The crucial line in this is the `grep`, which takes all the lines from `example.log` containing `% diagramf` and puts them in `example.dia`.

And so we've achieved labelled diagrams in METAFONT. The `diagramf` package is free software, and is available from the Aston archive.

3 Acknowledgements

The inspiration, and many of the original ideas, for this article came from Alan Hoenig's talk on the same subject at Cork. I'd also like to thank Jeremy Gibbons and Damian Cugley for comments, advice and allowing me to bounce ideas off them.

◇ Alan Jeffrey
Programming Research Group
Oxford University
11 Keble Road
Oxford OX1 3QD
Alan.Jeffrey@prg.ox.ac.uk

Graphics

X Bitmaps in TEX

Reinhard Fößmeier

Abstract

A new LATEX style, `bitmap.sty`, allows the direct inclusion of bitmaps from the X Window System in LATEX documents. With a tiny modification, the macros can be used with plain TEX, too.

Resumo

Nova ordonaro `bitmap.sty` *por LATEX permesas rektan enkludon de bit-matricoj el la fenestro-sistemo X en LATEX-aj dokumentoj. Post eta modifo, la makrooj estas uzeblaj ankaŭ por simpla TEX.*

1 Introduction

The X Window System uses a special C language syntax to describe bitmaps, images made up from black and white pixels. The syntax consists of several C definitions that specify the width and height of the bitmap and possibly the position of a "hot spot", and the declaration of a character array for the bitmap information, with initializers in hexadecimal notation (see figure 1). Each pixel line starts with a new byte; the last byte in a line is normally padded with zero bits.

```
#define bildo_width 50
#define bildo_height 30
static char bildo_bits[] = {
    0x00, 0x60, 0xff, ...
                     ...
          ... 0xe0, 0x01};
```

Figure 1: Example of the C bitmap format in X

Apart from being suited for inclusion in C programs where it can be processed by the Xlib routines `XCreateImage` or `XCreatePixmapFromBitmapData`, this format can be read by the Xlib routine `XReadBitmapFile` or written by `XWriteBitmapFile`. It is also supported by several programs, including a graphic editor (*bitmap*), conversion programs from/to ASCII character maps (*atobm*, *bmtoa*), and a screen dump utility (Bruce Schuchardt's *xgrabsc*). So it has become a true standard for the representation of bitmaps.

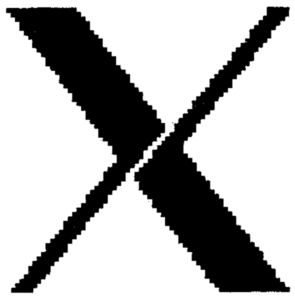

Figure 2: An example from the X Window System bitmaps (*xlogo64*)

2 The `bitmap.sty` style

A new LATEX style "bitmap" provides a macro to include and print such bitmaps in LATEX documents. The macro is called `\Bitmap` and has two arguments: the name of the file containing the bitmap, and the pixel size desired. The latter is saved in `\bmpsiz` and used to set the value of `\baselineskip`:

```
1 \newcount\bmhpoz
2 \newcount\bmwid
3 \newif \ifbmblack
4 \newdimen\bmrlen
5 \newdimen\bmpsiz
6 \catcode`,=\active
```

```
\def\Bitmap#1#2{\bgroup
\bmpsiz=#2
\baselineskip=\bmpsiz
\parindent=0pt
\bmrlen=0pt
```

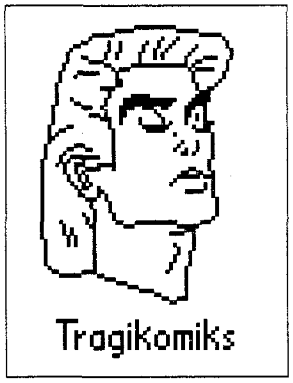

Figure 3: Example of an included bitmap, taken from an Asterix cartoon

The comma, separating the hex-numbers in the file, is made an active character; its function is to analyse the hexadecimal information and translate it into TEX `\vrule`s. It also throws away the leading `0x` of the hex numbers; the second and third argument (the hex figures) are swapped and each given to `\HexFig` for further analysis. Then, `\bmhpoz` is incremented by 8 columns and checked against the width of the bitmap; if the former exceeds the latter, the line is terminated and `\bmhpoz` is reset to 1.

```
\catcode`,=\active
\def,##10x##2##3{%
 \HexFig{##3}\HexFig{##2}%
 \advance\bmhpoz 8
 \ifnum\bmhpoz>\bmwid
 \0\hfil \vskip 0pt
 \bmhpoz=1 \bmrlen=0pt
 \fi}
```

The `\HexFig` macro analyses a hexadecimal figure and translates it to four calls of the macros `\0` or `\1`, for white and black pixels, respectively. As the hex figures `A` to `F` normally are not capitalized in bitmap files, `\uppercase` is used to convert them if necessary.

```
\def\HexFig##1{%
 \uppercase{\ifcase "##1}
 \0\0\0\0\or
 \1\0\0\0\or
 \0\1\0\0\or
 \1\1\0\0\or
```

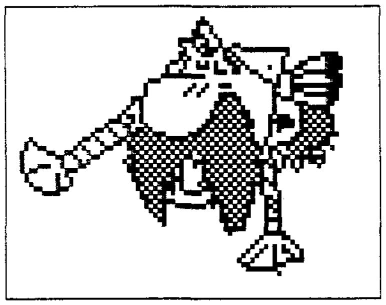

Figure 4: Another example (Abraracourcix, Gaulish tribe chief)

```
 \0\0\1\0\or
 \1\0\1\0\or
 \0\1\1\0\or
 \1\1\1\0\or
 \0\0\0\1\or
 \1\0\0\1\or
 \0\1\0\1\or
 \1\1\0\1\or
 \0\0\1\1\or
 \1\0\1\1\or
 \0\1\1\1\or
 \1\1\1\1\fi
 }
```

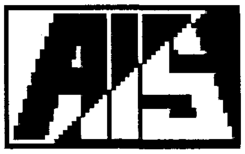

Figure 5: The bitmap logo of the *Akademio Internacia de la Sciencoj (AIS) San Marino*

Black pixels are printed as rectangular spots, produced by `\vrule`. To save space in TEX's memory, `\0` and `\1` collect sequences of equal bits and put them together to longer `\vrule`s. Invisible `\vrule`s of height 0 are used for the white space rather than `\hskip`s because they, too, save a little memory space. The length of the current sequence of `0` or `1` bits is kept in the dimension `\bmrlen`, the color of the current pixel is remembered in the value of `\ifbmblack`.

```
\def\0{\ifbmblack
 \vrule width \bmrlen height \bmpsiz
 \bmrlen=\bmpsiz \bmblackfalse
 \else
 \advance \bmrlen\bmpsiz
 \fi}
```

#1:	`#define bildo_`	(discarded)
#2:	`50`	width
#3:	`#define ... _bits`	(discarded)
#4:	`=`	(discarded)
#5:	`0x00, 0xe0, ... 0x01`	picture

Table 1: Arguments of \BmContent in the example of figure 1

```
45  \def\1{\ifbmblack
46   \advance \bmrlen\bmpsiz
47   \else
48   \vrule width \bmrlen height 0pt
49   \bmrlen=\bmpsiz \bmblacktrue
50   \fi}
```

The only pieces of information used from the bitmap file are the width of the bitmap and its picture information; height and hot spot coordinates are discarded. The contents of the file are read with \@@input; a macro \BmContent gathers the width as its second and the picture information as its fifth argument. The other arguments are ignored; their only purpose is to get rid of the rest of the bitmap header. \BmContent is prefixed with \expandafter, to capture the result of \@@input in its arguments.

```
51  \def\BmContent
52     ##1_width ##2 ##3[] ##4 ##5;{%
53   \bmwid=##2
54   \bmhpoz=1
55   ,##5
56   }% end of \BmContent
57  \expandafter\BmContent\@@input #1
58 \egroup}% end of \Bitmap
```

Finally, the comma's \catcode is reset to "other":

```
59 \catcode`,=12 % other
```

In the example of figure 1, the arguments of \BmContent would be like shown in table 1. Obviously, the 3rd and 4th argument could be combined into one; yet this way provides some more security against erroneous use.

The macros can be used with plain TeX, too, if \@@input in the last line of \BmContent is replaced with \input. The LaTeX macro \input doesn't work with the \expandafter technique.

Figure 2 shows the X logo bitmap, included in this document from the X bitmap file *xlogo64*. Figures 3 to 7 show some more examples, which may serve to show the versatility of even small bitmaps. All bitmaps presented here are far under 100x100 pixels.

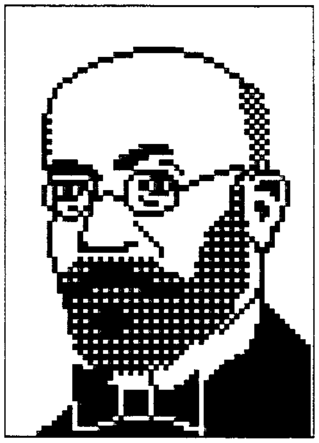

Figure 6: Yet another example ...

3 Possible extensions

To draw boxes around bitmaps, they can be put into a \vbox. In its present form, the macro doesn't compute the necessary width of such a box, although this information is available from the bitmap header. This functionality can be achieved by adding to \BmContent, after the definition of \bmwid in line 53, the statement

```
\hsize \bmwid\bmpsiz
```

Sometimes it is interesting to have statistics about the complexity of a bitmap. To this end, a \newcount\Complty may be introduced, initialized to zero and incremented by

```
\advance\Complty1
```

either in \1 before the \ifbmblack at line 45 (to count black pixels) or in \0 within the \ifbmblack at line 39 (to count \vrules). The result may be issued by a \message statement.

4 Problems, conclusion

Unfortunately, typesetting bitmaps is slow. A 60x85 bitmap (like the one shown in figure 6) can easily consume more of TeX's time than an ordinary text page. Moreover, the many \vrules consume a lot of TeX's space, even with runlength encoding. Tests show that drawing bitmaps with two printing characters (e. g., - and +) for \0 and \1 saves some time but needs even more space. For really big bitmaps, Big TeX may have to be used; it was, however, not necessary for this article.

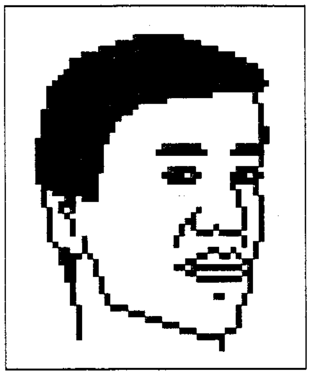
Figure 7: ... and another.

This problem could be alleviated by creating 16 special drawing characters for the bitmap patterns corresponding to the 16 hexadecimal figures. The complexity of a bitmap, however, will always be reflected in the cost of its typesetting. Remember that the aforementioned bitmap contains about as many pixels as there are characters on an average page.

Serious problems may arise if the bitmap file contains C language comments. They are discouraged when using `bitmap.sty`. The *bitmap* editor discards them anyway, so they aren't normally used; however, the *terminal* bitmap used in the X System contains a comment.

If the bitmap width is not a multiple of 8, the algorithm in "`\,`" depends on the last byte in each line being padded with 0's. This could be changed by putting the check of `\bmhpoz` against `\bmwid` into `\1`. It turned out to be hardly ever necessary in practice.

Pictures with small pixel sizes come out better if a multiple of the printer resolution is chosen for the pixel size. (Oh well, I know TEX input should be device independent...) Here sometimes the "big point" (`1bp` = 1/72 in) unit is useful if the resolution is related to inches; e.g. the resolution of a 300dpi printer is `0.24bp`; with 400dpi, `0.18bp`.

Although the use of bitmaps in documents is limited by TEX's resources, they provide a comfortable way to put small images into documents and a useful interface to the X Window System, e.g. for documentation.

◇ Reinhard Fößmeier
iXOS Software GmbH
Bretonischer Ring 12
D-W-8011 Grasbrunn
Germany
`refo@ixos.de`

Output devices

Report on the DVI Driver Standard

Joachim Schrod
Secretary
TUG DVI Driver Standards Committee

The DVI Driver Standard will be available in several stages. The basic stage is now called level 0. It covers only those driver capabilities which are really necessary to output a DVI document on an output device. All other driver capabilities will be called features (and may even be realized outside a driver). In the future we will publish several additional standard documents which will cover ranges of features; those documents will represent "tiers" built upon level 0 or on previous tiers. In this way they will be available as future stages of a complete standard. (One may doubt whether the standard will ever be complete as there may be always new features to standardize.)

The basic stage, level 0, consists of three parts:

(1) The pure standard document telling what a driver must be able to do.
(2) Definitions of all file formats spoken of in part 1.
(3) A rationale describing why the committee has chosen the given definition in part 1, recalling discussions that led to particular decisions.

A draft of the level 0 document is about to be published for public review. Part 1 of the draft is (almost) ready; a few spelling errors and such have to be removed. Part 2 was ready, but D. E. Knuth has changed the GF documentation, and this change must be incorporated. Part 3 exists only in part.

The committee will publish the draft as soon as possible. It may be that the draft of the rationale will not be finished in time; in that event we will publish part 1 by itself. This is considered to be useful (although not desirable) so that we will get responses very soon — and especially to change the status from "draft" to "released" as soon as possible. The file formats will not be published in *TUGboat*; they are available on several file servers. For people who do not have access to file servers I've prepared a brochure covering all file formats.

When complete, the standard will be published in the *TEXniques* series. The style will be modified slightly to follow formal standards conventions. The body of the standard will form the main text; this will be followed by a number of "annexes". The