SOME SPECIAL FEATURES

Examples and discussions of special graphic features

using BASIC with machine language routines

1) Redefining Characters
2) Vertical Smooth Scrolling
3) Horizontal Smooth Scrolling

ATARI, INC.
CONSUMER PRODUCT SERVICE
PRODUCT SUPPORT GROUP
1312 Crossman Ave.
Sunnyvale, CA 94086

(800) 672-1404 inside CA
(800) 538-8543 outside CA

DEMOPAC #7
Rev.2 9-82/JB

DISCLAIMER OF WARRANTY ON PROGRAMS CONTAINED HEREIN

All computer programs contained herein are distributed on an "as is" basis by Atari, Inc. ("Atari") without warranty of any kind. Any statements concerning the capabilities or utility of the computer programs are not to be construed as express or implied warranties.

Atari shall have no liability or responsibility to the user or any other person or entity with respect to any claim, loss, liability, or damage caused or alleged. to be caused directly or indirectly by the computer programs contained herein. The entire risk as to the quality and performance of such programs is with the user.

Every effort has been made to ensure the accuracy of this document. However, because of ongoing improvements and updating of our computer software and hardware, Atari cannot guarantee the accuracy of printed material after the date of publication and disclaims any liability for changes, errors, or omissions.

Correspondence regarding this pack should be forwarded to Manager of Technical Support, Consumer Product Service, Atari, Incorporated, 1312 Crossman Avenue, Sunnyvale, CA 94086.

REDEFINING CHARACTERS
JB 8/82

The Operating System maintains a pointer to the ROM character set, which can be changed to point to your own character set in RAM. You are free to define an entire set at a RAM location of your choosing, or you may call the ROM set down into RAM and redefine only a few of the characters. The redefined characters appear on the screen in any text mode when the corresponding internal character code is placed in screen RAM, either by the display handler or by your program.

The program on the following page illustrates some techniques for redefining characters. The example uses an arbitrary RAM location (page 12 of memory) for the redefined set. Since it uses regular characters as well as redefined ones, the program calls the existing set from ROM into RAM, and redefines characters 1-7. The character numbers are the internal codes, and reflect the order of the ROM set. A chart of internal codes can be found on page 55 of the BASIC Reference Manual.

Characters 1-7 are chosen because they are special (non-letter) characters and are not used elsewhere in the program. Character number 0 is the space, so it cannot be redefined without filling the screen with the new character.

PEEKing each ROM location and POKEing into RAM takes a long time, so the actual transfer of data is accomplished with a simple USR call. The example uses upper case only, so the first half, or upper case set is called down. Each character requires 8 bytes of data, so the first half, 64 characters, takes 512 bytes. In the machine language routine, two loops of 256 bytes are used to accomplish the transfer.

Once the set is in RAM, new data is POKEd into the locations of characters 1-7. Again, 8 bytes of data are required for each character definition.

In order to access the new character set, location 756 (CHARBASE) is pointed to the chosen RAM location. In the example, the redefined characters are POKEd directly into screen RAM, and the original characters are PRINTed. This is for convenience; the redefined characters may be PRINTed, but you would have to refer back to the original character. For example, to get character number 1, you could use the statement:

POKE <screen location>,1 or
POSITION <screen location> :PRINT#6;"!"

The exclamation point is the original character number 1. If you use POKE, you do not have to keep track of the correspondence between original and redefined characters.

A simple animation effect is achieved by POKEing each redefined character into the same location, in series. Changing the length of the delay loop changes the speed of the animation.

```
1 REM REDEFINE CHARACTERS
2 REM JB 8/82
3 REM call character set from ROM into RAM with a USR function (listing
4 REM follows), redefine 7 characters (#1-7 of internal set)
5 REM then display characters in series for animation effect.
6 REM ************************************************************
10 GRAPHICS 2:REM .                          set up mode 2 screen
20 CHBAS=12:REM .                           new char set starts on page 12
30 REM .                                   (arbitrary location away from screen)
40 REM ************************************************************
50 DATA 104,169,0,133,204,133,206,169,224,133,205,104,104,133,207,162,2,160
55 DATA 0,177,204,145,206,136,208,249,230,205,230,207,202,208,240,96
60 FOR I=1536 TO 1569
70 READ X:POKE I,X:REM .                   poke in codes for usr function
80 NEXT I
85 REM ************************************************************
90 X=USR(1536,CHBAS):REM .                   pass address of new char set
95 REM ************************************************************
100 FOR CHAR=1 TO 7:REM .                    redefine characters 1-7
110 POS=(CHBAS*256)+(CHAR*8):REM .           address of character in new set
111 DATA 255,129,189,165,165,189,129,255
112 DATA 0,126,66,90,90,66,126,0
113 DATA 0,0,60,36,36,60,0,0
114 DATA 0,0,0,24,24,0,0,0
115 DATA 0,0,36,24,24,36,0,0
116 DATA 0,102,102,24,24,102,102,0
117 DATA 231,231,231,24,24,231,231,231
119 REM ************************************************************
120 FOR X=0 TO 7:REM .                       poke in bit pattern for character
130 READ A:POKE (POS+X),A
140 NEXT X:REM .                            each character is 8 bytes long
150 NEXT CHAR
155 REM ************************************************************
160 POKE 756,CHBAS:REM .                     point to new character set
170 SCR=PEEK(88)+256*PEEK(89):REM .          starting location of screen RAM
180 PRINT #6;"REDEFINED CHARACTERS"
190 FOR I=1 TO 7:POKE SCR+46+I,I:NEXT I:REM look at individual characters
200 POSITION 0,4:PRINT #6;"  ANIMATION EFFECT:"
210 REM ************************************************************
220 FOR I=1 TO 7:REM .                       display characters in series
230 POKE SCR+130,I
240 FOR DELAY=1 TO 25:NEXT DELAY
250 NEXT I
260 FOR I=6 TO 2 STEP -1:REM .               run series backwards
270 POKE SCR+130,I
280 FOR DELAY=1 TO 25:NEXT DELAY
290 NEXT I
300 GOTO 220
```

```
                10 ;MOVE CHARACTER SET
                20 ; USR ROUTINE
                30 ; JB 8/82
                40 ;
                50 ;  DEFINITIONS
                60 ;
00CC            70 CHARSET   =     $CC         ;free bytes on zero page
00CE            80 NEWSET    =     $CE         ;two more free bytes on zero page
                90 ;
0000            0100         *=    $600
0600 68         0110         PLA               ;take # of parameters from stack
0601 A900       0120         LDA   #0
0603 85CC       0130         STA   CHARSET
0605 85CE       0140         STA   NEWSET      ;lo bytes are both 0
0607 A9E0       0150         LDA   #$E0
0609 85CD       0160         STA   CHARSET+1   ;hi-byte of ROM character set
060B 68         0170         PLA
060C 68         0180         PLA               ;hi-byte of new location passed on stack
060D 85CF       0190         STA   NEWSET+1
060F A202       0200         LDX   #2          ;outside loop (2 loops of 256)
0611 A000       0210 LOOP1   LDY   #$00
0613 B1CC       0220 LOOP2   LDA   (CHARSET),Y ;get value from ROM location
0615 91CE       0230         STA   (NEWSET),Y  ;move to RAM location
0617 88         0240         DEY
0618 D0F9       0250         BNE   LOOP2       ;do 256 bytes
061A E6CD       0251         INC   CHARSET+1
061C E6CF       0252         INC   NEWSET+1    ;bump hi-bytes
061E CA         0260         DEX               ;outside loop
061F D0F0       0270         BNE   LOOP1       ;next 256 bytes
0621 60         0280         RTS               ;return from usr routine
```

SCROLLING
Vertical Fine and Coarse Scrolling
JB 9/82

A smooth scrolling effect is achieved by combining fine and coarse scrolls. Fine scrolling is used to scroll a character across a pixel, and coarse scrolling moves the character to the next pixel. Fine scrolling uses special registers, HSCROL and VSCROL, together with the scrolling-enable bits in the display list mode lines. Coarse scrolling is most easily done by manipulating the Load Memory Scan (LMS) address in the display list.

All display list instructions use the lower nybble. The top nybble is reserved for the four special display list functions:

D4: Enable Horizontal Fine Scrolling
D5: Enable Vertical Fine Scrolling
D6: Load Memory Scan Register
D7: Display List Interrupt

When bit D5 is set on a display list instruction, vertical fine scrolling is enabled on that line. Using decimal numbers, add 32 to each mode line to enable fine scrolling on the whole screen.

Once fine scrolling is enabled, set the VSCROL register (54277, or $D405). The number at VSCROL is the number of horizontal scan lines to scroll. Each pixel takes up 1 to 16 scan lines, depending on the mode. In BASIC mode 2, for example, each pixel is 16 scan lines high. To scroll a character halfway up the pixel, the value in VSCROL would be 8.

To fine scroll from one end of the pixel to the other, VSCROL must be incremented from 0 to 16, or decremented from 16 to 0. The character moves across the pixel until it reaches the last line. At this point, switch to the coarse scroll, to move it to the next pixel. VSCROL is set back to the other end, so that the image shows up on the correct side of the new pixel.

To accomplish a coarse scroll, change the LMS address by one line length. The LMS address is the starting location of screen memory. If screen memory starts one line length later, the whole screen image moves up one line. Remember that different modes take different numbers of bytes per line. BASIC mode 2 takes 20 bytes per line, so to scroll the screen up one pixel, add 20 to the LMS address. In a default display list, the first mode line (the fourth byte in the list) has the LMS bit set. The following two bytes (the fifth and sixth bytes in the display list) contain the LMS address.

The example on the following page sets up a display list for BASIC mode 2, which is ANTIC mode 7. Each mode line has the vertical scroll bit set, resulting in the instruction 39 (7+32). The first mode line also has the LMS bit set (7+32+64=103) and is followed by the LMS address. The original address is 0. The screen display moves through memory from the beginning. If you look carefully you can see the Real Time Clock at locations 18,19 and 20.

The machine language routine, which is executed during the vertical blank, increments the VSCROL register from 0 to 16. When it reaches 16 it is set back to 0, and the LMS address is incremented by a line length of 20. If the low byte exceeds 255, the high byte is incremented.

In the example the screen is scrolled smoothly through memory. In your own program, of course, scroll from the start of your own screen data to the end of your extended data area. For any scrolling screen, you must set up the data yourself, so that the LMS points to an area with valid data in it.

```
1 REM SCROLL
2 REM WB/JB 8/82
3 REM Vertical fine scrolling! a vblank routine scrolls through memory
4 REM using mode 2
5 REM ************************************************************************
10 GRAPHICS 2
20 REM ***** data for vblank code:(listing follows) *****
30 DATA 206,80,6,208,18,169,2,141,80,6,238,81,6,173,81,6,201,16,240,6
31 DATA 141,5,212,76,98,228,169,0,141,81,6,141,5,212,173,3,156,24,216,105
32 DATA 20,141,3,156,173,4,156,105,0,141,4,156,76,23,6
39 REM ***** data for display list *****
40 DATA 112,112,103,0,0,39,39,39,39,39,39,39,39,39,39,39,39,7,65,0,156
45 REM ************************************************************************
50 FOR I=1536 TO 1590:REM .            poke in vblank code on page 6
55 READ X:POKE I,X
60 NEXT I
70 FOR I=39936 TO 39956:REM .          poke in modified display list
75 READ X:POKE I,X
80 NEXT I
85 POKE 560,0:POKE 561,156:REM .       location of new display list
90 REM ************************************************************************
100 POKE 54286,0:REM .                 disable nmi
110 POKE 548,0:POKE 549,6:REM .        set up vblank vector
120 POKE 54286,64:REM .                reenable nmi
```

```
                10 ;SCROLL
                20 ; WB/JB 8/82
                30 ;
                40 ;DEFINITIONS
D405            50 VSCROLL  =    $D405
9C03            60 LMS      =    $9C03
0650            70 SPEED    =    $650
0651            80 TSCROLL  =    $651       ;temp shadow for scroll value
E462            90 XITVBV   =    $E462
0000           100          *=   $600
               110 ;
0600 CE5006    120          DEC  SPEED
0603 D012      130          BNE  END
0605 A902      140          LDA  #2          ;default speed every other vblank
0607 8D5006    150          STA  SPEED
060A EE5106    160          INC  TSCROLL
060D AD5106    170          LDA  TSCROLL
0610 C910      180          CMP  #16         ;top of pixel?
0612 F006      190          BEQ  COARSE      ;yes, coarse scroll
0614 8D05D4    200          STA  VSCROLL     ;no, fine scroll
0617 4C62E4    210 END      JMP  XITVBV
061A A900      220 COARSE   LDA  #0
061C 8D5106    230          STA  TSCROLL     ;back to bottom of pixel
061F 8D05D4    240          STA  VSCROLL
0622 AD039C    250          LDA  LMS
0625 18        260          CLC
0626 D8        270          CLD
0627 6914      280          ADC  #20         ;add a line length to lms address
0629 8D039C    290          STA  LMS
062C AD049C    300          LDA  LMS+1
062F 6900      310          ADC  #0
0631 8D049C    320          STA  LMS+1
0634 4C1706    330          JMP  END
```

SCROLLING
Horizontal Fine and Coarse Scrolling
JB 9/82

A smooth scrolling effect is achieved by combining fine scrolling, (moving an image across a pixel) with coarse scrolling (jumping an image to the next pixel). Fine scrolling requires two things: 1) a scrolling bit must be set in the display list instruction, and 2) the scrolling register must keep track of how far the image has gotten across the pixel. When the image is all the way across, it must be jumped to the next pixel. The coarse scroll is accomplished by changing the Load Memory Scan (LMS) address in the display list, so that it looks for data lower or higher in memory.

Horizontal scrolling differs from vertical scrolling in two important ways:

1) For horizontal coarse-scrolling, each horizontal line of data is defined separately. LMS must be set on every mode line of the display list. Instead of simply adding a line length to the one starting address of the screen, add (or subtract) one byte from the starting address of each line of display data.

2) The direction of the fine and coarse scrolls are opposite; when you reach the highest value in HSCROL, subtract from the LMS addresses. When you reach the lowest value, add to the LMS addresses. With vertical scrolling, the directions are the same. The values of VSCROL and of the LMS address both decrease or increase.

The vertical fine scroll value (at VSCROL) is measured in scan lines. The horizontal fine scroll value (HSCROL) is measured in color clocks. Different modes have different numbers of scan lines and color clocks per pixel. If the number of color clocks in the mode you are using is less than 16 (as it is in the example) you may fine-scroll across more than one pixel. The example on the following page fine-scrolls across 2 pixels, and then jumps each LMS address by 2 bytes.

You must use a customized display list for any kind of scroll. For vertical scrolling, simply set the Fine-Scroll bits of existing instructions. For horizontal scrolling, set both the Fine-Scroll bit and the Read-LMS bit on each instruction, and then follow each instruction with the two-byte address of the data area for that line.

The total data area for a line is determined by how far you want to scroll. For example, if your mode line is normally 20 bytes long, and you want to scroll across 4 screens of data, each data line must by 80 bytes long. Each LMS address would be the same as the last, plus 80.

In the example, each line of data is assumed to be 255 bytes long, and to start on a page boundary in memory. This simplifies the LMS-updating algorithm, as we do not have to worry about the low byte. In your own application, set up the data the way you want it. Take into consideration that a line of data may cross a page boundary, and you must update the low byte when necessary.

```
1 REM HORIZONTAL SCROLL
2 REM WEB/JB 9/82
3 REM set up custom display list, use VBLANK routine to smooth-scroll
4 REM horizontally- each line is 255 bytes long,(one page of memory)
5 REM of which 20 bytes are displayed at one time.
6 REM ************************************************************
10 GRAPHICS 0:PRINT "SETTING UP CUSTOM DISPLAY LIST..."
20 RESTORE 1000:REM .                    get data for custom display list
30 DL=16336:REM .                        display list will start at top of 16K
40 READ INSTRUCTION
50 IF INSTRUCTION=-1 THEN GOTO 100
60 POKE DL,INSTRUCTION:REM .             poke in display list instructions
70 DL=DL+1:GOTO 40
99 REM ************************************************************
100 PRINT "SETTING UP VBLANK ROUTINE..."
110 RESTORE 2000:REM .                   data for vblank code--listing follows
120 ADDRESS=1536
130 READ BYTE:IF BYTE=-1 THEN 200
140 POKE ADDRESS,BYTE:REM .              poke in object code for scroll routine
150 ADDRESS=ADDRESS+1:GOTO 130
199 REM ************************************************************
200 POKE 560,208:POKE 561,63:REM .       point to new display list
210 NMIEN=54286:VVBLKD=548:REM .         set up vertical blank vector
220 POKE NMIEN,0
230 POKE VVBLKD,0:POKE VVBLKD+1,6
240 POKE NMIEN,64
250 END :REM .                            VBLANK routine is in place...
260 REM .                                use joystick 0 to scroll screen
999 REM ************************************************************
1000 DATA 112,112,119,0,1,119,0,2,119,0,3,119,0,4,119,0,5,119,0,6
1010 DATA 119,0,7,119,0,8,119,0,9,119,0,10,119,0,11,119,0,12
1020 DATA 87,0,13,65,208,63,-1
1999 REM ************************************************************
2000 DATA 173,48,2,133,203,173,49,2,133,204,173,120,2,41,4,208,3,32,82,6
2001 DATA 173,120,2,41,8,208,3,32,33,6,76,98,228,173,254,6,201,15,240,10
2002 DATA 24,105,1,141,254,6,141,4,212,96,160,3,177,203,201,0,208,1,96,169
2003 DATA 0,141,4,212,141,254,6,177,203,56,233,2,145,203,200,200,200,192,42
2004 DATA 208,242,96,173,254,6,201,0,240,10,56,233,1,141,254,6,141,4,212,96
2005 DATA 160,3,177,203,201,234,208,1,96,169,15,141,4,212,141,254,6,177,203
2006 DATA 24,105,2,145,203,200,200,200,192,42,208,242,96,224,2,225,2,0,0,-1
```

```
                        *HORIZONTAL SCROLL
                        * VERTICAL BLANK ROUTINE
                        * read joystick 0 and scroll screen right or left
                        *  WBB/JB 9/82
                        *
                        * definitions
                        *
      = E462            XITVBV = $E462 ;exit vector
      = D404            HSCROL = $D404 ;horiz scroll register
      = 06FE            HSHADW = $6FE ;keep own RAM shadow
      = 0278            STICK0 = $278 ;joystick 0 register
      = 00CB            LMS = $CB ;temp lms adr
      = 0230            DL = $230 ;display list pointers
                        *
0000  = 0600             ORG $600
                        *
0600  AD3002             LDA DL ;display list location is starting point
0603  85CB               STA LMS ;from which to find lms adr
0605  AD3102             LDA DL+1
0608  85CC               STA LMS+1
                        *
                        * check joystick for horizontal motion
                        *
060A  AD7802            JOY1 LDA STICK0
060D  2904               AND #4 ;check bit d3 (0000 0100)
060F  D003 ^0614         BNE JOY2 ;if not 0, keep checking
0611  205206             JSR LEFT ;if 0, go move image left
0614  AD7802            JOY2 LDA STICK0
0617  2908               AND #8 ;check bit d4 (0000 1000)
0619  D003 ^061E         BNE END ;if not 0, exit
061B  202106             JSR RIGHT ;if 0, go move image right
061E  4C62E4            END JMP XITVBV ;exit normally
                        *
                        * right and left scroll routines
                        *
                        ****** scroll right ******
                        *
                        * fine scroll
                        *
0621  ADFE06            RIGHT LDA HSHADW ;remember last fine-scroll value
0624  C90F               CMP #15 ;limit of fine scroll? (2 pixels)
0626  F00A ^0632         BEQ R1 ;yes, go do coarse scroll
0628  18                 CLC ;otherwise, do fine scroll
0629  6901               ADC #1
062B  8DFE06             STA HSHADW ;keep new value
062E  8D04D4             STA HSCROL ;update register
0631  60                 RTS
                        *
                        *** coarse scroll ***
                        *
0632  A003              R1 LDY #3 ;new lms adr every 3 bytes
0634  B1CB                LDA (LMS),Y ;we're only looking at lo byte
0636  C900               CMP #0 ;limit of line size?
0638  D001 ^063B         BNE R2 ;no, do coarse scroll
063A  60                 RTS ;yes, limit reached, return
                        *
063B  A900              R2 LDA #0 ;reset fine-scroll register
```

```
063D  8D04D4        STA HSCROL
0640  8DFE06        STA HSHADW
                    *
0643  B1CB          R3 LDA (LMS),Y ;get each lms lo-byte
0645  38            SEC
0646  E902          SBC #2 ;subtract 2 to move 2 pixels right
0648  91CB          STA (LMS),Y
064A  C8            INY
064B  C8            INY
064C  C8            INY ;new lms every 3 bytes
064D  C02A          CPY #42 ;last one?
064F  D0F2 ^0643    BNE R3 ;no, keep going
0651  60            RTS ;yes, all lines scrolled, return
                    *
                    ****** scroll left ******
                    *
                    * fine scroll
                    *
0652  ADFE06        LEFT LDA HSHADW ;remember last fine scroll value
0655  C900          CMP #0 ;end of pixel?
0657  F00A ^0663    BEQ L1 ;yes, go do coarse scroll
0659  38            SEC ;no, continue fine scroll
065A  E901          SBC #1
065C  8DFE06        STA HSHADW ;remember new value
065F  8D04D4        STA HSCROL ;update register
0662  60            RTS ;fine scroll done, return
                    *
                    * coarse scroll
                    *
0663  A003          L1 LDY #3 ;lms lo byte every 3 bytes
0665  B1CB          LDA (LMS),Y ;check current lms lo byte
0667  C9EA          CMP #234 ;limit of line size?
0669  D001 ^066C    BNE L2 ;no, continue
066B  60            RTS ;yes, limit reached, return
                    *
066C  A90F          L2 LDA #15 ;reset fine scroll register (2 pixels)
066E  8D04D4        STA HSCROL
0671  8DFE06        STA HSHADW
                    *
0674  B1CB          L3 LDA (LMS),Y ;get each lms lo-byte
0676  18            CLC
0677  6902          ADC #2 ;move 2 pixels
0679  91CB          STA (LMS),Y
067B  C8            INY
067C  C8            INY
067D  C8            INY ;next lms, 3 bytes later
067E  C02A          CPY #42 ;last one?
0680  D0F2 ^0674    BNE L3 ;no, keep going
0682  60            RTS ;yes, return
```

no ERRORs, 17 Labels, $4A0E free.