The Master VideoMode Function

Intro

As our video mode changing functions increase in complexity, it becomes very difficult to keep track of how we can switch from one mode to the next. An even larger problem is how to keep all screen based variables correct at all times. The creation of a Master Video Mode changing function would really be useful.

All video modes boil down to 3 basic types: Standard, VESA, and Tweaked. Our VideoMode function will be able to accept any of them and be able to use it correctly. If each type is enclosed in its own function, we will be able to easily add more modes as we progress without having to worry about modifying our master VideoMode function.

Function Layout

Before getting into anything with detail lets decide roughly what we want. We already know that we will be calling a specific function that actually does the video mode changing depending on which type of mode we are attempting to change to. Since we will be most likely using an offscreen buffer, we can delete the current buffer and reallocate one that fits the new resolution. I know this all sounds like a big pain in the butt, but i know that this will really help us in the long run by giving us a LOT of functionality. The best part is that all we need to remember after this function has been constructed is that the screen variables have been set correctly, the offscreen buffer is now large enough to fit the new resolution, and that we are working in a new video mode. So we are basically looking at a function something like this:

```
unsigned char Video::VideoMode(short Xres,short Yres, short Bits)
{ unsigned short index=0;

 index = GetModeIndex(Xres,Yres,Bits);
 if(index != 666)
  { VESAMode(index); }
 else
 { if(Xres ==320 && Yres == 200 && Bits==8)
    {StandardVideoMode(0x13);  }
   else if(Xres==80 && Yres==25)
    {StandardVideoMode(3);    }
   else
    {ModeX(Xres,Yres);  }
 }
 ReAllocate();
 return 1;
}
```

This is the function in a nut shell! As you can see it really isn't that complicated. What isn't realized right away is the real power of its layout. If you go back to the Video Modes tutorials (VESA, Standard, Tweaked) you will see that Screen attributes (Width, Height, XCenter, YCenter, Size) are set within each function. All we really need to do is try to set the best video mode according to the resolution and color depth supplied.

We first try to see if the resolution is possible under VESA. We want to try that first because we have a chance of getting access to a Linear Frame Buffer which would be really nice. This would be especially nice if some of our Tweaked Video Modes are supported! If VESA doesn't support it, we see if it is a Standard Video Mode. Notice that I'm only testing two video modes, 0x13 and 0x3 or 320x200 Graphics and 80x25 Text. The main reason for this is that I don't use any other standard video mode because they suck so bad! If you want to experiment with some more standard resolutions, you could build a function that tests all the resolutions and color depths to see if it can be supported and return a value like we did with the GetModeIndex function for VESA. Anyways, you can do that if you're crazy enough :) Last but not least, we default to calling the ModeX function because it is most likely some weird Tweaked Video Mode! After one of the functions has been set, we know that:

A. The new video mode has been set.
B. Screen variables have been set to the new mode's specifications.

All we have left is to re-allocate our offscreen buffer to match the size of our new video mode. The ReAllocate function does just this!

```
void ReAllocate()
{free(video_buffer);
 video_buffer = (unsigned char*)malloc(Screen_Size);

 if(video_buffer == NULL)
  {ERROR(2);
  }
  memset(video_buffer,0,Screen_Size);
}
```

This function is responsible for reallocating the offscreen buffer to the new size of the screen. We MUST use this function AFTER the VideoMode function so that Screen_Size is set correctly. Here we are de-allocating the current buffer, reallocating the new size, making sure that it allocated correctly and finally setting it to all 0's. Notice that this function is not testing wether video_buffer has been allocated before it frees it. On first run, if it isn't allocated we will be screwing things up! I use this function as listed because my offscreen buffer is allocated when I create an instance of my Video class. If you want, make sure that video_buffer is set to NULL when it is declared, then you can be sure that it has been allocated if it doesn't equal that.

I hope this is as useful to you as it is to me! Its always nice when you can create a function that is completely self sufficient, and one that lets you focus on other problems. If you have any questions, comments, rude remarks please give me some feedback!

I just wanted to write something in this spot so you didn't think that I wasted an entire piece of your paper, not to mention your ink to print this last page, just so I could put my lousy Contact Information in! Well, I guess I made this last page worth while with my little rant :)

Contact Information

You can get a hold of me through my website or direct email. Please feel free to email me about anything. I can't guarantee that I'll be of ANY help, but I'll sure give it a try :-)

Email : deltener@mindtremors.com
Webpage : http://www.inversereality.org

Created by
Justin Deltener