Programming the VGA

The Standard VGA mode 13h is the easiest and fastest color graphics programming. It uses a single long, linear bitmap, with each byte controlling one pixel. If I had a bitmap of 320x200 with 256 colors with each pixel represented by a single byte, I can quickly put that bitmap on the screen by copying it to memory location 0A000:0000h. This makes for extremely fast screen updates and flicker free animation. Also, if I have a small bitmap of 25x25 that I want to scroll across the screen, All I need to do is put the background on using the tech. above and then copy the 25x25 space of the background into a temp memory position, copy my 25x25 bitmap to that location, wait, copy the saved 25x25 background bitmap back into its place and move to the next position. This way I don't even touch the rest of background, and makes for even faster and better animation. I could even get better animation by rotating through 3 or more 25x25 bitmaps of a man walking while moving across the screen to give a better representation of animation.

Mode X is the undocumented mode of the VGA. It is very close to Mode 13h (320x200x256) except it has more vertical pixels (320x240x256) and we have to send our data to the VGA's registers rather than just putting it into 0A000:0000h

You can also program the VGA to have a resolution of 320x400 with 256 colors. This mode is still 13h but we now use 4 bitplanes. The standard VGA mode 13h is really a 320x400 res. mode, except the designers wanted to only use 64k so they made each line display twice, hence the 320x200 res mode. If we use 4 bitmaps, now we can take control of that second printed line.

In this discussion, I will include a source code listing for a small animation demo for the standard VGA mode 13h. You can get a copy of it here, or below, and unzip it (See my links page for PKUNZIP) in a directory of your choice, then view ANI.TXT for more info on running the demo and view the source code (MASM 5.1).

As far as the rest of the graphic modes described on this page, I suggest that you get a copy of the following book. It is a very good, source filled documentation on all the graphical modes of the CGA, EGA, VGA, MGA, MCGA, and many more. It includes a CD with all the code on it, and the complete text to his book, *Zen of Assembly Language*.

Michael Abrash's ZEN of GRAPHICS PROGRAMMING
2nd Edition 1996
The Coriolis Group
ISBN 1-883577-89-6: $44.99 (US dollars)
830+ pages of detailed documentation on programming graphics
See here for more books like this one.

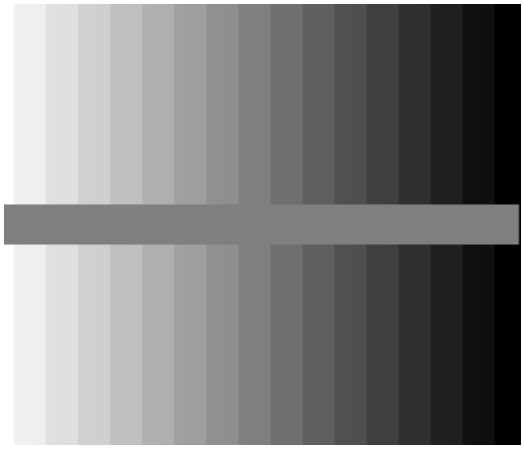
ANI.ZIP (31k)
To see the demo, type ANI at the DOS prompt.

Here is an example of MODE 12h video graphics with 4 plains using Write Mode 2.
Simply draws a colorful line from (0,0) to (479,479) on the 640x480x16 screen.

```
; This code was assembled with NBASM

.model tiny
.code
.186
           org 100h

           mov  ax,0012h               ; set mode to 640x480x16
           int  10h

           mov  ax,0A000h
           mov  es,ax

           ; start line from (0,0) to (639,479)
           mov  word X,0001h           ; top most pixel (0,0)
           mov  word Y,0001h           ;
           mov  byte Color,00h         ; start with color 0
           mov  cx,480                 ; 480 pixels
DrawLine:  call putpixel               ; put the pixel
           inc  word X                 ; move down a row and inc col
           inc  word Y                 ;
           inc  byte Color             ; next color
           and  byte Color,0Fh         ; 00h - 0Fh only
           loop DrawLine               ; do it
```

```
           xor  ah,ah                  ; wait for key press
           int  16h

           mov  ax,0003               ; return to screen 3 (text)
           int  10h

           .exit                      ; exit to DOS


; on entry X,Y = location and C = color (0-15)
putpixel   proc near uses ax bx cx dx

; byte offset = Y * (horz_res / 8) + int(X / 8)

           mov  ax,Y                  ; calculate offset
           mov  dx,80                 ;
           mul  dx                    ; ax = y * 80
           mov  bx,X                  ;
           mov  cl,bl                 ; save low byte for below
           shr  bx,03                 ; div by 8
           add  bx,ax                 ; bx = offset this group of 8 pixels

           mov  dx,03CEh              ; set to video hardware controller

           and  cl,07h                ; Compute bit mask from X-coordinates
           xor  cl,07h                ;  and put in ah
           mov  ah,01h                ;
           shl  ah,cl                 ;
           mov  al,08h                ; bit mask register
           out  dx,ax                 ;

           mov  ax,0205h              ; read mode 0, write mode 2
           out  dx,ax                 ;

           mov  al,es:[bx]            ; load to latch register
           mov  al,Color
           mov  es:[bx],al            ; write to register

           ret
putpixel   endp

X          dw 00h
Y          dw 00h
Color      db 00h

.end
```

Here is an "optical trick" you can do on a standard VGA in mode 13h, with 256 colors.
Sorry to those who's browsers don't view .BMP file formats. This was the only format that I could get to show this image with this much definition.



The middle bar is actually one color. Hold two pieces of paper up to the screen so you only see the middle bar.

This "trick" was found in:
More Tricks of the Game Programming Gurus, 1995, Sams Publishing
See here for more books like this one.

Here is the code to get this image.

```
; This code was assembled with NBASM

.model tiny
.code

           org  100h

           .start

           push ds                     ; make sure ds=es
           pop  es

           mov  cx,64                  ; set up our palette
           xor  ax,ax                  ;  of  0.0.0, 1.1.1, 2,2,2, ...
           mov  di,offset Palette      ;
PLoop:     stosb                       ;
           stosb                       ;
           stosb                       ;
           inc  ax                     ;
           loop PLoop                  ;

           mov  ax,0013h               ; set video mode to 320x200x256
           int  10h                    ;

           mov  dx,offset Palette      ; set the palette (DAC)
           xor  bx,bx                  ;
           mov  cx,64                  ;
           mov  ax,1012h               ;
           int  10h                    ;

           mov  ax,0A000h              ; point to VGA memory
           mov  es,ax                  ;

           mov  di,14464               ; place image in center of screen

           call Fade                   ; print top part

           mov  cx,10                  ; print bar (10 lines)
           mov  al,32                  ;  middle color
ALoop:     push cx                     ;
           mov  cx,128                 ;
           rep                         ;
           stosb                       ;
```

```
            add  di,192                 ;
            pop  cx                     ;
            loop ALoop                  ;

            call Fade                   ; print third part

            xor  ah,ah                  ; wait for key
            int  16h

            mov  ax,0003h               ; set screen back to text (80x25)
            int  10h                    ;

            .exit                       ; exit to DOS

Fade        proc near                   ; print first and third parts
            mov  cx,50                  ;  50 lines
PLoop1:     push cx                     ;
            mov  cx,64                  ; 64 colors
PLoop2:     mov  al,cl                  ;
            dec  al                     ;
            stosb                       ; 2 columns each
            stosb                       ;
            loop PLoop2                 ;
            add  di,192                 ;
            pop  cx                     ;
            loop PLoop1                 ;
            ret                         ;
Fade        endp                        ;

Palette     dup  768,?                  ; our palette buffer

.end                                    ; done assembling
```