9. More Bit-Mapped Graphics

Background

In this lab we will look at some more aspects of bit mapped graphics, such as line drawing and using the mouse as a graphics pointer.

Objectives:

Understand:

A. Basic line drawing operations

B. Using the mouse via the BIOS 33h function

C. Using the XOR operation for line erasure

Pre-Lab

Make sure that you read this entire lab before you attend class.

Lab

A. Using the Mouse to draw a horizontal line

The program below (*hline.asm*) illustrates how to draw a horizontal line and use the mouse to set the endpoints.

```
.model  small
.586
.stack 200h

.data
vmode  db  0
vpage  db  0
mse_x  dw  0    ;mouse X
mse_y  dw  0    ;mouse Y
del_x   dw  0    ;delta x
xor_flag  db 0
linecolor db 15   ; color 15 on default palette is WHITE

.code
main    proc
       mov     ax,@data
       mov    ds,ax
       mov    ah,0Fh       ; get current video mode
       int    10h
       mov    vmode,al
       mov    vpage,bh     ; save vid mode
       mov     ax,13h
       int     10h            ; Set video mode to 320x200x256.

       mov    ax,01          ;enable the mouse
       int    33h

mainlp1:      ; loop until key pressed or left mouse button pressed
        mov    ah,6
       mov    dl,0ffh
       int    21h
       jnz    mainexit    ;; exit if key pressed
       ;; check mouse left button
       mov    ax,3    ;; the CX/DX position information is the value
       int    33h    ;;
       test   bl,01h  ;; bx LSB = 1 if left button pressed
       jz     mainlp1 ;; loop if left button not pressed
;; left button pressed, save coordinates
       shr    cx,1   ;; Pixel X = mouse X / 2 for 320x200
       mov    mse_x,cx  ;; save X
       mov    mse_y,dx  ;; save Y
mainlp2:               ;; loop until button released
         mov ax,3
        int    33h
        test   bl,01h    ;; LSB = 0 if left button released
        jnz    mainlp2   ;; loop until button released


       mov    ax,2      ;; hide the  mouse before drawing
       int    33h

;; draw horizonatal line
        ;; compute delta_x = new_x - old_x
        shr    cx, 1     ; Pixel X = mouse X / 2 for 320x200
        sub    cx,mse_x  ; cx = new_x - old_x
        mov    bx,cx     ; bx = delta for HLINE call
        mov    cx,mse_y  ; cx = Y for HLINE call
        mov    dx,mse_x  ; dx = Y for HLINE call
        mov    ah, linecolor   ; AH = linecolor for HLINE call
        mov    al, xor_flag   ; AL = xor flag for HLINE call
        call   drawhline      ; draw line

        mov   ax,1
        int   33h    ;;turn mouse back on
        jmp    mainlp1
mainexit:;; restore mode
       mov    al,vmode       ;restore video mode
       mov    bh,vpage
       mov    ah,0
       int    10h
       Mov    ax, 4c00h    ;exit to DOS
       Int    21h
main   endp

;; draw horizontal line  DX: x, CX: y, BX: delta, AH: linecolor
;; AL = 0, do XOR,  AL = 1, don't do XOR
;; delta can be negative
drawhline  proc
       push   0A000h
       pop    ES          ;; set ES to Video segment
       ;; see if we have a negative delta
       or     bx,bx
       jns    drawh_sk1
       push    0ffffh          ;; save negative 1 on stack
       neg    bx         ;; bx = 0 - bx, so delta is now positivie
       jmp    drawh_sk2
drawh_sk1:
       push    0001h    ;; push a '1' on stack
drawh_sk2:
       mov    si,cx      ;; get y
       shl    si,6       ;; *64
       shl    cx,8       ;; *256
       add    si,cx
       add    si,dx      ;; add X to get starting address
       mov    cx,bx       ;; mov delta X to CX
       inc    cx         ;; inc CX to include line endpoint
       pop    bx         ;; get increment value into bx
drawh_lp1:
       mov    dl,ah       ;; move line color to dl
       cmp    al,0
       jnz    drawh_noxor
       xor    dl,es:[si] ;; xor line color with pixel value
drawh_noxor:
       mov     es:[si],dl ;; write pixel
       add    si,bx      ;; add increment value (-1 or 1) to SI
       loop   drawh_lp1
       ret
drawhline endp
end     main
```

This program first sets the graphic mode to 320x200x256. The program then enters a main loop where it waits for either a keypress (will exit program) or a left mouse button press. If the left mouse is pressed, the current mouse X,Y position is saved and then the program waits for the left mouse button to be released. Upon left button release, the program computes the change in the X coordinate value (delta_X) calls a procedure to draw a horizontal line starting with initial X,Y coordinates and the computed 'delta_X' value. If the delta_X is positive, the line goes from left to right. If the delta_X is negative, the line goes from the right to left.

How does this program work?

MAIN LOOP

1. Before the mouse can be read, it must first be displayed via the BIOS function 33h, AX=1. The current position of the mouse and button status is read via the BIOS function 33h, AX=3. The X,Y position of the mouse returns in registers CX, DX. This function always returns mouse coordinates in the range X=0..639, Y=0..199 regardless of the video mode. For the video mode 320x200, the mouse Y coordinate is the same as a pixel Y coordinate. However, the mouse X coordinate must be divided by 2 (shift right by 1) before we can save it as a pixel X coordinate. The button status is returned in register BX, where bit0 (LSB) is a '1' if the left button is pressed, and '0' otherwise. Bit 1 is used for the right button status.
2. After the left button is pressed, the pixel coordinates for the current mouse position is stored to locations *mse_x*, *mse_y*. The program then loops waiting for the left mouse button to be released by continuously calling INT 33h, AX=3 until the LSB of BX returns as '0'. After the button is released, the difference in the X values of the new X mouse position and the old X mouse position is computed by subtracting the old X from the new X. This *delta_X* value is needed by the horizontal line drawing procedure. Note that the *delta_X* value is the length of the line in pixels, and can be a negative value. A negative length means that the line is drawn starting at the initial X,Y location and is drawn from right to left.
3. The horizontal line procedure is called with the starting X,Y position, the *delta_X* (length of the line in pixels), the line color, and an XOR flag (explained later). The mouse is hidden via the INT 33h, AX=2 procedure before the horizontal line procedure is called because the mouse pointer can interfere with pixel values if the mouse is displayed during drawing. After the line is drawn, the mouse is turned back on via the INT 33h, AX=1 function.

DRAWHLINE Procedure

The *drawhline* procedure draws a horizontal line and expects an initial X,Y (dx=X, cx=Y), a *delta_X* (line length, bx= delta_X), linecolor (passed in AH), and an XOR flag passed in AL (explained below).

1. The *drawhline* procedure writes directly to video memory, which starts at location E000:0000. For the 320x200x256 video mode, each pixel takes a byte and is stored in row major order. So the first 320 locations (first row) correspond to X,Y locations of 0,0 to 319,0 (location 0,0 is in the upper left corner, location 319,199 in the lower right corner). The ES segment register is used to point to the video segment and is set to A000h within the *drawhline* procedure.

2. The starting offset of the first pixel is 320*Y + X. This calculation can be done as (256+64)*Y + X, or 256*Y+64*Y+X. Note that multiplication by 256 and 64 is just a shift left by 8 and 6 respectively. On older processors it was faster to do the calculation this way rather than by using the MUL instruction; on modern processors it may be faster to do a single MUL rather than multiple instructions.
3. The SI register will be used to point to the initial memory location corresponding to the X, Y value. As the line is written, the SI register will either be incremented (if the delta_X is positive and line is going left to right) or decremented (if the line is going from right to left). The delta_X value is initially tested, and a -1 will be stored in BX if negative or a +1 stored in BX if positive. The BX register is added to the SI register each time through the loop that writes a pixel.
4. The number of pixels to be written will be the absolute value of delta_X + 1 (the +1 means that endpoint of the line is included). This value is stored in register CX that is used to control the loop that writes the pixels.
5. The AL register value passed to DRAWHLINE controls whether or not the linecolor (passed in AH) is XOR'ed with the current pixel value or simply stored over the current pixel. The XOR operation can be used to 'erase' a line that was previous written. The first time the line is drawn, the new pixel value is the linecolor XOR'ed with the current pixel value. If the same line is drawn again, the XOR operation restores the previous pixel value (the line is 'erased'). The *xor_flag* memory value is passed to the DRAWLINE procedure by the main program - if '0' then the XOR operation is done, if nonzero then no XOR is done.
6. Note that the memory accesses to the pixels explicitly specify the ES segment register (mov ES:[SI], dl). By default, the SI register will use the DS segment register.

Lab Question 1: Assemble this program and execute it. To draw a line, hold down the left mouse button and drag either left or right. The line will appear upon button release. Hit any key to exit the program.

A. Draw several lines -- make sure that the xor_flag = 0 which means that the XOR operation is turned on in the DRAWHLINE procedure. What happens if you draw a new line over an old line? In your lab report, show how this 'erasing' operation occurs using a sample pixel value, and a line color value, and two XOR operations.

B. Set the 'xor_flag' value to a non-zero value and reassemble, re-execute the program. What happens now when you try to draw over old lines?

C. Write a procedure called DRAWVLINE that will draw a vertical line based on the passed parameters: X,Y (dx=X, cx=Y), a delta_Y (line length, bx= delta_Y, can be negative), linecolor (passed in AH), and an XOR flag passed in AL. Modify the main program so that a vertical line is drawn instead of a horizontal line when the left mouse button is clicked and dragged. Include the assembled listing of this program in your report. (HINT: most of the code in DRAWHLINE can be reused but you will need to increment/decrement the SI register by a different value)

B. Windows Bitmap files

A portion of the example *hline2.asm* program is shown below. This program is a duplicate of the *hline.asm* program except that a bitmap file is first loaded into video memory.

```
.model  small
.586
.stack 200h

.data
ifile db "h:\tmp\lab9\tpic.bmp",0
vmode  db  0
vpage  db  0
mse_x  dw  0    ;mouse X
mse_y  dw  0    ;mouse Y
del_x   dw  0    ;delta x
xor_flag  db 0
linecolor db 15   ; color 15 on default palette is WHITE

.code
extrn   ShowBMP:proc

main    proc
       mov     ax,@data
       mov    ds,ax
       mov    ah,0Fh       ; get current video mode
       int    10h
       mov    vmode,al
       mov    vpage,bh     ; save vid mode

       mov    dx,offset ifile
       call   ShowBMP   ;; expects 320x200x256 bitmap file

       mov    ax,01          ;enable the mouse
       int    33h
```

….. the rest of this program is the SAME as *hline.asm*

The program *hline2.asm* uses an external procedure called *ShowBMP* (found in Chapter 12 of the Irvine text and in the *bitmap.asm* file) to load a 320x200x256 bitmap file into video memory. The pathname of the file must be passed as a null-terminated string in DS:DX (the label 'ifile' shows you where I placed the file in my system, the 'tpic.bmp' file is available on the WWW page of this lab). If you place the bitmap file in a different location, be sure to edit this pathname (because it is a DOS pathname, make sure that all directory names in the path are 8 characters or less).

A windows bitmap file contains its own color palette for the image in the file, so part of the loading process sets the color palette to the palette found in the bitmap file.

Because of its length, the *ShowBMP* procedure is in a separate file (*bitmap.asm*). Assemble this file and produce a *bitmap.obj* file. Assemble the *hline2.asm* file and produce a *hline2.obj* file. To link these files together into one executable, do "link hline2+bitmap" . You will also need to specify the *irvine.lib* library when prompted for a library.

Lab Question 2: Assemble this program, execute it and draw several lines. Explain the differences between the lines you now see and the lines drawn by the original *hline.asm* program in your lab report. Set the *xor_flag* to a non-zero value and re-assemble, re-execute. Explain the differences between the lines you now see and the lines drawn by the original *hline.asm* program.

C. A Programming Task - A Rubber-banding Selection Box

Pick an empty spot on your Windows 95/98/ME/NT/2000/XP/MS/IS/EVL desktop/laptop, click the left button, and drag the mouse around. You see that 'rubber banding' selection rectangle that disappears when you release the left mouse button? Good -- your job is to modify the *hline2.asm* example so that you can duplicate this effect.

It is not as difficult as it may seem at first glance. When the left mouse button is initially pressed, write the X,Y position to memory locations: START_X, START_Y and LAST_X, LAST_Y. Then, enter a loop that continually reads the mouse position and button status. If the new position is different from the values in LAST_X, LAST_Y do the following:

A. If it is the FIRST time that the new X,Y is different from the LAST_X, LAST_Y values, draw a BOX with one corner being START_X, START_Y and the other corner being the new X,Y. Save the X,Y values in the LAST_X, LAST_Y memory locations. Make sure that you draw the two horizontal lines and two vertical lines that make up the box with the XOR mode turned on. You MUST hide the cursor before drawing any lines.

B. If it is not the first time that the new X,Y is different from the LAST_X, LAST_Y values, then you need to erase the last drawn box. This can be done by simply redrawing the box that was drawn last time (corners at START_X, START_Y and LAST_X, LAST_Y). After this box is drawn (the XOR mode causes this box to be 'erased'), draw a new box with corners at START_X, START_Y and the nex X,Y. Save the new X,Y values in the locations LAST_X, LAST_Y. Make sure that all lines are drawn (for both boxes) with the XOR mode turned on. You MUST hide the cursor before drawing any lines.

When the left mouse button is released, your program should exit the box drawing loop and re-enter the main loop to wait for another left mouse click. You must erase the last box that was drawn by drawing a box with corners at START_X, START_Y and LAST_X, LAST_Y. Note that the values of START_X, START_Y never change after the left mouse button is pressed so one corner of the rubber-banding rectangle is always fixed.

You will need to use the horizontal and vertical line drawing subroutines from section A to implement the box drawing procedure.

Lab Question 3: Include the assembled listing of this program in your lab report and make sure that you have at least one comment for every two x86 instruction lines.

Lab Report

A. Describing What You Learned

Include the answers to all "**Lab Questions**" in your report.

B. Applying What You Learned

Demonstrate the programs you wrote for Lab Questions 1 and 3 to the TA.