VGA Fonts

From OSDev Wiki

So you know how to display characters in text mode, and now you want to do it in graphics mode. It's not complicated, but definitely more complex than writing and ASCII code at a specific offset in memory. You'll have to do it pixel by pixel.

But how do you know what to draw? It's stored in data matrix called bitmap fonts.

Contents

Decoding of bitmap fonts

How is a character stored in memory? It's quite simple, 0 encodes background, 1 encodes foreground color. VGA fonts are always 8 bits wide so that each byte contains exactly one row. For letter 'A' in the typical 8x16 font it would be (in binary):

```
00000000b  byte  0
00000000b  byte  1
00000000b  byte  2
00010000b  byte  3
00111000b  byte  4
01101100b  byte  5
11000110b  byte  6
11000110b  byte  7
11111110b  byte  8
11000110b  byte  9
11000110b  byte 10
11000110b  byte 11
11000110b  byte 12
00000000b  byte 13
00000000b  byte 14
00000000b  byte 15
```

The full bitmap contains bitmaps for every character, thus it's 256*16 bytes, 4096 bytes long. If you want to get the bitmap for a specific character, you have to multiply the ASCII code by 16 (number of rows in a character), add the offset of your bitmap and you're ready to go.

How to get fonts?

There're several ways. You can have it in a file on your filesystem. You can hardcode it in an array. But sometimes 4k is so much that you cannot afford, and reading a file is not an option (like in a boot loader), in which case you'll have to read the one used by the card (to display text mode characters) from VGA RAM.

Store it in an array

Easiest way, but increases your code by 4k. There are several sources that provide the entire font in binary or source format so you do not need to manually write it out.

Store it in a file

Most modular way. You can use different fonts if you like. Downside you'll need a working filesystem implementation.

Get the copy stored in the VGA BIOS

It's a standard BIOS call (no need to check it's persistence). If you're still in real mode, it's quite easy to use.

```
;in: es:di=4k buffer
;out: buffer filled with font
push                    ds
push                    es
;ask BIOS to return VGA bitmap fonts
mov                     ax, 1130h
mov                     bh, 6
int                     10h
;copy charmap
push                    es
pop                     ds
pop                     es
mov                     si, bp
mov                     cx, 256*16/4
rep                     movsd
pop                     ds
```

Get from VGA RAM directly

Maybe you're already in protected mode, so cannot access BIOS functions. In this case you can still get the bitmap by programming VGA registers. Be careful that the VGA always reserves space for 8x32 fonts so you will need to trim off the bottom 16 bytes of each character during the copy:

```
                ;in: edi=4k buffer
                ;out: buffer filled with font
                ;clear even/odd mode
                mov                     dx, 03ceh
                mov                     ax, 5
                out                     dx, ax
                ;map VGA memory to 0A0000h
                mov                     ax, 0406h
                out                     dx, ax
                ;set bitplane 2
                mov                     dx, 03c4h
                mov                     ax, 0402h
                out                     dx, ax
                ;clear even/odd mode (the other way, don't ask why)
                mov                     ax, 0604h
                out                     dx, ax
                ;copy charmap
                mov                     esi, 0A0000h
                mov                     ecx, 256
                ;copy 16 bytes to bitmap
@@:             movsd
                movsd
                movsd
                movsd
                ;skip another 16 bytes
                add                     esi, 16
                loop                    @b
                ;restore VGA state to normal operation
                mov                     ax, 0302h
                out                     dx, ax
                mov                     ax, 0204h
                out                     dx, ax
                mov                     dx, 03ceh
                mov                     ax, 1005h
                out                     dx, ax
                mov                     ax, 0E06h
                out                     dx, ax
```

It worth mentioning that it has to be done **before** you switch to VBE graphics mode, because VGA registers are usually not accessible afterwards. This means you won't be able to map the VGA card's font memory to screen memory, and you will read only garbage.

Set VGA fonts

If you're still in text mode and want the VGA card to draw different glyphs, you can set the VGA font. It's worthless in graphics mode (because characters are displayed by your code there, not by the card), I only wrote this section for completeness. Modifying the font bitmaps in VGA RAM isn't hard if you read carefully what's written so far. I'll left it to you as a homework.

Set fonts via BIOS

Hint: check Ralph Brown Interrupt list Int 10/AX=1110h.

Set fonts directly

Hint: use the same code as above, but swap source and destination for "movsd".

Displaying a character

And finally we came to the point where we can display a character. I'll assume you have a putpixel procedure ready. We have to draw 8x16 pixels, one for every bit in the bitmap.

```
//this is the bitmap font you've loaded
unsigned char *font;

void drawchar(unsigned char c, int x, int y, int fgcolor, int bgcolor)
{
        int cx,cy;
        int mask[8]={1,2,4,8,16,32,64,128};
        unsigned char *gylph=font+(int)c*16;

        for(cy=0;cy<16;cy++){
                for(cx=0;cx<8;cx++){
                        putpixel(glyph[cy]&mask[cx]?fgcolor:bgcolor,x+cx,
                }
        }
}
```

The arguments are straightforward. You may wonder why to subtract 12 from y. It's for the baseline: you specify y coordinate as the bottom of the character, not counting the "piggy tail" in a glyph that goes down (like in "p","g","q" etc.). I other words it's the most bottom row of letter "A" that has a bit set.

Although it's mostly useful to erase the screen under the glyph, in some cases it could be bad (eg.: writing on a shiny gradiented button). So here's a slightly modificated version, that uses a transparent background.

```
//this is the bitmap font you've loaded
unsigned char *font;

void drawchar_transparent(unsigned char c, int x, int y, int fgcolor)
{
        int cx,cy;
        int mask[8]={1,2,4,8,16,32,64,128};
        unsigned char *gylph=font+(int)c*16;

        for(cy=0;cy<16;cy++){
```

```
                for(cx=0;cx<8;cx++){
                        if(glyph[cy]&mask[cx]) putpixel(fgcolor,x+cx,y+cy
                }
        }
}
```

As you can see, we have only foreground color this time, and the putpixel call has a condition: only invoked if the according bit in the bitmap is set.

Of course the code above will be excruciatingly slow (mostly due to doing one pixel at a time, and repeatedly recalculating the address for each pixel within the "putpixel()" function). For much better performance, the code above can be optimised to use boolean operations and a "mask lookup table" instead. For example (for an 8-bpp mode):

```
//this is the bitmap font you've loaded
unsigned char *font;

void drawchar_8BPP(unsigned char c, int x, int y, int fgcolor, int bgcolc
{
        void *dest;
        uint32_t *dest32;
        unsigned char *src;
        int row;
        uint32_t fgcolor32;
        uint32_t bgcolor32;

        fgcolor32 = fgcolor | (fgcolor << 8) | (fgcolor << 16) | (fgcolor
        bgcolor32 = bgcolor | (bgcolor << 8) | (bgcolor << 16) | (bgcolor
        src = font + c * 16;
        dest = videoBuffer + y * bytes_per_line + x;
        for(row = 0; row < 16; row++) {
                if(*src != 0) {
                        mask_low = mask_table[*src][0];
                        mask_high = mask_table[*src][1];
                        dest32 = dest;
                        dest32[0] = (bgcolor32 & ~mask_low) | (fgcolor32
                        dest32[1] = (bgcolor32 & ~mask_high) | (fgcolor32
                }
                src++;
                dest += bytes_per_line;
        }
}


void drawchar_transparent_8BPP(unsigned char c, int x, int y, int fgcolor
{
        void *dest;
        uint32_t *dest32;
        unsigned char *src;
```

```
        int row;
        uint32_t fgcolor32;

        fgcolor32 = fgcolor | (fgcolor << 8) | (fgcolor << 16) | (fgcolor
        src = font + c * 16;
        dest = videoBuffer + y * bytes_per_line + x;
        for(row = 0; row < 16; row++) {
                if(*src != 0) {
                        mask_low = mask_table[*src][0];
                        mask_high = mask_table[*src][1];
                        dest32 = dest;
                        dest32[0] = (dest[0] & ~mask_low) | (fgcolor32 &
                        dest32[1] = (dest[1] & ~mask_high) | (fgcolor32 &
                }
                src++;
                dest += bytes_per_line;
        }
}
```

In this case the address in display memory is only calculated once (rather than up to 128 times) and 8 pixels are done in parallel (which removes the inner loop completely).

The main downside for this approach is that you need a different function for each "bits per pixel", except that 15-bpp and 16-bpp can use the same code. For worst case (32-bpp) the lookup table costs 8 KiB. The lookup table for 32-bpp can be re-used for 24-bpp, and for 4-bpp no lookup table is needed at all. To support all standard bit depths that VBE is capable of; this gives a total of 5 versions of each "draw character" function (4-bpp, 8-bpp, 15-bpp and 16-bpp, 24-bpp, 32-bpp) and 3 lookup tables (8-bpp, 15-bpp and 16-bpp, 24-bpp and 32-bpp) which cost a combined total of 14 KiB of data if you use static tables (rather than dynamically generating the desired lookup table if/when needed).

See Also

- VGA Hardware

External Links

- UNI-VGA (http://www.inp.nsk.su./~bolkhov/files/fonts/univga/) - A free Unicode VGA font (.bdf)
- bdf2c (http://sourceforge.net/projects/bdf2c/) - .bdf font to C source converter.