"Love is not jealous, it does not brag, and it is not proud."

# 32 Mode 13h Programming

Mode 13h is considered to be the standard mode for graphics programming under DOS. Mode 13h programming is also referred as VGA programming or VGA register programming. Almost all DOS Game software uses this mode 13h.

## 32.1 Mode 13h

### 32.1.1 Palette Register

Mode 13h is supported by VGA cards. In this mode, we've got 256 colors and 320x200 pixel resolution. And thus it is sometimes referred as 320x200x256 mode.

In this mode 13h, we have 320x200 = 64,000 pixels. Each pixel takes 1 byte (8 bits) each. One important thing: these bytes do not hold color values; instead hold pointer or index to the color-lookup table. This lookup table is technically referred as '*palette registers*'. This lookup table is an array of 256 colors, each with 3 bytes. The structure of lookup table or palette register will be:

```
palette[256][3] = { {0, 0, 0},
                        :
                        :
                    };
```

**Note**

For the sake of simplicity, palette register is very often referred as a single dimensional array : palette[768].

| | Red | Green | Blue |
|---|---|---|---|
| Palette[255] | | | |
| Palette[254] | | | |
| | : | | |
| | : | | |
| | : | | |
| Palette[0] | 6 bits | 6 bits | 6 bits |

Here the 3 bytes hold Red, Green & Blue values. For example { 0,0,0 } represents White. Important note: VGA uses only 6 bits in each Red, Green & Blue bytes. So we can use $2^6$ combination of Red, $2^6$ combination of Green, $2^6$ combination of Blue values. And we have the maximum of $2^6 \times 2^6 \times 2^6 = 262144$ colors. Thus at a given time, the screen can have maximum of 256 colors out of the possible 262144 combination.

The next question is how to set these palette registers? We can use BIOS interrupts to set the palette registers. But it would be very slow and not good for professional programming. So we directly use the palette registers found on our VGA card. Palette registers are accessed via port 3C8h and 3C9h. First, we have to send 0 to port 3C8h and then the corresponding pixel values to port 3C9h. The sequences of operations should be:

1. OUT 0 at port 3C8h
2. OUT all pixel values one by one at port 3C9h (There would be 768 OUTs)

Another important point I want to insist is: loading palette registers refers to choosing 256 colors out of 262144 possible combinations and the screen holds just index or pointer to the look up table.

### 32.1.2 Vertical Retrace

The electron gun in our monitor refreshes each pixel with their current and correct values according to the refresh rate. The refresh rate may vary from system to system and usually it is 60Hz i.e., each pixel is refreshed in $1/60^{th}$ of a second. The electron gun fires electron at each pixel, row by row. Horizontal retrace is the time the electron gun takes to return from the right to left side of the screen after it has traced a row. For mode 13h programming, we don't bother about horizontal retrace.

Vertical retrace is the very short time in which the electron gun moves diagonally to the upper-left corner from the bottom-right corner of the screen, after tracing the entire screen. During the vertical retrace the screen is not being updated from video memory to monitor. So during this time if we update the screen, it won't result in flickering. In other words, you *may* get flickering if you don't consider vertical retrace. On the fast computers available today, it is not a big problem. However it wise to consider vertical retrace for good portability.

We can check the vertical retrace by noticing the value of the `INPUT_STATUS (0x3DA)` port on the VGA card. This is a number that represents the VGA's current state. Bit 3 tells if it is in a vertical blank. We first wait until it is not blanking; to make sure we get a full vertical blank time for our copy. Then we wait for a vertical blank. Now that we can update the whole screen. The following code fragment explains the concept.

```
#define INPUT_STATUS    (0x3DA)

/* copy the off screen buffer to video memory */
```

```
void UpdateBuffer(void)
{
   // wait for vertical re-trace
   while ( inportb(INPUT_STATUS) & (1<<3) )
          ;
   while ( !(inportb(INPUT_STATUS) & (1<<3)) )
          ;

   /* Now, copy everything to video memory */
   _fmemcpy( video_memory, off_screen, screen_size);
}
```

## 32.2 Optimization Note

When you program in mode 13h, you must understand the fact that our system RAM is faster than the video RAM. So real graphics programmers use a separate buffer (which will be stored in system RAM) for operations on the pixel values. And whenever the buffer value gets changed, it is being updated to the video RAM.

We may need to use mathematical functions like `cos( )`, `sin( )` etc with our graphics program for certain purpose. These functions would take more time to calculate. So it is wise to store the corresponding values in array when you begin your program. Now you can fetch the values for a given angle as `cos[30]` instead of `cos(30)`. It would almost double the speed of your program.