Bitmaps and Halftones with BM2FONT

Friedhelm Sowa
Heinrich-Heine-Universität Düsseldorf
Universitätsrechenzentrum
Universitätsstraße 1
D–4000 Düsseldorf
FRG
Bitnet: tex@dd0rud81

Abstract

The program BM2FONT converts different kinds of bitmap files to TEX fonts and writes an input file for integration of those graphics into documents. It is the link between a lot of graphic systems and TEX. The main part of BM2FONT is the conversion of colored pictures to halftone output. This paper describes the method of graphics integration done by BM2FONT and the most important aspects of the program.

Integration of graphics into TEX has been done in many ways within the last few years. Most of those methods use the `\special{whatever}` primitive that needs the special features of driver programs. Most of the public domain drivers do not support this kind of graphics integration. That seems to be the reason why the `special` primitive can't become the base for standard graphics integration. Another reason may be the lack of knowledge by TEX about the exact dimensions of the picture or the possible time lag when writing the code for the `special` command and the corresponding text into the `dvi` file.

The best way seems to be to treat graphics as text. In general, drivers do not have any problems printing documents that contain ordinary text. The only difficulty with this is to find the lowest common level of driver programs. In other words: What are their known bugs and what should a font look like to avoid falling into the trap? The answers to this question are:

- The character of a graphic font should be up to half an inch wide and high.

  We will not have overflow errors because of large run counts contained in packed font files.
- The graphic font should not contain more than 65.536 bytes.

  Even on small systems, drivers can load the font by allocating memory dynamically.

Taking into consideration the above, we can be sure of being successful when using most of the existing drivers for printing our documents.

But why not all of the existing drivers? Another lowest common denominator is the correct decoding of the operation codes in a dvi file. Inspite of the fact, that some of them are not used by the current version of TEX, some drivers (especially on small systems) have reduced the ability of using all the fonts, that are allowed by TEX. Some drivers dynamically load the font files into memory and stop the run, when the available memory is exhausted. Those drivers are not correct. There should exist only those limitations, which are concerned to TEX.

The steps to publishing a document should:

1. Make pictures using a suitable graphics system.
2. Translate the pictures by BM2FONT to a language TEX is able to understand.
3. Write the document.
4. TEX it.
5. Print it.

At no point of the process does the author have to be concerned with compatibilities with the output device of the publisher.

Supported Bitmap Formats

Let's start with unsupported vector graphics. This type of graphics information has to be converted to a bitmap format for integration via fonts. There are a lot of conversion programs available, both public domain and commercial, that do that job. So it would have been redundant to write any code concerning that item.

The decision on what formats to support was very easy to make. Only those bitmap formats that have become a quasi standard on the software market could be considered for the choice. The other condition was the availability of documentation. So

BM2FONT now supports the following bitmap formats:

PCX: The PCX format was introduced by ZSOFT. It uses runlength encoding and allows up to 256 colors (up to version 3.0, 16 colors).

TIFF: The Tag Image File Format of Aldus is one of the most used bitmap formats, especially for scanned pictures. The different compression methods of this format are not supported by BM2FONT.

IFF: The IFF standard was introduced by Electronics Arts for storing graphics in files. It was developed for the Amiga, and is now spread all over the PC world.

GIF: The graphics interchange format was developed by CompuServe in 1987. It uses the LZW-algorithm for data compression. If BM2FONT reads a fragmented file, it only uses the first picture; the following ones are ignored.

BMP: The device-independent bitmap format is found in the Windows world. It supports bitmaps as well as RLE compression and allows up to 256 colors. BM2FONT does not support RLE compressed pictures. The reason is that no examples have been available to the author to test the decompression coding.

IMG: The GEM Image File Format uses RLE, bitstreams, patterns, and repetitions. It is used by a lot of graphics systems, and not only on PCs.

CUT: The CUT-format is used, among others, by the ImagePro system, which captures pictures with a video camera. It supports up to 256 colors and uses runlength encoding compression.

Bitmaps: If the used bitmap format is not on this list and conversion to one of those formats is impossible, the picture can be extracted and written as pure bitmap by the user. BM2FONT accepts pure bitmaps, too.

Steps of integration

There is a picture file named `cheeta.gif` that seems to be good enough to illustrate an article. The document will be printed on a 1200-dpi device. Now it's time to use BM2FONT. To make sure that all files will be in the right directories, we set environment variables like

```
set texinputs=\tex\inputs
set texfonts=\tex\fonts\tfm
set dirpxl=\tex\fonts\pk
```

After that we run the program:

```
bm2font cheeta.gif -u5 -h1200 -v1200 -ry
```

and get the following files:

```
\tex\fonts\tfm\acheeta.tfm
        up to
\tex\fonts\tfm\xcheeta.tfm
\tex\fonts\pk1200\acheeta.pk
        up to
\tex\fonts\pk1200\xcheeta.pk
\tex\inputs\cheeta.tex
```

The file `cheeta.tex` contains the TEXnical description of our picture, generated for a rather high-resolution device.

```
\newbox\cheetabox
\newdimen\cheetaw
\font\acheeta=acheeta
\font\bcheeta=bcheeta
⋮
\font\wcheeta=wcheeta
\font\xcheeta=xcheeta
\setbox\cheetabox=\vbox{\hbox{
\acheeta !\bcheeta !\ccheeta !%
\dcheeta !\echeeta !\fcheeta !}}
\cheetaw=\wd\cheetabox
\setbox\cheetabox=\hbox{\vbox{
\hsize=\cheetaw
\parskip=0pt\offinterlineskip\parindent0pt
\acheeta !\bcheeta !\ccheeta !%
\dcheeta !\echeeta !\fcheeta !\vskip0pt%
\gcheeta !\hcheeta !\icheeta !%
\jcheeta !\kcheeta !\lcheeta !\vskip0pt%
\mcheeta !\ncheeta !\ocheeta !%
\pcheeta !\qcheeta !\rcheeta !\vskip0pt%
\scheeta !\tcheeta !\ucheeta !%
\vcheeta !\wcheeta !\xcheeta !}}
\ifx\parbox\undefined
    \def\setcheeta{\box\cheetabox}
\else
    \def\setcheeta{\parbox{\wd\cheetabox}
                  {\box\cheetabox}}\fi
```

All we have to do now is to put `cheeta.tex` into our document and then set it where we want to see it.

Figure 1: Cheeta, looking for a fat mouse

Dithering

For generating halftone pictures, single pixels must be represented on paper by black dots of different size. So we get the impression of different grey shades. Within a defined grid of bits for every pixel

of the original picture, a certain amount of bits are set from white to black. This is called dithering.

If a grid with u pixels in the width and c pixels in the height is used, we will have an amount of G grey shades, calculated by $G = 4uc$. A 3x3 grid will have the following rasters:

shade	raster 1	2	3	4
1	●○○ ○○○ ○○○			
2	●○○ ○○○ ○○○	○○○ ○○○ ○○●		
3	●○○ ○○○ ○○○	○○○ ○○○ ○○●	○○○ ○○○ ●○○	
4	●○○ ○○○ ○○○	○○○ ○○○ ○○●	○○○ ○○○ ●○○	○○● ○○○ ○○○
5	●○○ ●○○ ○○○	○○○ ○○○ ○○●	○○○ ○○○ ●○○	○○● ○○○ ○○○
6	●○○ ●○○ ○○○	○○○ ○○● ○○●	○○○ ○○○ ●○○	○○● ○○○ ○○○

⋮

shade	raster 1	2	3	4
20	●●○ ●●○ ●○○	○○● ○●● ○●●	●○○ ●●○ ●●○	○●● ○●● ○○●
21	●●● ●●○ ●○○	○○● ○●● ○●●	●○○ ●●○ ●●○	○●● ○●● ○○●
22	●●● ●●○ ●○○	○○● ○●● ●●●	●○○ ●●○ ●●○	○●● ○●● ○○●
⋮	⋮	⋮	⋮	⋮
34	●●● ●●● ●●●	●●● ●●● ●●●	●●○ ●●● ●●●	●●● ●●● ○●●
35	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ○●●
36	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●

For every row of the dithered picture the rasters r_i are used in the order:

r_1	r_4	r_1	r_4	r_1	r_4	...
r_2	r_3	r_2	r_3	r_2	r_3	...
r_4	r_1	r_4	r_1	r_4	r_1	...
r_3	r_2	r_3	r_2	r_3	r_2	...
r_1	r_4	r_1	r_4	r_1	r_4	...
r_1	r_4	r_1	r_4	r_1	r_4	...

The quality of a halftone picture depends on the resolution of the output device. A high-resolution device allows us to choose a greater grid with more grey shades than does a low-resolution device. BM2FONT dithers on the base of the matrix

row \ column	1	2	3	4	5	6	7
1	01	03	06	10	18	26	38
2	02	04	08	12	20	28	40
3	05	07	09	14	22	30	42
4	11	13	15	16	23	32	44
5	17	19	21	24	25	34	46
6	27	29	31	33	35	36	48
7	37	39	41	43	45	47	49

where the rows and columns define the height and width of the grid BM2FONT was told to work with. Each element of the matrix represents a grey shade. The different grey rasters are filled with black pixels in the corresponding positions up to that grey shade.

On paper a grey dot is composed by four different grey rasters so that we make one grey dot from four original pixels. This means that each grey shade may appear with four different expressions.

Gradation

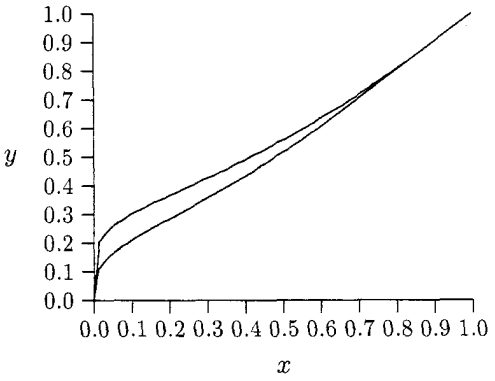
As a lot of printers tend to become dark very quickly in the darker part of the grey scale, it is necessary to compensate for this behavior. This is done by gradation.

The following function describes the development of the output values of halftone pixels, which are changed depending on parameters `t` and `z`. In the darker area ($x = 0$ represents black), an intensive lightening is done that decreases in the brighter area until it disappears in point x_0.

$$f(x) = \begin{cases} \frac{1}{2x_0^\alpha} x^{1+\alpha} + \frac{1}{2} x_0^\alpha x^{1-\alpha} & 0 \le x < x_0 \\ x & x \ge x_0 \end{cases}$$

$$\text{with } \alpha = \tfrac{t}{100},\ x_0 = \tfrac{z}{100}$$

With the default value 70 of both `t` and `z` BM2FONT generates a picture with low lightening, which works on 70 percent of the grey scale.



The upper curve represents the lightening when using value 80 for parameter `t` and 90 for parameter `z`. Because of memory problems — the plot was done by PICTEX — only very few points were used. That's why the curves look rather bad.

Gradation will not occur if BM2FONT was called with `-t0`. In the case of working on an original picture with only 16 or 32 grey shades or colors, gradation may produce a halftone picture by losing a lot of information. Then lightening should be done by using parameter `b`x, where x stands for the amount of shades to be subtracted from the available number of shades.

Error Distribution

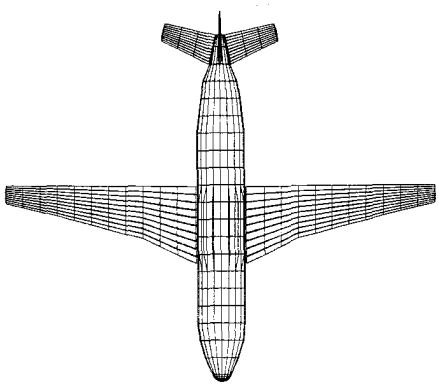
When generating halftones, there is a problem concerning the number of colors in the input file and the number of grey shades that are available depending on the raster size. Usually there is a difference, which means that we will loose some information when reducing the amount of colors. We get rounding errors. A well-known algorithm (Floyd/Steinberg) solves this problem by distributing the rounding errors to the neighboring pixels cumulatively.

BM2FONT uses a similar method by using different expressions of rasters. In spite of that, the loss of information is sometimes visible by very harsh changes of neigboring shades. For smoothing those changes, BM2FONT walks throw the pixel rows, looking for neighboring pixels with different colors. In this phase, the colors are already scaled down for output. Then the grey value becomes the average of the neighboring pixels by considering the rounding error. In most cases, a much better quality picture can be achieved by this method.

Nevertheless, sometimes it is not useful to distribute rounding errors this way. It usually makes sense when working on photographs. Comics or art work should be generated by switching off error distribution.

Some Examples

The first figure is converted from a vector graphics file (HPGL) to a PCX file by using the public domain `hp2xx`, which is written by Heinz W. Werntges, Heinrich-Heine-University Düsseldorf.



The following haltone images show the same picture of the same size. The second image is done by using a greater size of the dot raster.

Bibliography

Born, Günter. *Referenzhandbuch Dateiformate*, Addison Wesley Publishing Co., 1990.

Burger, Peter, and Duncan Gillies. *Interactive Computer Graphics*, Addison Wesley Publishing Co., 1989 .

Childs, Bart, Alan Stolleis, and Don Berryman. "A Portable Graphics Inclusion." *TUGboat* 10(1), pages 44–46, 1989.

Clark, Adrian F. "Halftone Output from TEX." *TUGboat* 8(3), pages 270–274, 1987.

Heinz, Alois. "Including Pictures in TEX." Pages 141–151 in *TEX Applications, Uses, Methods: TEX88 Proceedings*, Malcolm Clark, ed. (Ellis Horwood Series in Computers and their Applications, 1990.)

Knuth, Donald E. *The TEXbook, Computers & Typesetting*—A. Addison Wesley Publishing Co., 1986.

Knuth, Donald E. *The* METAFONT*book, Computers & Typesetting*—C. Addison Wesley Publishing Co., 1986

Knuth, Donald E. "Fonts for Digital Halftones." *TUGboat* 8(2), pages 135–160, 1987.

Messinger, Heinz. *Langenscheidts Großwörterbuch, "Der kleine Muret-Sanders", Deutsch-Englisch.* Langenscheidt, 1982.

Pickrell, Lee S. "Combining Graphics with TEX on IBM PC-Compatible Systems and LaserJet Printers." *TUGboat* 11(1), pages 26–31, 1990.

Pickrell Lee S. "Combining Graphics with TEX on IBM PC-Compatible Systems and LaserJet Printers, Part II." *TUGboat* 11(2), pages 200–206, 1990.

Rogers, David F. "Computer Graphics and TEX, A Challenge." *TUGboat* 10(1), pages 39–44, 1989.

Simpson, Richard O. "Nontraditional Uses of METAFONT." In *TEX Applications, Uses, Methods: TEX88 Proceedings*, Malcolm Clark, ed. Pages 259–271. (Ellis Horwood Series in Computers and their Applications, 1990.)

Ulichney, Robert. *Digital Halftoning.* The MIT Press Cambridge, Mass., 1987

Wilcox, Patricia. "METAPLOT: Machine-Independent Line Graphics for TEX." In *TUGboat* 10(2), pages 179–187, 1989.