OF POINTS
1225 Y = Y + 1: REM ...........POINT TO X COORDINATE
1240 BASH(Y) = BASH(Y) + INCR: REM  ADD INCREMENT
1270 Y = Y + 2: REM ...........POINT TO Y COORDINATE
1275 CTR = CTR - 1: REM .......A POINTS BEEN USED
1280  IF CTR <  > 0 THEN 1225
1285  RETURN
1900  DIM BASH(96): FOR X = 0 TO 96: READ BASH(X): NEXT
    X: RETURN
2000  DATA  32,10,0,14,29,0,14,16,0,11,23,0,11,14,0,1
    2,25,0,12,12,0,13,27,0,13,18,0,10,21,0,10,10,0,15
    ,12,0,15,15,0,15,16,0,15
2010  DATA  19,0,15,20,0,15,23,0,15,24,0,15,27,0,15,29
    ,0,15,11,0,16,28,0,16,12,0,17,27,0,17,13,0,18,26,
    0,18,14,0,19,25,0,19,16,0,20,23,0,20,19,0,21,20,0
    ,21
```

LISTING 2

```
90 SHAPE = 147:INCR = 4: HGR
100  POKE 251,SHAPE: POKE 207,INCR: REM  SET SHAPE# AN
    D INCREMENT
110  CALL 38095: REM  ERASE
120  IF  PEEK (37636) + PEEK (37637) * 256 >  = (279 -
    INCR) THEN 180: REM   TEST RIGHT EDGE OF SCREEN
130  CALL 38167: REM  MOVE RIGHT 1 INCREMENT
140  CALL 38102: REM  DRAW
150  GOTO 110: REM  KEEP MOVING RIGHT
160  CALL 38095: REM  ERASE
170  IF  PEEK (37633) + PEEK (37634) * 256 <  = (0 +
    INCR) THEN 130: REM   TEST LEFT EDGE OF SCREEN
180  CALL 38196: REM  MOVE LEFT 1 INCREMENT
190  CALL 38102: REM  DRAW
200  GOTO 160: REM  KEEP MOVING LEFT
```

LISTING 3

```
90 SHAPE = 147:INCR = 4
100  HGR : HPLOT 50,50 TO 50,60: HGR2 : POKE 251,SHAPE
    : POKE 207,INCR: REM  CLEAR BOTH PAGES-SET UP SHA
    PE# AND INCREMENT
110  IF  PEEK (37636) + PEEK (37637) * 256 >  = (279 -
    INCR) THEN 150: REM    TEST FOR RIGHT EDGE OF SCR
    EEN
120  CALL 38312: REM  CALL FLIPR1 ROUTINE
130  GOTO 110: REM  KEEP MOVING RIGHT
140  IF  PEEK (37633) + PEEK (37634) * 256 <  = (0 +
    INCR) THEN 120: REM    TEST FOR LEFT EDGE OF SCRE
    EN
150  CALL 38356: REM  CALL FLIPL1
160  GOTO 140: REM  KEEP MOVING LEFT
```

POKE 381165 to specify shape **type**, and then **CALLing** a **FLIP** routine. When we get to the end of a loop we need to take a few extra steps to **ERASE** our arrow on **BOTH** pages, without doing a FLIP before we REVERSE it (in readiness for the return trip).

If you **stop** the program, you'll need to **reload shape #147** before restarting it so that everything works properly.

At this point you should know about all there is to know about HPLOT shapes, and have a flexible tool to use in manipulating these shapes. That certainly doesn't mean to say that you couldn't add your own enhancements (some FLIP routines for diagonal movement might be handy), or that things couldn't be done differently. But you should have a solid grasp on how to create and animate shapes.

WHAT HAPPENS WHEN IT DOESN'T WORK?

There are several ways that can do you in when working with the driver, many of which I found out about first hand.

First: The easiest mistake to make is trying to use CHR$(4) to load shape tables, the driver itself, or whatever, without first setting **HIMEM**. If you don't **set HIMEM below your lowest shape** you'll damage the driver, starting with the FLIPL2 routine. If you make this error, and try to use FLIPL2, you'll drop through into DOS, possibly making it impossible to recover your program.

The **next** thing that can cause lots of headaches is forgetting what mode you presently have the driver in. As you've found out, there are several POKEs that will modify its function. For instance, if you try to go back and rerun some of our tests using **shape #147**, after having run our final test using 3 shapes, there's good chance (2 to 1) that your shape will be distorted. When we ran our first tests, I knew that the driver was set for SOLID shapes (which shape #147 needs), but in our final test (with 2 of the 3 shapes) we set the driver for OPEN shapes. So if you don't reset **POKE 38165,204** before rerunning earlier tests, things may not work right.

Another possible pitfall arises with the **FLIP** and **FLP** routines. If for instance you went back and tried to rerun our one-shape page-flip test after having run the multiple shape tests, you wouldn't see very much happening. When we did the one-shape test I knew the **NOPs** were in place in the flip routines, but when we went to multiple shapes we replaced the **NOPs** with **RTSs**. Going back to the 1 page test, without replacing the NOPs would do **ALL** the drawing on the **hidden** screen, (page 2) and we'd never see anything. The point is simply this, **if something doesn't work, check the status of the driver to see if it's in the proper mode.**

Probably the best way to do this is to **PEEK** the locations that we normally **POKE** to find how they're set.

Finally, if you're keeping your shapes on the screen with **FOR...NEXT** loops, rather than actually testing for the edges of the screen, be sure that your shapes are in the proper starting position when you start a program. You can often get away with letting your shape go a few points off the right edge of the screen (coordinates **greater than 279**); however any attempt to go past the left edge (coordinates **less than 0**) will spell instant disaster, destroying your shape, and requiring a RESET to recover.

By the way, if anyone out there in civilization comes up with any other useful improvements to the driver, I'd appreciate hearing from you. In our next discussion we'll start to look at BLOCK shapes, a method of shape creation that is most often used in those high speed animation games that we're all so fond of. See you then!!

LISTING 4

```
199  REM  MODIFY FLIP ROUTINES FOR MULTIPLE SHAPE/SET
     INCREMENT
200  POKE 38333,96: POKE 38377,96: POKE 207,4
205  HGR : HGR2 : REM  CLEAR THE HI-RES SCREENS
210  FOR X = 1 TO 30
215  REM  DISPLAY PAGE 1-DRAW PAGE 2
220  POKE 251,147: CALL 38312: REM MOVE SHAPE #147 -->

240  POKE 251,146: CALL 38356: REM MOVE SHAPE #146 <--

255  REM  DISPLAY PAGE 2-DRAW PAGE 1
260  POKE 251,147: CALL 38334: REM MOVE SHAPE #147 -->

280  POKE 251,146: CALL 38378: REM MOVE SHAPE #146 <--

300  NEXT
310  REM  GO BACK THE OTHER DIRECTION
320  FOR X = 1 TO 30
325  REM DISPLAY PAGE 1-DRAW PAGE 2
330  POKE 251,147: CALL 38356: REM MOVE SHAPE #147 <--

350  POKE 251,146: CALL 38312: REM MOVE SHAPE #146 -->

360  REM  DISPLAY PAGE 2-DRAW PAGE 1
370  POKE 251,147: CALL 38378: REM MOVE SHAPE #147 <--

390  POKE 251,146: CALL 38334: REM MOVE SHAPE #146 -->

400  NEXT : GOTO 210
```

LISTING 5

```
50  REM   REQUIRES: M/L DRIVER AND SHAPES #146 AND 147
      TO BE BLOADED INTO MEMORY TO RUN.
99  REM  POKE SHAPE #146 INTO MEMORY
100  FOR X = 37376 TO 37391: READ A: POKE X,A: NEXT : POKE
     207,20: POKE 251,146: CALL 38167: REM  POKE SHAPE
      #146 IN MEMORY/MOVE IT RIGHT
105  DATA  5,252,0,50,233,0,50,240,0,55,233,0,50,240,0
     ,45
110  POKE 252,0: REM  TELL REVDIR WE USED RULE #5-OPTI
     ON A
120  FOR X = 37120 TO 37135: READ A: POKE X,A: NEXT : REM
      POKE SHAPE #145 IN MEMORY
130  DATA  5,133,0,5,147,0,5,147,0,20,133,0,20,133,0,5

199  REM  MODIFY FLIP ROUTINES FOR MULTIPLE SHAPES/SET
      INCREMENT
200  POKE 38333,96: POKE 38377,96: POKE 207,4
202  POKE 37906,96: POKE 37956,96: POKE 253,2
205  HGR : HGR2 : REM  CLEAR THE HI-RES SCREENS
210  FOR X = 1 TO 30
215  REM  DISPLAY PAGE 1-DRAW PAGE 2
220  POKE 251,147: POKE 38165,204: CALL 38312: REM  MO
     VE SHAPE #147 -->
240  POKE 251,146: POKE 38165,226: CALL 38365: REM  MO
     VE SHAPE #146 <--
245  POKE 251,145: CALL 37891: REM  MOVE SHAPE #145 DO
     WN
255  REM  DISPLAY PAGE 2-DRAW PAGE 1
260  POKE 251,147: POKE 38165,204: CALL 38334: REM  MO
     VE SHAPE #147 -->
270  POKE 251,145: POKE 38165,226: CALL 37916: REM  MO
     VE SHAPE #145 DOWN
280  POKE 251,146: CALL 38387: REM  MOVE SHAPE #146 <-
     -
285  NEXT
290  POKE 230,64: CALL 38167: CALL 38095: REM   ERASE
     ARROW ON PAGE 2
295  POKE 230,32: CALL 38196: CALL 38095: REM   ERASE
     ARROW ON PAGE 1
300  CALL 37982: CALL 38102: REM  REVERSE AND REDRAW A
     RROW
310  REM  GO BACK THE OTHER DIRECTION
320  FOR X = 1 TO 30
325  REM  DISPLAY PAGE 1-DISPLAY PAGE 2
330  POKE 251,147: POKE 38165,204: CALL 38356: REM  MO
     VE SHAPE #147 <--
350  POKE 251,146: POKE 38165,226: CALL 38321: REM  MO
     VE SHAPE #146 -->
355  POKE 251,145: CALL 37941: REM  MOVE SHAPE #145 UP
     ^
360  REM  DISPLAY PAGE 2-DRAW PAGE 1
370  POKE 251,147: POKE 38165,204: CALL 38378: REM  MO
     VE SHAPE #147 <--
380  POKE 251,145: POKE 38165,226: CALL 37966: REM  MO
     VE SHAPE #145 UP ^
390  POKE 251,146: CALL 38343: REM   MOVE SHAPE #146 -
     ->
400  NEXT
410  POKE 230,64: CALL 38196: CALL 38095: REM   ERASE
     ARROW ON PAGE 2
415  POKE 230,32: CALL 38167: CALL 38095: REM   ERASE
     ARROW ON PAGE 1
420  CALL 37982: CALL 38102: REM  REVERSE AND REDRAW A
     RROW
430  GOTO 210
```

SHAPE #146

```
9200- 20 FD 00 22 10 01 22 03
9208- 01 1F 0A 01 1F 01 01 20
9210- 0C 01 20 FF 00 21 0E 01
9218- 21 05 01 1E 08 01 1E FD
9220- 00 23 FF 00 23 02 01 23
9228- 03 01 23 06 01 23 07 01
9230- 23 0A 01 23 0B 01 23 0E
9238- 01 23 10 01 23 FE 00 24
9240- 0F 01 24 FF 00 25 0E 01
9248- 25 00 01 26 0D 01 26 01
9250- 01 27 0C 01 27 03 01 28
9258- 0A 01 28 06 01 29 07 01
9260- 29
```

SHAPE #147

```
9300- 20 0A 00 0E 1D 00 0E 10
9308- 00 0B 17 00 0B 0E 00 0C
9310- 19 00 0C 0C 00 0D 1B 00
9318- 0D 12 00 0A 15 00 0A 0A
9320- 00 0F 0C 00 0F 0F 00 0F
9328- 10 00 0F 13 00 0F 14 00
9330- 0F 17 00 0F 18 00 0F 1B
9338- 00 0F 1D 00 0F 0B 00 10
9340- 1C 00 10 0C 00 11 1B 00
9348- 11 0D 00 12 1A 00 12 0E
9350- 00 13 19 00 13 10 00 14
9358- 17 00 14 13 00 15 14 00
9360- 15
```

```
:ASM

                        0001 *
                        0002 *  HPLOT DRIVER
                        0003 *  BY ROBERT DEVINE
                        0004 *
                        0005 *  COPYRIGHT 1983 BY MICROSPARC, INC.
                        0006 *
                        0007 *  SC ASSEMBLER
                        0008 *
                        0009  .OR $93FA
                        0010  .TA $800        ** FOR ASSEMBLY ONLY
F457-                   0015 HPLOT .EQ $F457
F53A-                   0020 HLIN .EQ $F53A
0009-                   0025 TEMPA .EQ $9     ** DECIMAL 9
0008-                   0030 TEMPX .EQ $8     ** DECIMAL 8
00FA-                   0035 BASL .EQ $FA     ** DECIMAL 250
00FB-                   0040 BASH .EQ $FB     ** DECIMAL 251 - SHAPE PAGE #
0006-                   0045 CTR .EQ $6       ** DECIMAL 6
00CF-                   0050 INCR .EQ $CF     ** DECIMAL 207
00FC-                   0060 TEST .EQ $FC     ** DECIMAL 252
00FD-                   0070 YINCR .EQ $FD    ** DECIMAL 253
93FA- A9 00             0300 FLPDN1 LDA #0    ** CALL 37882 TO ENTER
93FC- 8D 54 C0          0305  STA $C054       ** DISPLAY PAGE 1
93FF- A9 40             0310  LDA #$40        ** MOVE SHAPE DOWN
9401- 85 E6             0315  STA $E6         ** DRAW PAGE 2
9403- 20 B8 94          0320  JSR GOUP        ** GO BACK 1 STEP
9406- 20 CF 94          0325  JSR ERASE       ** ERASE SHAPE
9409- 20 A1 94          0330  JSR GODOWN      ** GO BACK TO START
940C- 20 A1 94          0335  JSR GODOWN      ** GO AHEAD 1 STEP
940F- 20 D6 94          0340  JSR DRAW        ** DRAW SHAPE
9412- EA                0345  NOP             ** RTS OR NOP HERE
9413- A9 00             0350 FLPDN2 LDA #0    ** CALL 37907 TO ENTER
9415- 8D 55 C0          0355  STA $C055       ** DISPLAY PAGE 2
9418- A9 20             0360  LDA #$20        **
941A- 85 E6             0365  STA $E6         ** DRAW PAGE 1
941C- 20 B8 94          0370  JSR GOUP        ** GO BACK 1 STEP
941F- 20 CF 94          0375  JSR ERASE       ** ERASE SHAPE
9422- 20 A1 94          0380  JSR GODOWN      ** GO BACK TO START
9425- 20 A1 94          0385  JSR GODOWN      ** GO AHEAD 1 STEP
9428- 20 D6 94          0390  JSR DRAW        ** DRAW SHAPE
942B- 60                0395  RTS
942C- A9 00             0400 FLPUP1 LDA #0    ** CALL 37932 TO ENTER
942E- 8D 54 C0          0405  STA $C054       ** DISPLAY PAGE 1
9431- A9 40             0410  LDA #$40        ** MOVE SHAPE UP ^
9433- 85 E6             0415  STA $E6         ** DRAW PAGE 2
9435- 20 A1 94          0420  JSR GODOWN      ** GO BACK 1 STEP
9438- 20 CF 94          0425  JSR ERASE       ** ERASE SHAPE
943B- 20 B8 94          0430  JSR GOUP        ** GO BACK TO START
943E- 20 B8 94          0435  JSR GOUP        ** GO AHEAD 1 STEP
9441- 20 D6 94          0440  JSR DRAW        ** DRAW SHAPE
9444- EA                0445  NOP             ** RTS OR NOP HERE
9445- A9 00             0450 FLPUP2 LDA #0    ** CALL 37957 TO ENTER
9447- 8D 55 C0          0455  STA $C055       ** DISPLAY PAGE 2
944A- A9 20             0460  LDA #$20        **
944C- 85 E6             0465  STA $E6         ** DRAW PAGE 1
944E- 20 A1 94          0470  JSR GODOWN      ** GO BACK 1 STEP
9451- 20 CF 94          0475  JSR ERASE       ** ERASE SHAPE
9454- 20 B8 94          0480  JSR GOUP        ** GO BACK TO START
9457- 20 B8 94          0485  JSR GOUP        ** GO AHEAD 1 STEP
945A- 20 D6 94          0490  JSR DRAW        ** DRAW SHAPE
945D- 60                0495  RTS
945E- A0 00             0500 REVDIR LDY #0    ** CALL 37982 TO ENTER
9460- 84 FA             0505  STY BASL        ** POINT TO START OF MEMORY PAGE
9462- C8                0510  INY             ** POINT TO X HI BYTE
9463- B1 FA             0515  LDA (BASL),Y    ** GET X HI BYTE
9465- 85 09             0520  STA TEMPA       ** STORE HI BYTE
9467- C8                0525  INY             ** POINT TO X LO BYTE
```

```
9468- B1 FA     0530  LDA (BASL),Y    ** GET X LO BYTE
946A- 85 08     0535  STA TEMPX       ** STORE LO BYTE
946C- C8        0540  INY             ** POINT TO Y BYTE
946D- C8        0545  INY             ** NEXT X HI BYTE
946E- A6 FC     0550  LDX TEST        ** GET RULE #5 FLAG
9470- CA        0555  DEX             ** CONDITION ZERO FLAG
9471- D0 03     0560  BNE T1          ** IF BOTH IN FIRST LINE-JUMP
9473- C8        0565  INY             ** MOVE TO FIRST
9474- C8        0570  INY             ** X COORDINATE OF
9475- C8        0575  INY             ** THE NEXT LINE
9476- 18        0580 T1 CLC
9477- B1 FA     0585  LDA (BASL),Y    ** GET X HI BYTE
9479- 65 09     0590  ADC TEMPA       ** ADD TO FIRST X HI BYTE
947B- 85 09     0595  STA TEMPA       ** STORE RESULT
947D- C8        0600  INY             ** POINT TO X LO BYTE
947E- B1 FA     0605  LDA (BASL),Y    ** GET X LO BYTE
9480- 65 08     0610  ADC TEMPX       ** ADD TO FIRST X LO BYTE
9482- 85 08     0615  STA TEMPX       ** STORE RESULT
9484- A0 00     0620  LDY #0          ** RESET POINTER
9486- 84 FA     0625  STY BASL        ** RESET TO START OF MEMORY PAGE
9488- B1 FA     0630  LDA (BASL),Y    ** GET # OF POINTS
948A- 85 06     0635  STA CTR         ** PUT IN COUNTER
948C- C8        0640 ST7 INY          ** POINT TO X HI BYTE
948D- 38        0645  SEC
948E- A5 09     0650  LDA TEMPA       ** GET HI VALUE
9490- F1 FA     0655  SBC (BASL),Y    ** SUBTRACT PRESENT HI BYTE
9492- 91 FA     0660  STA (BASL),Y    ** PUT NEW HI BYTE IN TABLE
9494- C8        0665  INY             ** POINT TO X LO BYTE
9495- A5 08     0670  LDA TEMPX       ** GET LO VALUE
9497- F1 FA     0675  SBC (BASL),Y    ** SUBTRACT PRESENT LO BYTE
9499- 91 FA     0680  STA (BASL),Y    ** PUT NEW LO BYTE IN TABLE
949B- C8        0685  INY             ** POINT TO Y BYTE
949C- C6 06     0690  DEC CTR         ** A POINTS BEEN CHANGED
949E- D0 EC     0695  BNE ST7         ** IF MORE-CONTINUE
94A0- 60        0696  RTS             ** DONE-EXIT ROUTINE
94A1- A0 00     0700 GODOWN LDY#0     ** CALL 38049 TO ENTER
94A3- 84 FA     0705  STY BASL        ** POINT TO START OF MEMORY PAGE
94A5- B1 FA     0710  LDA (BASL),Y    ** GET # OF POINTS
94A7- 85 06     0715  STA CTR         ** PUT IN COUNTER
94A9- C8        0720 ST6 INY          ** POINT TO X HI-BYTE
94AA- C8        0725  INY             ** POINT TO X LO-BYTE
94AB- C8        0730  INY             ** POINT TO Y-COORDINATE
94AC- 18        0735  CLC
94AD- B1 FA     0740  LDA (BASL),Y    ** GET Y-COORDINATE
94AF- 65 FD     0745  ADC YINCR       ** ADD INCREMENT
94B1- 91 FA     0750  STA (BASL),Y    ** PUT IN TABLE
94B3- C6 06     0755  DEC CTR         ** DECREMENT COUNTER
94B5- D0 F2     0760  BNE ST6         ** IF MORE-CONTINUE
94B7- 60        0765  RTS             ** DONE-EXIT ROUTINE
94B8- A0 00     0800 GOUP LDY #0      ** CALL 38072 TO ENTER
94BA- 84 FA     0805  STY BASL        ** POINT TO START OF MEMORY PAGE
94BC- B1 FA     0810  LDA (BASL),Y    ** GET # OF POINTS
94BE- 85 06     0815  STA CTR         ** PUT IN COUNTER
94C0- C8        0820 ST5 INY          ** POINT TO X HI-BYTE
94C1- C8        0825  INY             ** POINT TO X LO-BYTE
94C2- C8        0830  INY             ** POINT TO Y-COORDINATE
94C3- 38        0835  SEC
94C4- B1 FA     0840  LDA (BASL),Y    ** GET Y-COORDINATE
94C6- E5 FD     0845  SBC YINCR       ** SUBTRACT INCREMENT
94C8- 91 FA     0850  STA (BASL),Y    ** PUT IN TABLE
94CA- C6 06     0855  DEC CTR         ** DECREMENT COUNTER
94CC- D0 F2     0860  BNE ST5         ** IF MORE-CONTINUE
94CE- 60        0865  RTS             ** DONE-EXIT ROUTINE
94CF- A9 00     0970 ERASE LDA #0     ** CALL 38095 TO ENTER
94D1- 85 E4     0975  STA $E4         ** SET COLOR=BLACK
94D3- 4C DA 94  0980  JMP START
94D6- A9 7F     0985 DRAW LDA #$7F    ** CALL 38102 TO ENTER
94D8- 85 E4     0990  STA $E4         ** SET COLOR=WHITE
94DA- A2 00     1000 START LDX #0     ** SET POINTER=0
94DC- 86 FA     1001  STX BASL        ** SET BASL=0
94DE- A1 FA     1005  LDA (BASL,X)    ** GET # OF POINTS
94E0- 85 06     1010  STA CTR         ** PUT IN COUNTER
94E2- E6 FA     1015 ST INC BASL
94E4- A1 FA     1020  LDA (BASL,X)    ** GET X LO BYTE
94E6- 85 08     1025  STA TEMPX       ** STORE IN HOLDER
94E8- E6 FA     1030  INC BASL
94EA- A1 FA     1035  LDA (BASL,X)    ** GET X HI BYTE
94EC- A8        1040  TAY             ** STORE IN Y-REGISTER
94ED- E6 FA     1045  INC BASL
94EF- A1 FA     1050  LDA (BASL,X)    ** PUT Y COORD IN ACCUM
94F1- A6 08     1055  LDX TEMPX       ** PUT LO BYTE IN X-REGISTER
94F3- 20 57 F4  1060  JSR HPLOT       ** HPLOT THE POINT
94F6- C6 06     1065  DEC CTR         ** A POINTS BEEN USED
94F8- A2 00     1070  LDX #0          ** RESET POINTER
94FA- E6 FA     1075  INC BASL
94FC- A1 FA     1080  LDA (BASL,X)    ** GET X LO BYTE
94FE- 85 09     1085  STA TEMPA       ** STORE IN HOLDER
9500- E6 FA     1090  INC BASL
9502- A1 FA     1095  LDA (BASL,X)    ** GET X HI BYTE
9504- 85 08     1100  STA TEMPX       ** STORE IN HOLDER
9506- E6 FA     1105  INC BASL
9508- A1 FA     1110  LDA (BASL,X)    ** GET Y COORD
```

```
950A- AB       1115  TAY              ** PUT IN Y-REGISTER
950B- A6 08    1120  LDX TEMPX        ** MOVE HI BYTE TO X-REGISTER
950D- A5 09    1125  LDA TEMPA        ** MOVE LO BYTE TO ACCUM
950F- 20 3A F5 1130  JSR HLIN         ** DRAW A LINE
9512- C6 06    1135  DEC CTR          ** A POINTS BEEN USED
9514- D0 CC    1140  BNE ST           ** POKE 38165,204=SOLID/226=OPEN
9516- 60       1145  RTS              ** DONE--EXIT ROUTINE
9517- A0 00    1146 MOVER1 LDY #0     ** CALL 38167 TO ENTER
9519- 84 FA    1210  STY BASL         ** SET BASL=0
951B- B1 FA    1215  LDA (BASL),Y     ** GET # OF POINTS
951D- 85 06    1220  STA CTR          ** PUT IN COUNTER
951F- C8       1225 ST1 INY           ** INCREMENT POINTER
9520- 18       1230  CLC
9521- B1 FA    1235  LDA (BASL),Y     ** GET X HI BYTE
9523- 65 CF    1240  ADC INCR         ** ADD INCREMENT
9525- 91 FA    1245  STA (BASL),Y     ** PUT IN TABLE
9527- C8       1250  INY              ** INCREMENT POINTER
9528- B1 FA    1255  LDA (BASL),Y     ** GET X LO BYTE
952A- 69 00    1260  ADC #0           ** ADD THE CARRY FLAG
952C- 91 FA    1265  STA (BASL),Y     ** PUT IN TABLE
952E- C8       1270  INY              ** POINT TO Y COORD
952F- C6 06    1275  DEC CTR          ** A POINTS BEEN USED
9531- D0 EC    1280  BNE ST1          ** IF MORE-CONTINUE
9533- 60       1285  RTS              ** DONE-EXIT ROUTINE
9534- A0 00    1300 MOVEL1 LDY #0     ** CALL 38197 TO ENTER
9536- 84 FA    1310  STY BASL         ** SET BASL=0
9538- B1 FA    1315  LDA (BASL),Y     ** GET # OF POINTS
953A- 85 06    1320  STA CTR          ** PUT IN COUNTER
953C- C8       1325 ST2 INY           ** INCREMENT POINTER
953D- 38       1330  SEC
953E- B1 FA    1335  LDA (BASL),Y     ** GET X HI BYTE
9540- E5 CF    1340  SBC INCR         ** SUBTRACT INCREMENT
9542- 91 FA    1345  STA (BASL),Y     ** STORE IN TABLE
9544- C8       1350  INY              ** INCREMENT POINTER
9545- B1 FA    1355  LDA (BASL),Y     ** GET X LO BYTE
9547- E9 00    1360  SBC #0           ** SUBTRACT ANY BORROW
9549- 91 FA    1365  STA (BASL),Y     ** STORE IN TABLE
954B- C8       1370  INY              ** INCREMENT POINTER
954C- C6 06    1375  DEC CTR          ** A POINTS BEEN USED
954E- D0 EC    1380  BNE ST2          ** IF MORE-CONTINUE
9550- 60       1385  RTS              ** DONE-EXIT ROUTINE
9551- A0 00    1400 MOVEL2 LDY #0     ** CALL 38225 TO ENTER
9553- 84 FA    1410  STY BASL         ** SET BASL=0
9555- B1 FA    1415  LDA (BASL),Y     ** GET # OF POINTS
9557- 85 06    1420  STA CTR          ** PUT IN COUNTER
9559- C8       1425 ST3 INY           ** INCREMENT POINTER
955A- 38       1430  SEC
955B- B1 FA    1432  LDA (BASL),Y     ** GET X HI BYTE
955D- E5 CF    1434  SBC INCR         ** SUBTRACT INCREMENT
955F- 91 FA    1436  STA (BASL),Y     ** PUT IN TABLE
9561- C8       1438  INY              ** INCREMENT POINTER
9562- B1 FA    1440  LDA (BASL),Y     ** GET X LO BYTE
9564- E9 00    1442  SBC #0           ** SUBTRACT ANY BORROW
9566- 91 FA    1444  STA (BASL),Y     ** PUT IN TABLE
9568- 88       1446  DEY              ** BACK TO HI BYTE
9569- B1 FA    1448  LDA (BASL),Y     ** GET X HI BYTE
956B- E5 CF    1450  SBC INCR         ** SUBTRACT INCREMENT
956D- 91 FA    1452  STA (BASL),Y     ** PUT IN TABLE
956F- C8       1454  INY              ** BACK TO LO BYTE
9570- B1 FA    1456  LDA (BASL),Y     ** GET X LO BYTE
9572- E9 00    1458  SBC #0           ** SUBTRACT ANY BORROW
9574- 91 FA    1460  STA (BASL),Y     ** PUT IN TABLE
9576- C8       1470  INY              ** POINT TO Y COORD
9577- C6 06    1475  DEC CTR          ** A POINTS BEEN USED
9579- D0 DE    1480  BNE ST3          ** IF MORE CONTINUE
957B- 60       1485  RTS              ** DONE-EXIT ROUTINE
957C- A0 00    1500 MOVER2 LDY #0     ** CALL 38268 TO ENTER
957E- 84 FA    1510  STY BASL         ** SET BASL=0
9580- B1 FA    1515  LDA (BASL),Y     ** GET # OF POINTS
9582- 85 06    1520  STA CTR          ** PUT IN COUNTER
9584- C8       1525 ST4 INY           ** INCREMENT POINTER
9585- 18       1530  CLC
9586- B1 FA    1532  LDA (BASL),Y     ** GET X HI BYTE
9588- 65 CF    1534  ADC INCR         ** ADD INCREMENT
958A- 91 FA    1536  STA (BASL),Y     ** PUT IN TABLE
958C- C8       1538  INY              ** INCREMENT POINTER
958D- B1 FA    1540  LDA (BASL),Y     ** GET X LO BYTE
958F- 69 00    1542  ADC #0           ** ADD THE CARRY FLAG
9591- 91 FA    1544  STA (BASL),Y     ** PUT IN TABLE
9593- 88       1546  DEY              ** BACK TO HI BYTE
9594- 18       1548  CLC
9595- B1 FA    1550  LDA (BASL),Y     ** GET X HI BYTE
9597- 65 CF    1552  ADC INCR         ** ADD INCREMENT
9599- 91 FA    1554  STA (BASL),Y     ** PUT IN TABLE
959B- C8       1556  INY              ** BACK TO LO BYTE
959C- B1 FA    1558  LDA (BASL),Y     ** GET X LO BYTE
959E- 69 00    1560  ADC #0           ** ADD THE CARRY FLAG
95A0- 91 FA    1562  STA (BASL),Y     ** PUT IN TABLE
95A2- C8       1570  INY              ** POINT TO Y COORD
```

```
95A3- C6 06     1575  DEC CTR          ** A POINTS BEEN USED
95A5- D0 DD     1580  BNE ST4          ** IF MORE-CONTINUE
95A7- 60        1585  RTS              ** DONE-EXIT ROUTINE
95A8- A9 00     1600 FLIPR1 LDA #0     ** CALL 38312 TO ENTER
95AA- 8D 54 C0  1605  STA $C054        ** DISPLAY PAGE 1
95AD- A9 40     1610  LDA #$40        ** MOVE SHAPE RIGHT --->
95AF- 85 E6     1615  STA $E6         ** DRAW PAGE 2
95B1- 20 34 95  1620  JSR MOVEL1      ** GO BACK 1 STEP
95B4- 20 CF 94  1635  JSR ERASE       ** ERASE SHAPE
95B7- 20 7C 95  1640  JSR MOVER2      ** GO FORWARD 2 STEPS
95BA- 20 D6 94  1655  JSR DRAW        ** DRAW SHAPE
95BD- EA        1656  NOP              ** RTS OR NOP HERE-POKE 38333,96 OR 234
95BE- A9 00     1660 FLIPR2 LDA #0     ** CALL 38334 TO ENTER
95C0- 8D 55 C0  1665  STA $C055        ** DISPLAY PAGE 2
95C3- A9 20     1670  LDA #$20        ** MOVE SHAPE RIGHT --->
95C5- 85 E6     1675  STA $E6         ** DRAW PAGE 1
95C7- 20 34 95  1680  JSR MOVEL1      ** GO BACK 1 STEP
95CA- 20 CF 94  1695  JSR ERASE       ** ERASE SHAPE
95CD- 20 7C 95  1700  JSR MOVER2      ** GO FORWARD 2 STEPS
95D0- 20 D6 94  1715  JSR DRAW        ** DRAW SHAPE
95D3- 60        1720  RTS
95D4- A9 00     1800 FLIPL1 LDA #0     ** CALL 38356 TO ENTER
95D6- 8D 54 C0  1805  STA $C054        ** DISPLAY PAGE 1
95D9- A9 40     1810  LDA #$40        ** MOVE SHAPE LEFT <---
95DB- 85 E6     1815  STA $E6         ** DRAW PAGE 2
95DD- 20 17 95  1820  JSR MOVER1      ** GO BACK 1 STEP
95E0- 20 CF 94  1825  JSR ERASE       ** ERASE SHAPE
95E3- 20 51 95  1840  JSR MOVEL2      ** GO FORWARD 2 STEPS
95E6- 20 D6 94  1855  JSR DRAW        ** DRAW SHAPE
95E9- EA        1856  NOP              ** RTS OR NOP HERE-POKE 38377,96 OR 234
95EA- A9 00     1860 FLIPL2 LDA #0     ** CALL 38378 TO ENTER
95EC- 8D 55 C0  1865  STA $C055        ** DISPLAY PAGE 2
95EF- A9 20     1870  LDA #$20        ** MOVE SHAPE LEFT <---
95F1- 85 E6     1875  STA $E6         ** DRAW PAGE 1
95F3- 20 17 95  1880  JSR MOVER1      ** GO BACK 1 STEP
95F6- 20 CF 94  1895  JSR ERASE       ** ERASE SHAPE
95F9- 20 51 95  1900  JSR MOVEL2      ** GO FORWARD 2 STEPS
95FC- 20 D6 94  1915  JSR DRAW        ** DRAW SHAPE
95FF- 60        1920  RTS
```