JemTEX 2.00 available for Japanese

François Jalbert

I have just released Version 2.00 of my Japanese [LA]TEX system.

JemTEX is a freeware package containing everything needed to typeset beautiful Japanese text. You should, of course, already have a Japanese text editor, TEX, and METAFONT. Turbo-Pascal sources and executables are included for DOS computers. UNIX users are now supplied with a C program (gcc) so they too can enjoy JemTEX.

If you are interested in METAFONT code for japanese and chinese, the program `jis2mf` will interest you. This much improved program generates METAFONT code automatically out of 24 × 24 bitmap files. Smoother and better positioned japanese characters are the main improvements. METAFONT code for 61 japanese fonts of 128 characters covering punctuation, english, hiraganas, katakanas, and kanjis (level 1 and 2) is included indirectly in JemTEX.

My program `jem2tex` will turn the output of your favorite DOS or UNIX japanese text editor into a standard TEX, LATEX, or MTEX document. Thanks to several users from Japan, it handles fine points of japanese punctuation, spacing, and hyphenation much better than before. Switching to C for `jem2tex.exe` has also improved the speed substantially since Turbo-C has buffered I/O for non-text files, unlike Turbo-Pascal.

The file `JEMTEX2.ZIP` includes a 40-page-long user's guide `jguide.tex` where you can find all the details. It is (or soon will be) available from:

- `SIMTEL` (USA) (26.2.0.74)
  (tenex FTP or e-mail server)
- `utsun` (Japan) (133.11.11.11) (binary FTP)

Please feel free to contact me if you wish more information, or to be added to the mailing list which is only now officially being started. I plan on using it to keep everybody informed of new versions and bug fixes.

◇ François Jalbert
220 Forest
Châteauguay, QC
Canada J6J 1R1
`jalbert@IRO.UMontreal.CA`

Fonts

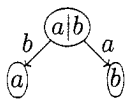
Labelled diagrams in METAFONT

Alan Jeffrey

1 Diagrams in METAFONT

In *TUGboat* 11(5), Alan Hoenig described a method of producing diagrams in METAFONT with labels provided by TEX. His method relied on passing information around via font dimensions. This is a standard method of passing information from METAFONT to TEX, but it has some drawbacks:

- There are only a limited number of font dimensions available, and each label uses up two of them.
- As METAFONT can only communicate with TEX via font dimensions, each label has to be assigned a font dimension, and it is difficult for the correspondence between font dimensions and labels to be kept automatically.
- Since TEX is providing the labels, and METAFONT is providing the diagrams, the diagrams have to be kept in a different file from the labels.
- There is no communication between TEX and METAFONT, so METAFONT cannot change the diagram depending on the size and shape of the labels. This is rather inconvenient for diagrams such as



  where the shape of the ovals depends on the size of the contents.

Fired with enthusiasm by Alan's talk at the European TEX Users Group meeting, I stole the best of his ideas, and slightly modified them to produce a simple METAFONT–TEX interface. This allows TEX code to be embedded within a METAFONT program, for example

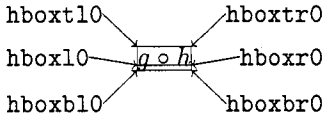
```
begindiagram(2,30pt#,7pt#,2pt#);
    hboxes(0);
    pickup pencircle scaled 0.4pt;
    .5[hboxl0,hboxr0] = (.5w,0);
    draw hboxbl0..hboxtl0
             ---hboxtr0..hboxbr0
             ---cycle;
    setbox0 "$g \circ h$";
enddiagram;
```

produces the diagram $\boxed{g \circ h}$. The new facilities used are:

`begindiagram(2,30pt#,7pt#,2pt#)` starts off diagram 2, which is 30pt wide, 7pt tall and 2pt deep.

`hboxes(0)` says that the only label we'll be using is number 0. This has a similar syntax to `labels`, so you can say `hboxes(1,2,7)` or `hboxes(3 upto 9)`.

`hboxl0` is the left point of label number 0, at the baseline. Similarly, `hboxbl0` is the bottom left, `hboxtr0` is top right, and so on. In this example, these points are



You can also use the numeric variables `hboxwd0`, `hboxht0` and `hboxdp0` which are the width, height and depth of label 0, and `hboxwd#0`, `hboxht#0` and `hboxdp#0` which are their sharp equivalents.

`setbox0 "$g \circ h$"` sets label number 0 to be $g \circ h$.

`enddiagram` finishes it all off.

The rest of the diagram is standard METAFONT. Within a TEX document you can use

`\diagramfile{example}` to load in the diagrams kept in `example.mf`,

`\diagramf{2}` to get the second diagram, and

`\everylabel` which is a token register added to every label, in the same fashion as `\everymath`. It should be set *before* saying `\diagramfile`.

These commands behave well inside groups, so if you say

```
\diagramfile{foo}
{\diagramfile{baz}\diagramf{1}}
\diagramf{2}
```

you get the first diagram from `baz` and the second diagram from `foo`.

2 How it all works

In the `diagramf` package, TEX and METAFONT communicate by auxiliary files, in a similar fashion to the MG TEX-PostScript interface ('Problems on the TEX/PostScript/graphics interface', *TUGboat* 11(3)).

When you run METAFONT on `example.mf` it reads in `example.dim`, which specifies the dimensions of all the boxes. In our example, part of `example.dim` is

```
wd#[2][0] := 20.3344pt#;
ht#[2][0] := 6.94444pt#;
dp#[2][0] := 1.94444pt#;
```

So, in diagram 2, label 0 has width 20.3344pt, height 6.94444pt and depth 1.94444pt. From this, METAFONT calculates where to put each label, and outputs a `.dia` file, containing TEX code. For example `example.dia` contains[1]:

```
\newdiagram{2}
\diagramlabel{0}{4.88908pt}{0pt}
$g \circ h$
\enddiagramlabel
\diagramchar{2}
\endnewdiagram
```

This tells TEX that diagram number 2 contains label 0 at coordinates (4.88908pt, 0pt) consisting of `$g \circ h$`. The diagram is character number 2 in the `example` font.

Similarly, when TEX encounters the instruction `\diagramfile{example}` it loads in `example.dia` and produces `example.dim`. And so we can have our METAFONT cake and eat it in TEX.

Well, almost. Unfortunately for all these grand ideas, METAFONT has *no* file-handling capabilities at all! The only files METAFONT generates are the `.tfm`, `.gf` and `.log` files.

This is rather annoying, but fortunately we can steal an idea from Section 7 of the Dirty Tricks appendix in *The METAFONTbook*. There, Knuth uses the `.log` file as a means of communicating between METAFONT jobs. Similarly, we use the `.log` file as a way of sending messages to TEX. Our `texoutput` macro is defined

```
def texoutput text t =
    for s = t:
        message s & "% diagramf";
    endfor
    message ""
enddef;
```

So `texoutput "Fred", "Ethel"` produces the output

```
Fred% diagramf
Ethel% diagramf
```

You can then use your favourite file-handling utility to filter the `.log` file, keeping only the lines containing `% diagramf`. On my UNIX set-up, for example, I have an alias `diagramf example` which expands out to

```
touch example.dim
mf example
grep "% diagramf" example.log > example.dia
echo Labels written on example.dia.
```

[1] Actually, each line ends with `% diagramf`.

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
```