Labeling Packages

- `labtex`: ask the author
- `diagramf`: archives such as Aston (tex.ac.uk) and SHSU (niord.shsu.edu)
- `PSFrag`: Aston
- `PSBox`: Aston
- `mfpic`: SHSU

Figure 15: *Labeling packages and how to get them.*

The author of `mfpic` makes no claim as to the completeness of his package. It is modeled on the famous PiC-TeX macros which attempt to the same thing, but entirely within TeX. Tom notes that `mfpic` is not complete, as not all of PiC-TeX's capabilities are a part of `mfpic`. He would like to see further enhancements made to `mfpic`, but he has no plans himself to do this.

A final approach to the problem of combining text with figures is a simple one: write your own implementation of a Metafont-like language to do this. That's the approach John Hobby took with his MetaPost system.

The macro packages I just mentioned appear to be freely available for use and can be downloaded from many good archives. I obtained `diagramf` from the `niord` server (anonymous ftp from `niord.shsu.edu` in directory `diagramf`) and the other two from the Aston archives (anonymous ftp from `tex.ac.uk`). `mfpic` is also available from `niord`.

5 TeX to Metafont

It's much easier to get TeX to talk to Metafont. TeX has many more file handling commands at its disposal, and it's perfectly possible to get TeX to create a file whose contents conform to proper Metafont syntax. Later, Metafont can read the file.

What purpose is served by having TeX talk to Metafont in this fashion? TeX can record metric information about sequences of letters it will typeset. Then, Metafont can use this information to create a character based upon the positions/measurements/states of characters which came before.

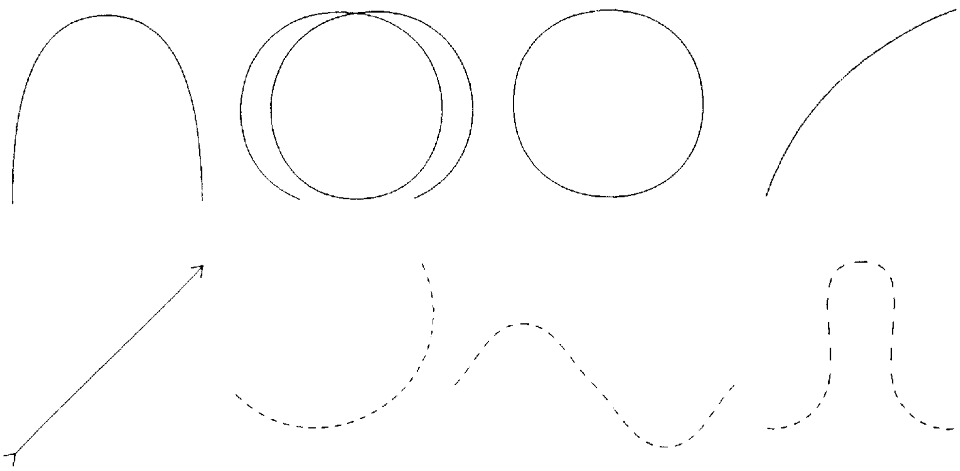

Figure 16: *Convex paths and non-convex paths (drawn with dashed lines).*

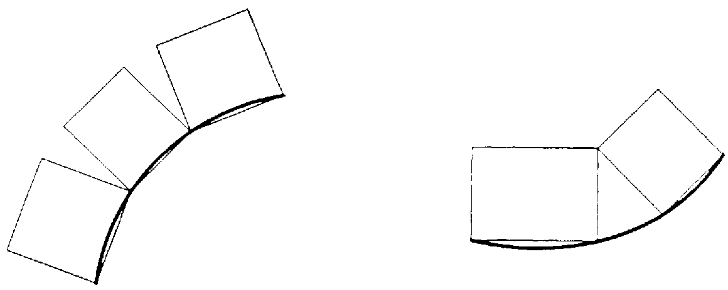

Figure 17: *Why we need a convex path.*

As a first application, suppose we want to typeset along a convex path. Informally speaking, a *convex path* is one which "sheds water." More precisely, if we start driving along it from the left to the right, we never turn our steering wheel to the left. See figure 16.

Figure 17 provides insight as to why we need a convex path. We imagine that the type sits on chords connecting to points of the curve. If the curve is convex, then the problem of placing adjacent types is a problem of the geometry of the curve. We simply butt the adjacent types at their bases, as in the left of figure 17. (The rectangles in this figure represent the bounding boxes of each character.) If the curve is not convex, the problem of placing adjacent type requires us to make sure that the types are far enough apart so they intersect only at their tops. How far apart shall they be? This is a much more difficult problem in general, although I shall say something about a special case below.

5.1 A Three-Pass Method

A three-pass method is necessary to create the special purpose font for curvilinear typesetting. Note that the end product is a special purpose font limited for *one use* only—the typesetting of one particular message around one particular path. This is quite different from the normal way we use Metafont. Normally, of course, a Metafont font is appropriate for an infinite number of messages. Fortunately, the processes involved—TEX and Metafont—do their jobs so swiftly that the making of special purpose fonts is not at all onerous.

Measure.
Make.
Mount.

Figure 18: *Measure—make—mount.*

Three passes are needed to create the special purpose font. First we *measure* the type with `measure.tex`. Then we *make* the type with macros contained in `make.mf` (plus others). Finally, we *mount* the letters on the page with `mount.tex`.

Let's examine the steps one by one.

5.2 Step One: Measuring the Message

TEX examines the text of your curved message in the first pass. The macros in this file, `measure.tex` examines each individual character in your message but don't typeset it. Using the techniques of exercise 11.5 (page 67) of *The TEXbook*, `measure` takes note of each character in the message and its width. It prepares two files for later use by Metafont. The file `measure.wid` contains `wid`th information about the individual characters in the message, while `measure.cha` contains information about the letters and `cha`racters themselves. Both these files are created using standard Metafont syntax. Shortly, Metafont will read these files.

"Exercise 11.5" ⇒ Each character ⇒ character information file / width information file

Figure 19: *Measuring the text.*

5.3 Step Two: Making the Message

One underlying assumption is that we are using the standard complement of Computer Modern fonts for curved typesetting. In preparation for step two, it is necessary to make a few changes to the standard font files. (Actually, such a strong assumption is not necessary. It is sufficient to have Metafont descriptions of the fonts, and with the appearance of utilities to convert outline font descriptions to Metafont descriptions, this is not a burden at all.)

The most important involves the actual program files. Typically, these have names like `romanu.mf` (containing the programs for Roman, uppercase letters) or `punct.mf` (with the programs for punctuation). In `romanu`, the program structure is something like

```
cmchar "The letter A";
beginchar("A", ...
 ...
endchar;
```

where the ellipses indicate the presence of material that is irrelevant to this discussion. We want to embed these definitions into Metafont macros. After all, there may be more than one 'A' in a curved text, and we need to be able to create as many A's as necessary. A series of tedious but trivial modifications transform the programs into subroutines. The idea is to relate a letter `n` with a subroutine `n_` which is a function of one argument—the angle of rotation that is necessary to rotate the letter and keep it on the curved baseline. The code fragment above might become

```
def A_(expr n, rotation_angle)=
 currenttransform:=identity rotated
                          rotation_angle;
 def t_=transformed currenttransform enddef;
 cmchar "The letter A";
 beginchar(n, ...
  ...
 endchar;
enddef;
```

and so on for the remaining character programs. We have to change the name of this file, and adjust the remaining files to call this file rather than the standard `romanu.mf` file.

- Fiddling with the macros.
- Measure special position.
- Measure angle of rotation.
- Pass the offset information to TEX.

Figure 20: *Making the special letters.*

Here are some other things we need do in the `make` step of the cycle. I use Metafont to draw the path along which the typesetting will go, and this path becomes the first character in our new font, `char0`.