Behavior Research Methods, Instruments, & Computers
1995, 27 (4), 496-501

Bit-plane layering for high-resolution EGA and VGA graphics on the IBM PC/XT/AT

FRANK D. BOKHORST
University of Cape Town, Rondebosch, South Africa

Each page of video memory comprises four parallel planes that can be manipulated and displayed independently or in combination. A technique is described that involves programming the video hardware to achieve this. The utility of well-known video programming technology, such as the tachistoscope display, is thereby extended. Assembly language code is included, and a demonstration program is described.

The present paper describes a technique that can substantially increase the usefulness of the IBM PC/XT/AT video display in behavioral research. Bit-plane layering (Wilton, 1987) involves the deployment of up to four distinct graphics images that can be displayed in rapid sequence or together in any combination. Possible applications are the tachistoscope, animated displays, and composite images requiring manipulation of the parts. For example, the spacing of a grid superimposed on a complex image could be altered without affecting the underlying image.

Consider the video-based tachistoscope. A recent review by Haussmann (1992) has given what appears to be a definitive exposition of its implementation on currently available video hardware for the IBM PC/XT/AT and PS/2 family of computers. Its potential and limitations are well documented, and software for its application has been made widely available. Yet the technique of bit-plane layering is overlooked, and there appears to be no accessible exposition of this technique in the context of behavioral research methods. One reason for the oversight may be that bit-plane layering is applicable only to the newer EGA and VGA systems. Also, the degree of complexity the programmer encounters when dealing with EGA and VGA hardware may possibly account for the lacuna. Techniques are required that are not applicable to the older CGA, Hercules, and MCGA (PS/2) systems.

The video tachistoscope can display an almost unlimited number of fields. However, this applies only to displays lasting hundreds of milliseconds or to very simple text (e.g., a single word in nongraphics mode). Standard EGA and VGA technology does not allow the generation of complex graphics displays at a tachistoscopic rate (Creeger, Millar, & Paredes, 1990). For this reason, multiple fields that are to be displayed in rapid succession must be generated off-line and stored in separate areas ready for activation. A conventional technique in this situation is the use of several "virtual pages" in video RAM (Segalowitz, 1987), and all IBM PC/XT/AT video systems have this capability in text-only modes. Given sufficient video RAM, the Hercules, MCGA, EGA, and VGA systems can do this in graphics modes as well. Bit-plane layering extends this technology in two ways: In EGA and VGA systems, each page of video RAM actually consists of four independent planes. Since each plane holds a full-screen image, the number of images that can be deployed is quadrupled. Furthermore, the four planes can be displayed simultaneously in any combination, whereas the video RAM pages cannot.

The increased potential of EGA and VGA systems over CGA, Hercules, and MCGA systems derives from a radical innovation in the design of EGA and VGA hardware. The older systems were based on a linear mapping between video memory and physical display; in EGA and VGA systems, video memory is arranged in four planes that are mapped in parallel to the display. The primary purpose of this arrangement relates to the increased flexibility it affords in programming elaborate color displays. Essentially, each plane represents one color (for this reason, the term *color planes* is sometimes used). At the cost of reduced flexibility in color display, however, the presence of four parallel planes in video memory quadruples the number of independent monochrome images available. With two pages of video memory, one thus has eight display fields. The limitation is that each field represents only one color against a common background color. This is not very serious, and mixing of colors by showing two or more fields simultaneously is not excluded.

In what follows, only EGA and VGA graphics mode systems are dealt with. For simplicity, display modes other than "native resolutions" of 640 × 350 pixels in the EGA and 640 × 480 pixels in the VGA are omitted. CGA, Hercules, and MCGA systems are referred to only when drawing out contrasts.

The Hardware

The video system comprises at least eight full-screen images in up to eight foreground colors in any sequence and with up to four foreground colors combined in any

Correspondence should be addressed to F. D. Bokhorst, Department of Psychology, University of Cape Town, Rondebosch 7700, South Africa (e-mail: bokkie@psipsy.uct.ac.za).

manner. The hardware that makes this possible will appear complex to the newcomer. A simplified presentation follows, dealing only with components relevant to the bit-plane layering technique.

The most important components are the video buffer in RAM, which stores data for video display, the CRT controller chip comprising a system of programmable subcomponents responsible for displaying video RAM on the monitor, and the CPU, which provides data to modify the contents of video RAM. The CRT controller is also involved in how the CPU interacts with video RAM, as will be explained shortly.

As already mentioned, video RAM is arranged in four planes. These planes are arranged in parallel and appear to occupy the same address space in RAM. Consecutive pixels are arranged in a linear fashion (i.e., no interleaving as in CGA and Hercules) with eight pixels to a byte. The pixel, although it appears as only one bit, can take any of 16 values because it is derived from all four bit planes stacked in parallel at that address. For example, if the four planes contain, respectively, 1, 0, 1, and 0, then the pixel value is 5 (i.e., 0101 in binary).

How video RAM is displayed is determined by a subcomponent of the CRT controller called the *attribute controller*. This comprises a set of data registers, and one of these, called the *color plane enable* (CPE) register, can be loaded with a numeric value determining which one, or which combination, of the four bit-planes is actually shown on the monitor. For example, the binary value 0001 stored in the CPE register will cause only the first bit-plane to be displayed, whereas a value of 1010 would cause the second and fourth bit-planes to be displayed simultaneously. This takes effect immediately, in the same manner as page-swapping (and must therefore be properly synchronized, as discussed below in the Demonstration Software and Performance Validation section). A bit-plane excluded from the display by the CPE register does not contribute to the pixel value, regardless of the value actually in the bit-plane. So, if the bit-planes contain 0, 1, 0, 1, and if the first plane is excluded, then the pixel value would be 4 (i.e., 0100 binary).

In EGA and VGA graphics modes, video RAM cannot be accessed directly by the CPU. Instead, all CPU access to and from this space is mediated by four latches (a latch is merely a type of data register). Each latch holds 8 bits. Therefore, during read and write operations, 32 bits are actually transferred to and from RAM. The CPU is only indirectly involved in this transfer. Two components of the CRT controller, called the *graphics controller* and the *sequencer*, mediate data transfers between latches and RAM or between the CPU data registers and the latches. There is even a mode of operation in which the CPU data registers are not at all involved. However, only the sequencer concerns us here: The map mask register in the sequencer holds a numeric value that determines CPU data access to the latches and, hence, the bit-planes. For example, if the sequencer map mask register is loaded with a binary value of 0001, then latches 2, 3, and 4 are excluded from CPU updates to video RAM. On the other hand, the binary value 1111 enables updates to all four latches and, hence, to all four bit-planes. In this way, the four bit-planes can be selectively modified prior to display.

There is one other aspect of the hardware relevant to bit-plane layering—namely, the *palette* registers. The four-bit value of each pixel determines the actual color of each pixel on the screen indirectly through the palette registers. Four bits in all combinations yield the values 0–15, and each of these values points to 1 of 16 palette registers. The content of each register determines the final pixel color, so there can be 16 different colors. In the case of VGA, this is a little more complicated, because the palette register value is converted to an analog color signal, but details of this do not concern the bit-plane layering technique.

In summary then, bit-plane layering involves two key components of the CRT controller: the sequencer map mask register and the attribute controller CPE register. Data stored in these registers determine, respectively, which bit-plane can be updated and which bit-plane contributes to the display. Finally, the pixel value derived from whichever bit-planes are enabled is used to select 1 of 16 palette registers, and the value stored there determines the display color. General methods suitable for programming the CRT controller and palette registers are presented next.

Programming the Hardware for Bit-Plane Layering

It is not within the scope of this presentation to deal with general techniques of graphics programming. It is assumed that software is already available so that graphics images of the required complexity can be generated. Fortunately, control of the hardware involved in bit-plane layering is quite simple. The components are all accessible as hardware ports, and the data registers can be modified by writing to the relevant port addresses. Alternatively, the system BIOS software can be used to do this indirectly.

The color plane enable (CPE) register. Consider a simple example in which three disks colored red, green, and blue are superimposed to produce white against a black background. In a sense, this is what the video system actually does when you create a circle filled in white. Pixels making up the white area on the screen would all have the value 0111b (decimal 7). For the black area, pixel values of 0000b would be stored, meaning zero in all bit-planes. Now, since each bit in the value 0111b is stored in a separate plane, it is possible to "deconstruct" the white into red, green, and blue components. The following steps (shown here in pseudocode) would produce a white disk, followed by blue, green, and red disks in succession:

```
draw circle;
fill white;
pause;
repeat;
CPE register <- 0001b;
pause;
CPE register <- 0010b;
pause;
CPE register <- 0100b;
pause;
end repeat;
```

To load the CPE register with a particular value (stored in location CpeDat), execute the following code, here in assembler language, to call the BIOS video service:

```
mov  ah,10h      ; AH=10h to call BIOS video function to
                 ; modify attribute controller
mov  al,00       ; AL=0 requests action to set an
                 ; attribute controller register
mov  bl,12h      ; BL contains attribute controller
                 ; register number for CPE register
mov  bh,CpeDat   ; BH contains data for the CPE register
int  10h         ; Call BIOS video service to update CPE
                 ; register
```

If the repeat loop shown in pseudocode above is executed without the pauses, the result would appear as a flickering whitish disk. This well illustrates the first basic principle of bit-plane layering, concerned with control over the display using the CPE register in the attribute controller.

The map mask register. The other basic principle is concerned with generating separate images in each of the four planes by manipulating the sequencer map mask. In the previous example, all four bit-planes were updated in parallel using the default write mode. Suppose you want instead a circle in plane 1, a square in plane 2, and a triangle in plane 3. The following pseudocode would suffice:

```
CPE register <- 1000b;   /* Display only plane 4
                            while doing this */
sequencer map mask <- 0001b;
draw circle;
sequencer map mask <- 0010b;
draw square;
sequencer map mask <- 0100b;
draw triangle;
```

As shown above, it is possible to prepare these images while the three bit-planes are not visible—for example, by enabling bit-plane 4 for display through the CPE register, or by enabling video RAM page 1 for display while writing to page 2. To program the sequencer map mask register with a particular value (stored in location MapDat), execute the following assembler code:

```
mov    al,02        ; Sequencer map mask register is #2
mov    dx,3C4h      ; Sequencer address register port
out    dx,al        ; Request register #2
mov    al,MapDat    ; Data for map mask register
out    dx,al        ; Update map mask register
```

The palette registers. The code shown above produces a different color for each bit-plane that is enabled. This is because the bit-planes are used to represent different colors indirectly through the palette registers (and hence the alternative name *color planes*). To see the relation between the bit-plane numbers and palette registers, remember that the palette register numbers 1, 2, 4, and 8 correspond to binary pixel values of 0001, 0010, 0100, and 1000, respectively. The 1 in the first bit position from the right is stored in the first bit-plane, the second bit comes from the second bit-plane, and so forth. So, if bit-planes 1–3 are disabled, a 1 in plane 4 will cause only the color value in palette register 8 to appear on the screen.

When using bit-plane layering, it may be necessary that the four planes all have the same color. In this event, the contents of the palette registers must be modified to reflect this. Suppose, for example, that four red images must appear in rapid succession with no overlap, against a blue background, and that each image is stored in one bit-plane. To achieve this, the color value for red must be stored in palette registers 1, 2, 4, and 8, and the color value for blue in palette register 0. Assuming the color value is stored in ColorVal, and the palette register number is stored in PalReg, the following assembly language code shows how to set a palette register using the BIOS video services:

```
mov    ah,10h         ; BIOS video service 10h
mov    al,00          ; Function to update palette register
mov    bh,ColorVal    ; Color value
mov    bl,PalReg      ; Palette register number
int    10h            ; Call BIOS service
```

The default color values in EGA/VGA 16 color modes are 1 for midintensity blue and 4 for midintensity red. Assembly language code is shown here, but most high-level programming environments would include simpler ways to do the same thing.

Consider finally a more complex example, where the images in the four planes are allowed to overlap. In this event, it may be necessary also to control the color of the overlapping areas. For example, areas where bit-planes 1 and 2 overlap would show as midintensity cyan (i.e., color value 0011b) unless the value in palette register 3 is modified. The simplest case would be to load the same color value in palettes 1–15, and the background color in register 0, so that only two colors appear on the screen. On the other hand, if the value 60 were put in all registers except 0, 1, 2, 4, and 8, then any overlapping areas would appear in high-intensity red.

Demonstration Software and Performance Validation

The design of tachistoscope and animation applications raises special considerations about timing (Haussmann, 1992). There is a basic restriction on the flexibility of timing video displays arising from the hardware itself: The minimum display duration depends on the hardware vertical refresh rate, and increments in display duration must be multiples of this minimum. Also, modifications to the display should occur only during the vertical retrace period, when the electron beam is not visible. Since the vertical retrace period lasts only about 1 msec, the question arises, is it possible to program the attribute controller CPE register during this interval? Unfortunately, given the code shown above using the BIOS video services, this is not the case.

Empirical tests were done to determine the execution time of various code fragments using the method described in Sheppard (1987). On a 486 processor at 33 MHz with a 70-Hz refresh rate, it was found that it takes about 14 msec on average to update the attribute controller CPE register with a call to the video BIOS. Clearly, this procedure could not switch between two bit-planes during the vertical retrace interval. Instead, it takes about one vertical display cycle to complete. Fortunately, more efficient code is available for the same purpose. Wilton (1987) suggests the technique used in the assembly language listing presented in the Appendix. The relevant part begins at location CPEnable and ends at the point where interrupts are reset. This portion of code was tested in the same way as for the BIOS call and was found to execute in about half of one millisecond on average. It is therefore possible to coordinate events so that the CPE register is modified while the vertical retrace is in effect and, thus, to display successive bit-planes during each vertical screen refresh. The assembly language code in the Appendix shows how this can be done in the general case. Up to four bit-planes or combinations thereof can be displayed in sequence with a minimum duration of one screen refresh, or for any multiple thereof.

A demonstration program is available (see below) that uses an assembly language module similar to that in the Appendix to display successive red, blue, and green bit-planes on the same screen location at variable rates. A visual test of this procedure at maximum speed shows a homogeneous whitish field with slight flicker. The colors can also be modified, and when the three bit-planes are of the same color, there is no noticeable flicker. The three bit-planes merge into one seemingly continuous display.

Problems and Limitations

Only four bit-planes are available per video RAM page, so sequences of more than four different images based on the listing in the Appendix would need to perform page switching also. Obviously, this would be a minor modification only. Some high-level graphics programming systems may introduce a complication in that bit-plane 4 would apparently not be available. This is because the high bit in the pixel value is often used as an intensity bit. In this case, any data in bit-plane 4 are ignored, and the programmer's efforts to create an image in that plane are stymied. The solution to this might be to intervene directly in the graphics controller register settings. These details will not be discussed here (however, see Wilton, 1987). In the author's experience, it was necessary to circumvent entirely the high-level graphics drawing procedures when drawing on bit-plane 4, using instead a custom-built pixel drawing procedure. The restrictions this imposes are serious only if complex graphics images are plotted onto bit-plane 4.

Program Availability

A demonstration program may be obtained via electronic mail by request to the author at bokkie@psipsy.uct.ac.za or bokkie@uctvax.uct.ac.za. To obtain the demonstration on a DOS floppy disk, send $5 to the author, specifying the required disk size (3.5 or 5.25 in.) and density (double or high).

REFERENCES

Creeger, C. P., Millar, K. F., & Paredes, D. R. (1990). Micromanaging time: Measuring and controlling timing errors in computer-controlled experiments. *Behavior Research Methods, Instruments, & Computers*, **22**, 34-79.

Haussmann, R. E. (1992). Tachistoscopic presentation and millisecond timing on the IBM PC/XT/AT and PS/2: A Turbo Pascal unit to provide general-purpose routines for CGA, Hercules, EGA, and VGA monitors. *Behavior Research Methods, Instruments, & Computers*, **24**, 303-310.

Segalowitz, S. J. (1987). IBM PC Tachistoscope: Text stimuli. *Behavior Research Methods, Instruments, & Computers*, **19**, 383-388.

Sheppard, B. (1987, January). High performance software analysis on the IBM PC. *Byte*, pp. 157-164.

Wilton, R. (1987). *Programmer's guide to PC and PS/2 video systems.* Redmond, WA: Microsoft Press.

APPENDIX
Assembly Language Listing

```
;——————————————————————————————————————————
; Code fragment to program a sequence of bit planes for arbitrary
; display durations, synchronized with the vertical retrace signal.
; The EGA/VGA Attribute Controller color plane enable register is
; modified directly. Because the data and address registers share
; the same port at 3C0h, to write to the address register, first
; do an I/O read to the CRT status register. This toggles the
; Attribute Controller to accept a register address with the next
; I/O write to port 3C0h. A second write to port 3C0h will send
; data to the register selected in this way.
;
; Inputs:
```

APPENDIX (Continued)

```
;
; Plane4  four consecutive bytes to specify a sequence
;         of bit-planes for display.
; Count4  four consecutive words to specify the display
;         duration for each bit plane.
; BX      the number of bit planes to display, from 0-3,
;         to index Plane4 and Count4 as tables.
;DS       points to the segment where Plane4 and Count4
;         are located.
;
; Duration is a multiple of the vertical refresh rate with minimum
; of 1 for a single refresh cycle.
; Bit planes are numbered in sequence with binary values as follows:
;one = 0001; two = 0010; three = 0100; four = 1000
;
;------------------------------------------------------------
        mov     ax,40h
        mov     es,ax               ; ES = BIOS Data Segment
        mov     dx,es:[63h]         ; DX = 3B4 or 3D4
        add     dl,6                ; CRT status register address
;----- Synchronize with START of display refresh cycle
L01:
        in      al,dx                                   ; Get status
        test    al,8                ; Test bit 3
        jnz     L01                 ; Loop while in vertical retrace
L02:
        in      al,dx
        test    al,8
        jz      L02                 ; Loop while not in vertical retrace
;----- Enable a color plane during vertical retrace period
;      using code based on Wilton (1987).
CPEnable:
        cli                         ; Clear interrupts

        in      al,dx               ; Reset attribute controller flip-flop
        push    dx                  ; Keep status register port
        mov     dl,0C0h             ; DX has address register port (3C0h)
        mov     al,12h              ; Color plane enable register no.
        out     dx,al               ; Write data to register
        jmp     $+2                 ; Wait a while

        mov     al,Plane4[bx]       ; Color plane number from table
        out     dx,al               ; Write to data register

        pop     dx                  ; Get CRT status register port again
        in      al,dx               ; Reset the flip-flop again
        push    dx
        mov     dl,0C0h             ; DX has address register port (3C0h)
        mov     al,20h              ; Restore register number
        out     dx,al               ; Write to address register
        sti                         ; Enable interrupts
; ----- End of Wilton's procedure
        pop     dx

; ----- Timing loop to count vertical refresh cycles
        mov     cx,Count4[bx]       ; CX = Timing counter
L03:
        in      al,dx               ; Get status
        test    al,8                ; Test bit 3
        jnz     L03                 ; Loop if still in vertical retrace
L04:
        in      al,dx
        test    al,8
```

APPENDIX (Continued)

```
        jz      L04             ; Loop while not in vertical retrace
        loop    L03             ; Repeat until CX = 0

        test    bx,0Fh          ; BX = 0 ?
        jz      Exit            ; Done?
        dec     bx              ; Else, do next bit plane
        jmp     CPEnable
Exit:
;--------------------------------------------------------------
```

(Manuscript received May 17, 1994;
revision accepted for publication August 11, 1994.)