CREATING AND DRAWING FONTS FOR GRAPHIC DISPLAY WITH LINEAR MEMORY MODEL

Filip Adamec
Doctoral Degree Programme (1), FEEC BUT
E-mail: xadame24@stud.feec.vutbr.cz

Supervised by: Tomáš Frýza
E-mail: fryza@feec.vutbr.cz

ABSTRACT

This article describes how to draw a character with variable font and size on graphical displays with a linear memory model. There are firstly described the restrictions for minimal draw algorithm demands, next how to store the font data on desired target and finally how to create a desired font.

1. INTRODUCTION

In many applications (field instruments, information panels etc.) it is necessary to represent text information via graphical displays. The displays have two types of memory organizations, paged memory or linear memory organizations. It is introduced an easy way how to create and draw text on displays with the linear memory model. This model can be seen in Figure 1.

1B	2B	.....	nB
(n+1)B	(n+2)B	.....	(2n)B
(2n+1)B	(2n+2)B	.....	(3n)B
⋮ ((r-1)n+1)B	⋮ ((r-1)n+1)B	.....	⋮ ((r-1)n+n)B

Figure 1: Relationship between display RAM and pixels

From Figure 1 it is obvious, that the best way to write a character bitmap to the display RAM memory is to write it by lines. Where *n* is number of pixels in x-axis and *r* is number of pixels in y-axis.

2. CHARACTER FORMAT AND GENERATION

The first problem need to be solved is how to store the font data. Because commonly C language is used the target application will be written in C language. The best way how to store a data is by arrays. It can be used two dimensional arrays, where the first dimension is a character number, that can be the ASCII number with some offset, and the second dimension is the character bitmap. We must now consider that we can have nice characters with variable width. With that condition it must be known width of each character and place it as a first number in

character bitmap. The type of our array depends on the maximum font width. For the font with eight pixel width it is BYTE variable, for 16 pixels it is WORD for 24 and 32 pixels it is DWORD variable.

Now is everything prepared to generate font. Because it is very hard to create it by own hands, an easy program in Visual C++ 6.0 was created. This program uses fonts from other programs and from operation system. In this case many fonts with variable size could be used. The basic of this program is very simple. It prints all printable ASCII characters to the memory bitmap on known position and then read it back pixel by pixel. After read back whole character, it prints that character into text file described above. In Figure 2 and 3 example of generated font for 16 bit font size, is shown.

```
const unsigned int font[224][17]  = {
……
{0xd,0x0,0x0,0x1c0,0x1c0,0x360,0x360,0x360,0x630,0x630,0x7f0,0xff8,0xc18,
0xc18,0x180c,0x180c,0x0},                    //A
{0xd,0x,0x,0xff0,0xff8,0xc0c,0xc0c,0xc0c,0xff0,0xff0,0xc0c,0xc0c,0xc0c,0x
c1c,0xff8,0xff0,0x0},                        //B
{0xd,0x0,0x0,0x1f0,0x63fc,0x70e,0x606,0xc00,0xc00,0xc00,0xc00,0xc00,0x606
,0x70c,0x3fc,0x0f0,0x0},                     //C
……
};
```

Figure 2: Font example

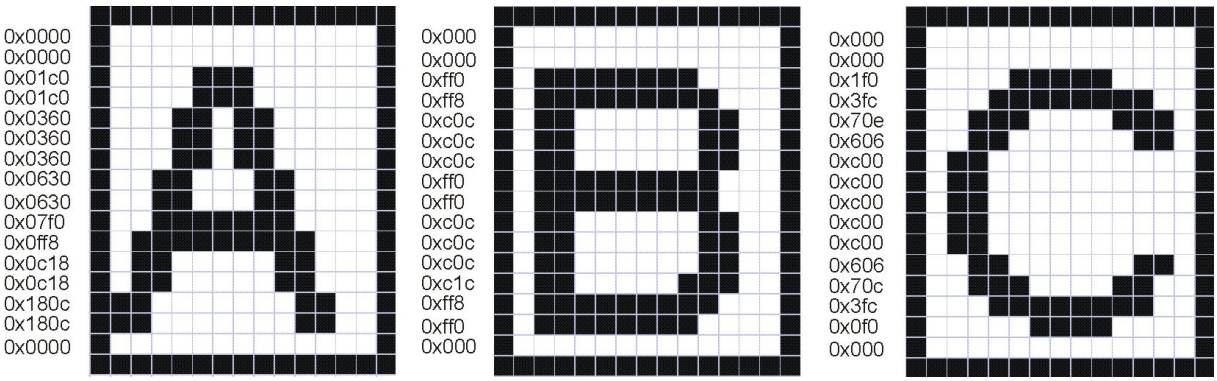

Figure 3: Bitmap of character “A”, “B” and “C”

3. DRAW CHARACTER

Now, is everything prepared for draw font on display. It must be considered the last thing and it is if an universal algorithm used or not. The universal algorithm can print all font sizes, but is less exacting than the type for fixed font size. I describe here the universal algorithm. The algorithm needs three global variables, first for current text position titled *cur_line,* then for count of pixel from beginning of a line *cur_pos* and the variable that represents actual character position on the line *cur_column*. The algorithm must change the line if the actually printed character could not be printed on actual line and do nothing if the actual number of line is higher than maximum. Also the interpret character for jump to new line is needed. The algorithm in C language is following.

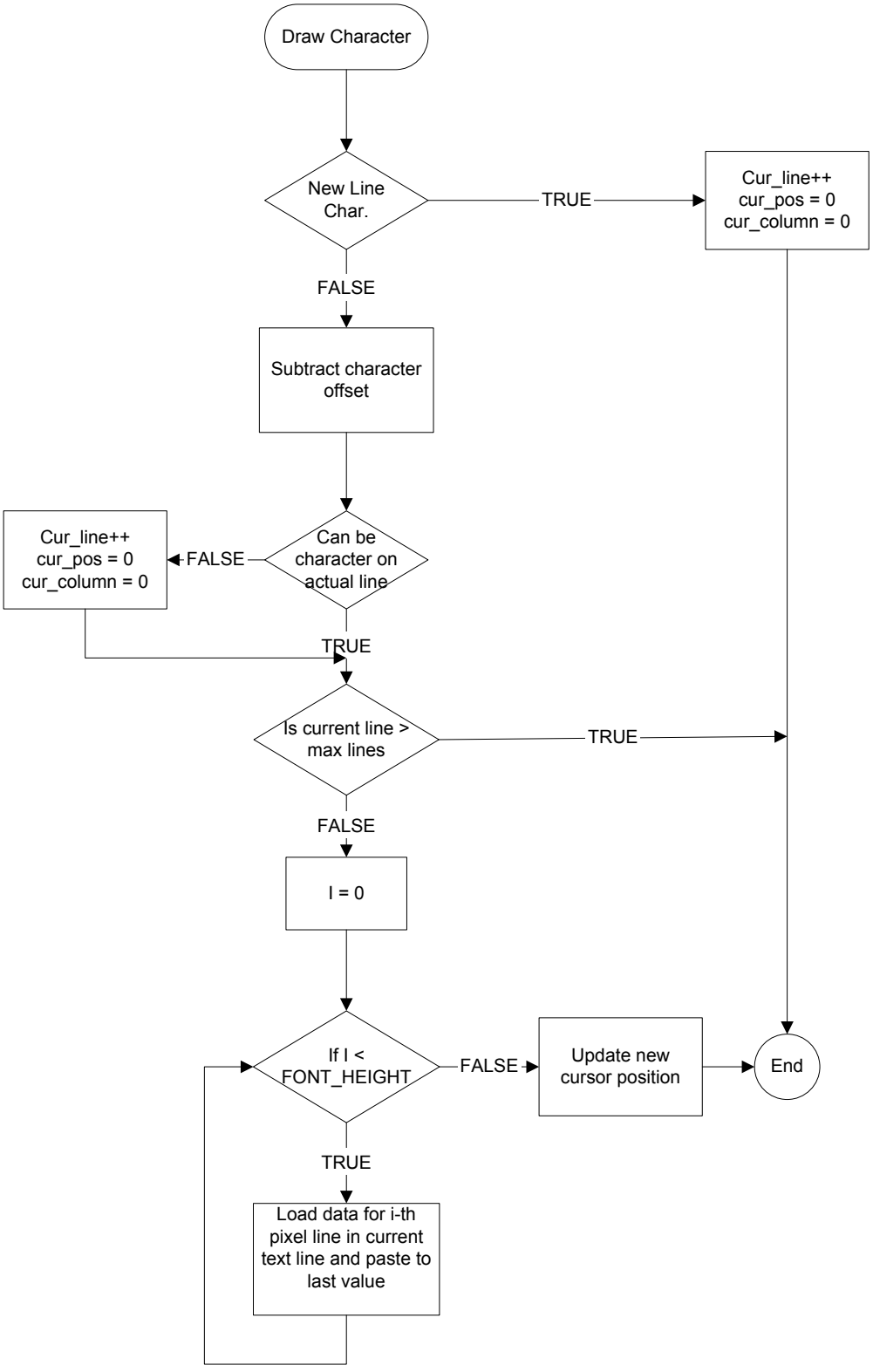

Figure 4: Proposed draw algorithm

The macro constant FONT_FIRST_CHAR is the offset for ASCII character, FONT_BYTES is the high of basic font array type and FONT_HEIGHT is height of font.

4. RESULTS

The best character size for the tested display was 16 pixels high. The 8 pixels high characters are badly visible and the higher pixels high have no meaning, because we can not represent useless information.

ACKNOWLEDGEMENTS

The author would like to thank for the financial support of the Czech Ministry of Education, Youth and Sports under grants no. MSM 002 163 0513 and MSM 102-08/H027.

REFERENCES

[1] David J. Kruglinski, George Shepherd a Scot Wingo. Programujeme v Microsoft Visual C++. Computer press 2000, ISBN: 8072263625.

[2] Toshiba, T6963T DOT MATRIX LCD CONTROL LSI. [online]. 1998 – [2.3.2009] Avaliable: http://audio.neora.ru/docs/T6963C_LCD_driver.pdf