Hi-Res Cursor Mover

by Doug Denby
137 Main St.
Unieuville, Ontario
Canada. L3R 2G6

When dealing with the high resolution screen, it is often necessary to move a point or cursor about the screen to identify a location or to place an object. When there are paddles attached via the game paddle input socket, then reading the paddle locations via PDL(0) and PDL(1) and translating the resultant numbers into screen coordinates works very well.

Mathematically, this is fairly easily done. The maximum value returned by a PDL(n) function is 255. The vertical dimension of the HGR screen is 160 units while the HGR2 screen is 192; both are much less than the potential value from the paddles. Thus a very simple function such as **Y = PDL(1) * 191 / 255** for the full screen or **Y = PDL(1) * 160 / 255** for HGR screen will generate a suitable Y coordinate.

The horizontal coordinate is somewhat more difficult. Both the HGR and HGR2 screens are **280** pixels wide, while the maximum reading from a paddle function is only **255**. Thus to get an X coordinate, a formula such as X = PDL(0) * 280 / 255 is needed. It is quite obvious that this means that a number of pixels cannot be reached using the paddles to input the coordinates.

There are ways around this, of course. You could always build your screen design such that it was only 255 pixels wide. Then XPDL(0) would give you the horizontal coordinate.

There are two reasons why I would not wish to do this. First I hate limiting the APPLE's capabilities. I bought it because of its capabilities.

Secondly, the paddles are not the most reliable of input devices. It is often difficult to get consecutive numerical readings from them, especially after many hours of use.

The new APPLE computers are being sold without the paddles. This is being done to reduce the amount of RF interference produced by the machines. This means that many owners of APPLEs, now and more so in the future, may not have paddles from which to input. This is another good reason to design programs with a better or alternate method of controlling cursor movements on the high resolution screen.

USING THE KEYBOARD

A way to reach all pixels is to use the keyboard for input. The familiar **'I'**, **'J'**, **'K'**, and **'M'** keys could easily be used to move the cursor about the high resolution screen as well as the text screen. These could be easily augmented by the use of the **'U'**, **'O'**, **'N'** and **','** keys to pick up diagonal movements.

In BASIC a loop to read the keyboard could be created, and every time one of those keys was struck the cursor could be moved in the appropriate direction.

For example:

```
100 IF PEEK(-16384) < 127 THEN 100: REM see if a key has been hit, if not then look again until one is struck
105 K$ = CHR$( PEEK(-16384) - 128): REM capture the key hit
110 POKE -16368,0: REM clear the keyboard strobe
120 X = X + (K$ = "K") + (K$ = "O") + (K$ = ",") - (K$ = "J") - (K$ = "U") - (K$ = "N")
130 Y = Y + (K$ = "M") + (K$ = "N") + (K$ = ",") - (K$ = "I") - (K$ = "U") - (K$ = "O")
140 REM note the use of the BOOLEAN functions to increment and decrement the values of X and Y. The logic used by the APPLE computer states that if an equality is examined and found to be true then it is equal to 1, but if not true then it is equal to 0.
150 IF X < 0 THEN X = 0
152 IF X > 279 THEN X = 279
154 IF Y < 0 THEN Y = 0
156 IF Y > 159 THEN Y = 159
158 REM this keeps the cursor on the HGR screen
160 HPLOT X,Y: REM put the cursor on the screen
```

To do this properly, the former cursor would have to be erased each time the new cursor is HPLOTed. This requires that the program be aware of the COLOR of the background on which the cursor has been placed. There is no way that this can be done easily in BASIC. However, when using shapes in the high resolution mode there is a function **(XDRAW N AT X,Y)** that places the shape (number 'N') at a location on the high resolution screen using coordinates X,Y. This function is used frequently to move shapes around the high resolution screen.

THE SHAPE CURSOR

A shape is stored in a shape table. Many shapes can be stored in one table. Reading the APPLESOFT MANUAL pages 92 to 97 will explain how to create a shape table. Here is a **sample table** that draws simple crosshairs:

```
01 — only one shape in this table
00 — don't use this byte
04 — low byte of shape location: the first and only shape starts four bytes from the beginning of the shape table
00 — high byte of shape location
24 — move up and plot twice
D6 — plot, move down and left
6F — plot, move left, plot move right twice
29 — move right, plot and move right
9F — plot, move left twice and then down
36 — plot, move down, plot, move down
00 — end of shape
```

Now to place this in memory. Pick a fairly safe place that won't be overwritten by the operating system of the computer as it runs your BASIC program. Hexadecimal **300** is usually a safe location. From BASIC type **CALL -151**. The prompt should now be an assembly language. Type **300:01 00 04 00 24 D6 6F 29 9F 36 00** and the **RETURN** key. Be sure to put the spaces in!

Be sure to put the spaces in!

Now type **CTRL-C** and **RETURN**. This should put you back into BASIC. Clear the screen with **HOME**. Type **FOR I = 768 TO 778: ? PEEK(I)",";: NEXT I** followed with a **RETURN**. The computer will print out the above shape table. The display is, however, now in decimal rather than hexadecimal. Using these values **we can store the shape table values in a DATA statement at the end of our program** and use a simple loop to READ the values and POKE them into place each time the program runs.

```
900 FOR I = 768 TO 778: READ J: POKE I,J: NEXT I: REM this puts the shape table into memory
910 POKE 232,0: POKE 233,3: REM this tells BASIC where to find the shape table
920 RETURN : REM this will now be a subroutine
1000 DATA 1, 0, 4, 0, 36, 214, 111, 41, 159, 54, 0: REM this is the shape table in decimal values
```

To use this new information will require keeping track of two sets of coordinates. Let's modify the program by typing in the following lines:

```
10 GOSUB 900: REM build the shape table and tell the computer where it is
20 ROT = 0: SCALE = 1: COLOR = 3: REM set initial values
30 X1 = 100: Y1 = 100: X = X1: Y = Y1: XDRAW 1 AT X1, Y1: REM put the cursor on the screen
160 XDRAW 1 AT X1,Y1: REM wipe out the old cursor
170 XDRAW 1 AT X,Y: REM draw the new cursor
180 X1 = X: Y1 = Y: REM transfer the new cursor coordinates
190 GOTO 100: REM let's move the cursor again
```

ADDING ZOOM

This will work very nicely. However, it moves the cursor one pixel at a time and this is very slow on the high resolution screen. It takes 280 key strikes to move the cursor from one side to another. Suppose we incorporate a scale or zoom factor. This can be done such that hitting a particular key, such as the SPACE BAR, will increase the amount of movement generated by the cursor key strikes.

```
25 Z = 1: GOSUB 800: REM set the zoom
115 IF K$ = " " THEN Z = Z + 1: GOSUB 800: REM if the space bar is hit, change the zoom factor
145 X = X - (X1 - X) * ZZ: Y = Y - (Y1 - Y) * ZZ:
```

This incorporates the zoom factor into the new coordinate values. Subtracting the new BOOLEAN enhanced coordinate values from the old coordinates yields values of -1, 0 or 1 depending on whether the old value is less than, equal to or greater than the new value. (This is true because lines 130 and 140 only allow a change of 1 or 0, due to the

BOOLEAN functions used.) Multiplying this by the zoom factor puts the change (if any) equal to the zoom factor, and subtracting the result from the BOOLEAN enhanced coordinate produces the new coordinate.

```
800 IF Z = 7 THEN Z = 1: REM use a wrap-around for the zoom value, never allowing it to exceed 6
810 ZZ = 2 ^ Z: REM using exponential values allows faster zoom shifting than mere addition. N.B. 2 ^ 6 = 32
820 RETURN: REM end the subroutine
```

Below is a final version of the program updated to include all the modifications that have been made as the program was developed. ALL of the REM statements have been removed at this stage.

```
10 GOSUB 900
20 ROT = 0: SCALE = 1: COLOR = 3
25 Z = 1: GOSUB 800
30 X1 = 100: Y1 = 100: X = X1: Y = Y1: XDRAW 1 AT X1, Y1
100 IF PEEK(-16384) > 127 THEN 100
105 K$ = CHR$( PEEK( -16384) - 128)
110 POKE -16368,0
115 IF K$ = " " THEN Z = Z + 1: GOSUB 800: GOTO 100
120 X = X + (K$ = "L") + (K$ = "O") + (K$ = ",") - (K$ = "J") - (K$ = "U") - (K$ = "N")
130 Y = Y + (K$ = "M") + (K$ = "N") + (K$ = ",") - (K$ = "I") - (K$ = "U") - (K$ = "O")
145 X = X - (X1 - X) * ZZ: Y = Y - (Y1 - Y) * ZZ
150 IF X < 0 THEN X = 0
152 IF X > 279 THEN X = 279
154 IF Y < 0 THEN Y = 0
156 IF Y > 159 THEN Y = 159
160 XDRAW 1 AT X1,Y1
170 XDRAW 1 AT X,Y
180 X1 = X: Y1 = Y
190 GOTO 100
800 IF Z = 7 THEN Z = 1
810 ZZ = 2 ^ Z
820 RETURN
900 FOR I = 768 TO 778: READ J: POKE I,J: NEXT I
910 POKE 232,0: POKE 233,3
920 RETURN
1000 DATA 1, 0, 4, 0, 36, 214, 111, 41, 159, 54, 0
```

STEERING THE CURSOR

The above program assumes that movement of the cursor will only occur upon striking of a key. In some applications the programmer may wish to have the cursor or other shape move around the screen **on its own** and only be guided by the cursor keys. This could be used to steer a shape around a track drawn on the screen.

To accomplish this only a few lines of our program need to be changed. Instead of waiting for a key to be struck before moving the shape, the shape will have to be moved continually in the direction last indicated by the striking of a key.

Delete lines 105, 110 and 115, revise line 100 and add subroutine 700 as follows:

```
100 IF PEEK(-16384) > 127 THEN GOSUB 700: REM if a key is struck then take action; otherwise continue to move the shape in the previous direction
700 IF CHR$(PEEK(-16384) - 128) = " " THEN 800: REM if the zoom key is hit, change the zoom only
710 K$ = CHR$( PEEK( -16384) - 128): POKE -16368, 0: RETURN
```

This changes the value after a cursor key is struck but does not cause a waiting for the key. **K$** retains its value for the calculations to be done in lines **120** and **130**. Note that hitting any key besides the zoom or cursor keys causes the shape to halt where it is.

Implementation of these lines will place the cursor on the screen, and when a cursor key is struck the cursor will begin to move in that direction at a rate determined by the zoom factor. It might be advisable to use the zoom as an addition factor rather than an exponential factor. This is done by changing the subroutine at 800:

```
800 ZZ = ZZ + 1: REM increase the zoom factor by adding one
810 IF ZZ = 7 THEN ZZ = 1: REM if the zoom is above 6 then loop it back to 1
```

Below is a final version of the routine as it appears after altering it to a self moving cursor, having its direction and rate of movement altered at the keyboard. I hope that you find these ideas helpful in your future programming.

```
10 GOSUB 900
20 ROT = 0: SCALE = 1: COLOR = 3
25 Z = 1: GOSUB 800
30 X1 = 100: Y1 = 100: X = X1: Y = Y1: XDRAW 1 AT X1, Y1
100 IF PEEK(-16384) > 127 THEN GOSUB 700
120 X = X + (K$ = "L") + (K$ = "O") + (K$ = ",") - (K$ = "J") - (K$ = "U") - (K$ = "N")
130 Y = Y + (K$ = "M") + (K$ = "N") + (K$ = ",") - (K$ = "I") - (K$ = "U") - (K$ = "O")
145 X = X - (X1 - X) * ZZ: Y = Y - (Y1 - Y) *ZZ
150 IF X < 0 THEN X = 0
152 IF X > 279 THEN X = 279
154 IF Y < 0 THEN Y = 0
156 IF Y > 159 THEN Y = 159
160 XDRAW 1 AT X1,Y1
170 XDRAW 1 AT X,Y
180 X1 = X: Y1 = Y
190 GOTO 100
700 IF CHR$(PEEK( -16384) - 128) = " " THEN Z = Z + 1: GOTO 800
710 K$ = CHR$(PEEK( -16384) - 128): POKE -16368, 0: RETURN
800 ZZ = ZZ + 1
810 IF ZZ = 7 THEN ZZ = 1
820 RETURN
900 FOR I = 768 TO 778: READ J: POKE I,J: NEXT I
910 POKE 232,0: POKE 233,3
920 RETURN
1000 DATA 1, 0, 4, 0, 36, 214, 111, 41, 159, 54, 0
```