Bitmap In The Terminal Using Braille Characters

02 January 2019 By An7ar35

Although not a new idea, using braille characters for the purpose of displaying bitmap is a very interesting and fun thing to look into.

A braille character is essentially a 2x4 pixel block:

A blank character: ⠀ Some dots on: ⡱ All dots on: ⣿

If we add blocks together we can create a pixel-map and thus, do pixel graphics on anything that supports UTF-8 braille character output (wide character support): ⠶⠶⡷

Braille character encoding

The Unicode standard on braille characters (range from `U+2800` to `U+28FF`) makes it very easy to produce any dotted patterns on a braille character with just simple additions.

A braille character is made up of 8 dots (2x4) and each are assigned a hexadecimal value (see right). A blank braille character's base hexadecimal value is 2800_{16}. If we want to turn on dots in a character we just need to add the inidvidual dot value to the base value.
e.g.: $2800_{16} + 1_{16} = 2801_{16}$ which produces: ⠁

To turn on more than one dot we just continue adding their values.
e.g.: $2800_{16} + 4_{16} + 20_{16} + 80_{16} = 28A4_{16}$ which produces: ⢤

01	08
02	10
04	20
40	80

Hexadecimal dot values

Bitmaps

Bitmaps can be constructed out of character blocks with sizes of $(height \times 4) \times (width \times 2)$ pixels.

For ease of use and clearer code, the braille dot values and base value are stored in an array and variable respectively:

```
unsigned BRAILLE_BLANK = 0x2800;
unsigned BRAILLE_DOT_VALUES[4][2] = {
    { 0x01, 0x08 },
    { 0x02, 0x10 },
    { 0x04, 0x20 },
    { 0x40, 0x80 }
};
```

A 2-dimensional array-like data-structure of hexadecimal values can be used for storing the bitmap (C++'s `std::vector<>` for example):

```cpp
auto char_map = std::vector<std::vector<unsigned>>(
                    char_height,
                    std::vector<unsigned>( char_width, BRAILLE_BLANK )
                );
```

A pixel setting method is needed to set the dots in the bitmap. From the (x, y) pixel given, the character block and its pixel must be deduced. To do that a simple division is used for the block and the modulo operator (`%`) for the block pixel.

Then the pixel's dot value within the block is added to the current block value:

```cpp
void setPixel( int x, int y, std::vector<std::vector<unsigned>> &char_map ) {
    int chars_x = x / 2;
    int pixel_x = x % 2;
    int chars_y = y / 4;
    int pixel_y = y % 4;

    char_map[ chars_y ][ chars_x ] += BRAILLE_DOT_VALUES[ pixel_y ][ pixel_x ];
}
```

With a little more code a simple working bit-mapper can be created. All is needed to print the hexadecimal values as UTF-8 characters is to set the locale prior to casting and sending the characters to the wide-character output stream.

Links & References

- DRAWILLE
- JP's Blog: Braille unicode pixelation
- Unicode Standard: Braille Patterns (pdf)