GREAT SHAPES

Use this simple technique to implement more than one shape table for your Hi-Res programs.

Imagine writing a program that uses graphic text on the Hi-Res screen. You want to draw other shapes, too — perhaps icons or special shapes to complement your text font. According to conventional programming wisdom, you'd make a huge shape table that contains all the shapes. After it was assembled, it might even be too big to fit into the location you want. Or your shape table might hog valuable memory.

Instead, try a technique I call sequential tabling. Instead of one big shape table, create a shape table that has just text characters, and a second table that contains just icons. All you need to do is to tell the computer where to find the table from which to draw. To do that, just POKE the address of the table you want to use into the shape table address pointers. The shape tables can be accessed immediately from memory rather than being loaded from disk. GREAT .SHAPES (**Listing 1**) demonstrates the sequential tabling technique.

MULTIPLE SHAPE TABLES

Ordinarily, zero-page memory locations 232 and 233 ($E8 and $E9 hex) hold the address of the beginning of a shape table in memory (in low byte, high byte format). Notice that there is no provision for referencing a second or third shape table. Only one may be accessed at a time.

TABLE 1: Program Outline

Lines	Function
10-20	Clear the screen, set the graphic parameters and set the locations of the load address and ProDOS' A-parameter.
30	Defines the Control-D variable for DOS commands.
40-50	Set the load address variable (in hex) for two shape tables, TABLE.1 and TABLE.2.
60	Displays the heading.
70-110	Load both shape tables, with prompts, and read the ProDOS' A-parameter.
90	Read the load address of the last BLOADed file. For each file loaded, a variable is assigned to the address for future POKEing.
100	Compares the address for the ProDOS' A-parameter with the value read before the BLOAD (PA) and if they are different, sets HI(I) and LO(I) to the 'A' address bytes.
130	Clears the text screen, defines Hi-Res page 1 for drawing and CALLs a subroutine to clear it to black before displaying it with an HGR and displaying prompts.
150-170	Prompts for the shape table for display.
180	POKEs the address of the selected shape table into the shape table pointer.
190	Assigns the number of shapes in the table to the variable SHAPES. The number of shapes in a table is found in the first byte of the table, which is the memory address (computed by multiplying the high byte by 256 and adding that product to the low byte).
200	Allows you to scroll through the shape table, shape by shape. For the examples given, only one shape per table is provided.
210	Clears the text screen window and reselects the shape table to display.

Using the sequential tabling technique, there is no practical limit to the number of shape tables you can have in memory. And since the tables used will be smaller, you can tuck them away in any convenient available memory location.

Here's how: For each table you BLOAD into memory, assign the high byte and low byte of the load address to variables. In **Listing 1**, the variables are stored in the arrays LO(*X*) and HI(*X*). When you wish to use a shape from table 1, simply POKE the low byte into location 232, POKE the high byte into 233, and use a DRAW or XDRAW command to draw the shape. To change tables, simply POKE the address of the second table into addresses 232 and 233.

When using this technique, remember these important points:

1. Know the load address of each shape table.
2. POKE the address of the table into the shape table vector immediately prior to drawing from it.
3. Find out how many shapes are in each table by reading the first byte of the table.

After mastering these simple steps, you'll be on the road to greater freedom and flexibility in the use of shapes and shape tables.

. . . there is no practical limit to the number of shape tables you can have in memory.

The two short shape tables provided contain a single shape each. TABLE.1 (**Listing 2**) depicts a turnip and TABLE.2 (**Listing 3**) contains the likeness of a tulip. If you have a few shape tables lying around on disk, you may substitute them for these listings. Simply rename them TABLE.1 and TABLE.2 for the purposes of the demonstration program. If you want to use more than two tables, just change **line 70** to reflect the number of tables and remember to modify **lines 130 and 150** to correspond to the change. As always, it is important to protect binary data from BASIC with appropriate HIMEM and/or LOMEM statements.

ENTERING THE PROGRAMS

To key in the demonstration program, enter the Applesoft program as shown in **Listing 1** and save it to disk with the command:

SAVE GREAT.SHAPES

To key in the shape tables in **Listings 2 and 3**, enter the Monitor with CALL −151 and type in the hex code. Save **Listing 2** to disk with the command:

BSAVE TABLE.1,A$6000,L$32

Save **Listing 3** to disk with the command:

BSAVE TABLE.2,A$1000,L$43

For help with entering *Nibble* listings, see the beginning of the Program Listings Section.

HOW IT WORKS

You may want to study the listing to see how it works. The demonstration program is documented in **Table 1**. Note that the procedure is a little more complicated under ProDOS, since it keeps the file's load address in $BEB9,$BEBA and the load command's A-parameter address in $BE58, $BE59. It is important to check both addresses, since a shape table is often used at a location different from the one where it was created.

Great Shapes listings begin on page 92

Listing 1 for Great Shapes

GREAT.SHAPES

```
1  REM ********************
2  REM *    GREAT.SHAPES    *
3  REM * BY MARK R. CRAVEN  *
4  REM * COPYRIGHT (C) 1987 *
5  REM * BY MICROSPARC, INC *
6  REM * CONCORD, MA  01742 *
7  REM ********************
10  TEXT : HOME
20  HCOLOR= 3: ROT= 0: SCALE= 1:PD =  PEEK (4
    8896) = 76:LA = 43634 + 5191 * PD:AA = 4
    8728
30 D$ =  CHR$ (4)
40 L$(1) = "6000": REM  FIRST LOAD ADDRESS
50 L$(2) = "1000": REM  SECOND LOAD ADDRESS
60  VTAB 1: HTAB 8: PRINT "MULTIPLE SHAPE TAB
    LE DEMO": FOR I = 1 TO 40: PRINT "-";: NEXT

70  FOR I = 1 TO 2
80  VTAB 10: HTAB 1: CALL  - 958: PRINT "LOAD
    ING TABLE "I;:PA =  PEEK (AA) + 256 *  PEEK
    (AA + 1): PRINT : PRINT D$"BLOAD TABLE."
    I;",A$"L$(I): REM LOAD FIRST SHAPE TABLE
    AT 24576, SECOND AT 4096
90 HI(I) =  PEEK (LA + 1):LO(I) =  PEEK (LA):
    REM  BLOAD ADDRESS POINTERS FOR HI AND
    LO BYTES
100  IF PD THEN  IF PA <  > PEEK (AA) + 256 *
    PEEK (AA + 1) THEN HI(I) =  PEEK (AA +
    1):LO(I) =  PEEK (AA): REM  PRODOS ADJUS
    TMENT FOR 'A' PARAMETER
110  GOSUB 220
120  NEXT
130  HOME : POKE 230,32: CALL 62450: HGR : VTAB
    22: PRINT "PRESS '1' FOR TABLE.1--'2' FO
    R TABLE.2"
140  VTAB 23: PRINT "PRESS <ESCAPE> TO EXIT P
    ROGRAM";:
150  VTAB 23: HTAB 32: GET X$: POKE  - 16368,
    0:X =  VAL (X$): ON X GOTO 180,180
160  IF X$ =  CHR$ (27) THEN  TEXT : HOME : END

170  PRINT  CHR$ (7): GOTO 150: REM BEEP AND
    INPUT AGAIN
180  POKE 232,LO(X): POKE 233,HI(X): REM SHAP
    E TABLE LOCATION
190 SHAPES =  PEEK (256 * HI(X) + LO(X)): REM
    NUMBER OF SHAPES IN THE TABLE
200  FOR J = 1 TO SHAPES: DRAW J AT 130,80: HOME
    : VTAB 21: HTAB 1: PRINT "SHAPE # ";J" O
    F TABLE."X: GOSUB 220: XDRAW J AT 130,80
    : NEXT
210  HOME : GOTO 130
220  VTAB 23: PRINT "PRESS <RETURN> TO CONTIN
    UE";: GET Y$: POKE  - 16368,0: RETURN
```

END OF LISTING 1

Listing 2 for Great Shapes

TABLE.1

```
6000- 01 00 04 00 4D 4D 09 8D
6008- DF DF FB 6E 69 4D F1 DF
6010- FB 73 0D 4D 1E FF 0E F5
6018- 17 2D 15 3F 3F 17 2D 2D
6020- 2D 3E 3F 3F 37 2D 2D 2D
6028- 1E 3F 3F 0E 2D 1E 4D 49
6030- 01 00
```

END OF LISTING 2

Listing 3 for Great Shapes

TABLE.2

```
1000- 01 00 04 00 49 49 49 FA
1008- 1B DF BB 2D 0D 2D 0D 2D
1010- 3E FF 3F 1F 3F 2E 6D 2D
1018- 0D 2D 3E 1F 3F 1F 3F 37
1020- 2D 6D 2D 0D 35 1F 3F FF
1028- 3F 37 2D 2D 0D 2D 0D 1E
1030- 3F FF 3F 37 2D 6D 2D F5
1038- FF 3F 77 2D 2D DE 36 4D
1040- 49 01 00
```

END OF LISTING 3