COMPUTERS & TYPESETTING

JUBILEE • 2021 • edition

DONALD E. KNUTH

COMPUTERS & TYPESETTING / C

The METAFONTbook

Let's summarize the general contents of `logo.mf`, now that we have seen it all, because it provides an example of a complete typeface description (even though there are only seven letters):

- The file begins by defining ad hoc dimensions and converting them to pixel units, using **mode_setup**, **define_pixels**, etc.
- Then come programs for individual letters. (These programs are often preceded by macro definitions for subroutines that occur several times. For example, we will see later that the 'A' and the 'O' of the logo are drawn with the help of a subroutine that makes half of a superellipse; the definition of this macro actually comes near the beginning of `logo.mf`, just before the programs for the letters.)
- Finally there are special commands like **ligtable** and **font_quad**, to define parameters of the font that are helpful when typesetting.
- The file is accompanied by parameter files that define ad hoc dimensions for different incarnations of the typeface.

We could make lots of different parameter files, which would produce lots of different (but related) variations on the METAFONT logo; thus, `logo.mf` defines a "meta-font" in the sense of Chapter 1.

▸**EXERCISE 11.6**
What changes would be necessary to generalize the `logo` routines so that the bar-line height is not always 45 per cent of the character height?

Assignments ('`:=`') have been used instead of equations ('`=`') in the parameter files `logo10.mf`, `logo9.mf`, and `logo8.mf`, as well as in the opening lines of `io.mf` in Chapter 5; this contradicts the advice in Chapter 10, where we are told to stick to equations unless assignments are absolutely necessary. The author has found it convenient to develop the habit of using assignments whenever ad hoc dimensions are being defined, because he often makes experimental files in which the ad hoc dimensions are changed several times. For example, it's a good idea to test a particular letter with respect to a variety of different parameter settings when that letter is first being designed; such experiments can be done easily by copying the ad hoc parameter definitions from parameter files into a test file, provided that the parameters have been defined with assignments instead of equations.

TeX users have found it convenient to have fonts in a series of magnifications that form a geometric series. A font is said to be scaled by 'magstep 1' if it has been magnified by 1.2; it is scaled by 'magstep 2' if it has been magnified by $1.2 \times 1.2 = 1.44$; it is scaled by 'magstep 3' if it has been magnified by $1.2 \times 1.2 \times 1.2 = 1.728$; and so on. Thus, if a job uses a font that is scaled by magstep 2, and if that entire job is magnified by magstep 1, the font actually used for printing will be scaled by magstep 3. The additive nature of magsteps makes it more likely that fonts will exist at the desired sizes when jobs are magnified. Plain METAFONT supports this convention by allowing constructions like

```
\mode=cheapo; mag=magstep 2; input logo9
```

if you want to generate the 9-point METAFONT logo for the *cheapo* printer, magnified by 1.44 (i.e., by magstep 2). You can also write '`magstep 0.5`' for what TeX calls '`\magstephalf`'; this magnifies by $\sqrt{1.2}$.

The sharped forms of dimensions are actually represented by plain METAFONT in terms of printer's points, so that '$pt^\#$' turns out to be equal to 1. However, it is best for programmers not to make use of this fact; a program ought to say, e.g., '$em^\# := 10pt^\#$', even though the '$pt^\#$' in this construction is redundant, and even though the computer would run a few microseconds faster without it.

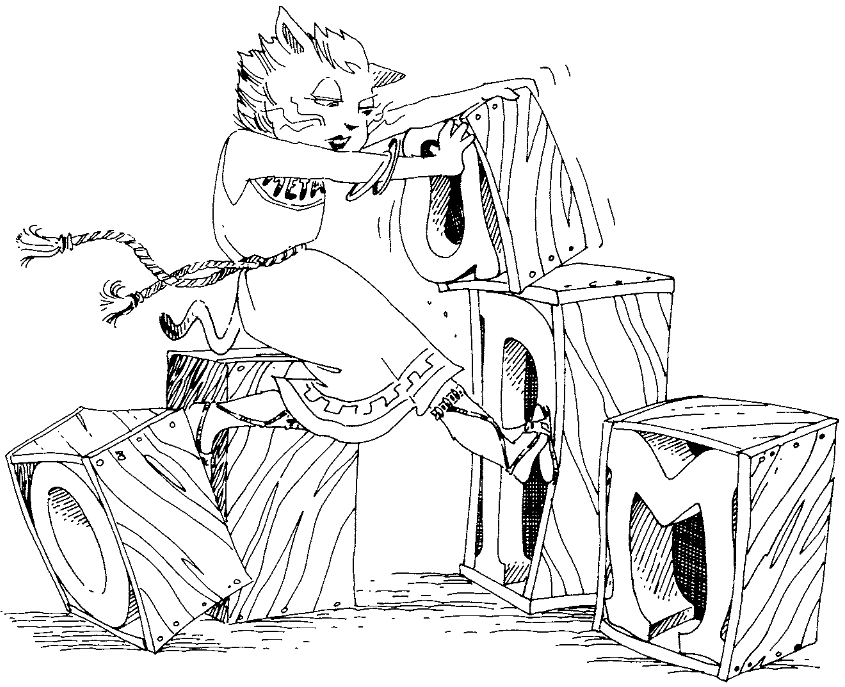
▸ **EXERCISE 11.7**
Suppose you want to simulate a low-resolution printer on a high resolution device; for concreteness, let's say that *luxo* is supposed to produce the output of *cheapo*, with each black *cheapo* pixel replaced by a 10×10 square of black *luxo* pixels. Explain how to do this to the `logo10` font, by making appropriate changes to `logo.mf`. Your output file should be called `cheaplogo10.2000gf`.

A great Temptation must be withstood with great Resolution.
— WILLIAM BURKITT, *Expository Notes on the New Testament* (c. 1700)

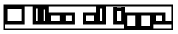
What some invent, the rest enlarge.
— JONATHAN SWIFT, *Journal of a Modern Lady* (1729)

12

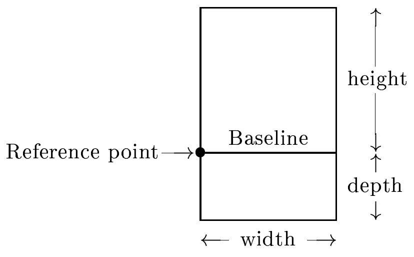
Boxes



Let's pause now to take a closer look at the "bounding boxes" that enclose individual characters. In olden days, metal type was cast on a rectangular body in which each piece of type had the same vertical extent, although the type widths would vary from character to character. Nowadays we are free of the mechanical constraints imposed by metal type, but the former metaphors are still useful: A typesetting system like TeX imagines that each character fits into a rectangular box, and words are typeset by putting such boxes snugly next to each other.

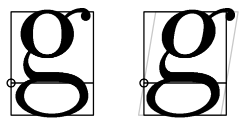
The main difference between the old conventions and the new ones is that type boxes are now allowed to vary in height as well as in width. For example, when TeX typesets 'A line of type.' it puts boxes together that essentially look like this: ''. (The 'A' appears in a box '□' that sits on a given baseline, while the 'y' appears in a box '□' that descends below the baseline.) TeX never looks inside a box to see what character actually appears there; TeX's job is to put boxes together in the right places on a page, based only on the box sizes. It is a typeface designer's job to decide how big the boxes should be and to create the characters inside the boxes.

Boxes are two-dimensional objects, but we ascribe three dimensions to them because the vertical component is divided into two quantities, the *height* (above the baseline) and the *depth* (below the baseline). The horizontal dimension is, of course, called the *width*. Here is a picture of a typical box, showing its so-called reference point and baseline:



The example characters in previous chapters have all had zero depth, but we will soon be seeing examples in which both height and depth are relevant.

A character shape need not fit inside the boundaries of its box. Indeed, *italic* and *slanted* letters are put into ordinary boxes just as if they were not slanted, so they frequently stick out at the right. For example, the letter 'g' in the font you are now reading (`cmr10`) can be compared with the '*g*' in the corresponding slanted font (`cmsl10`):



The slanted '*g*' has been drawn as if its box were skewed right at the top and left at the bottom, keeping the baseline fixed; but TeX is told in both cases that the box is 5 pt wide, 4.3055 pt high, and 1.9444 pt deep. Slanted letters will be spaced properly in spite of the fact that their boxes have been straightened up, because the letters will match correctly at the baseline.

Boxes also have a fourth dimension called the *italic correction*, which gives TEX additional information about whether or not a letter protrudes at the right. For example, the italic correction for an unslanted 'g' in `cmr10` is 0.1389 pt, while the corresponding slanted letter in `cmsl10` has an italic correction of 0.8565 pt. The italic correction is added to a box's width when math formulas like g^2 or g^2 are being typeset, and also in other cases as explained in *The TEXbook*.

Plain METAFONT's **beginchar** command establishes the width, height, and depth of a box. These dimensions should be given in terms of "sharped" quantities that do not vary with the resolution or magnification, because the size of a character's type box should not depend in any way on the device that will be used to output that character. It is important to be able to define documents that will not change even though the technology for printing those documents is continually evolving. METAFONT can be used to produce fonts for new devices by introducing new "modes," as we have seen in Chapter 11, but the new fonts should still give the same box dimensions to each character. Then the device-independent files output by TEX will not have to be changed in any way when they are printed or displayed with the help of new equipment.

The three dimensions in a **beginchar** command are given in reverse alphabetical order: First comes the width, then the height, then the depth. The **beginchar** routine converts these quantities into pixel units and assigns them to the three variables w, h, and d. In fact, **beginchar** rounds these dimensions to the nearest whole number of pixels; hence w, h, and d will always be integers.

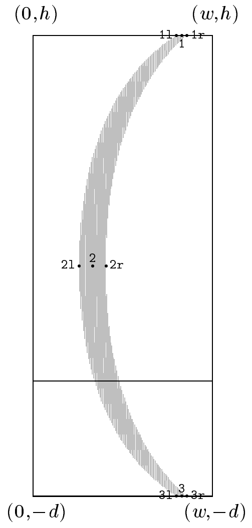
METAFONT's pixels are like squares on graph paper, with pixel boundaries at points with integer coordinates. The left edge of the type box lies on the line $x = 0$, and the right edge lies on the line $x = w$; we have $y = h$ on the top edge and $y = -d$ on the bottom edge. There are w pixels in each row and $h + d$ in each column, so there are exactly $wh + wd$ pixels inside the type box.

Since w, h, and d are integers, they probably do not exactly match the box dimensions that are assumed by device-independent typesetting systems like TEX. Some characters will be a fraction of a pixel too wide; others will be a fraction of a pixel too narrow. However, it's still possible to obtain satisfactory results if the pixel boxes are stacked together based on their w values and if the accumulated error is removed in the spaces between words, provided that the box positions do not drift too far away from their true device-independent locations. A designer should strive to obtain letterforms that work well together when they are placed together in boxes that are an integer number of pixels wide.

You might not like the value of w that **beginchar** computes by rounding the device-independent width to the nearest pixel boundary. For example, you might want to make the letter 'm' one pixel wider, at certain resolutions, so that its three stems are equally spaced or so that it will go better with your 'n'. In such a case you can assign a new value to w, at any time between **beginchar** and **endchar**. This new value will not affect the device-independent box width assumed by TEX, but it should be respected by the software that typesets `dvi` files using your font.

Here's an example of a character that has nonzero width, height, and depth; it's the left parenthesis in Computer Modern fonts like `cmr10`. Computer Modern typefaces are generated by METAFONT programs that involve lots of parameters, so this example also illustrates the principles of "meta-design": Many different varieties of left parentheses can be drawn by this one program. But let's focus our attention first on the comparatively simple way in which the box dimensions are established and used, before looking into the details of how a meta-parenthesis has actually been specified.

```
"Left parenthesis";
numeric $ht^\#, dp^\#$;
$ht^\# = body\_height^\#$; $.5[ht^\#, -dp^\#] = axis^\#$;
beginchar ("(", $7u^\#, ht^\#, dp^\#$);
italcorr $ht^\# * slant - .5u^\#$;
pickup $fine.nib$;
$penpos_1(hair - fine, 0)$;
$penpos_2(.75[thin, thick] - fine, 0)$;
$penpos_3(hair - fine, 0)$;
$rt\, x_{1r} = rt\, x_{3r} = w - u$; $lft\, x_{2l} = x_1 - 4u$;
$top\, y_1 = h$; $y_2 = .5[y_1, y_3] = axis$;
filldraw $z_{1l}\{(z_{2l} - z_{1l})$ xscaled 3$\} \ldots z_{2l}$
      $\ldots \{(z_{3l} - z_{2l})$ xscaled 3$\} z_{3l}$
      -- $z_{3r}\{(z_{2r} - z_{3r})$ xscaled 3$\} \ldots z_{2r}$
      $\ldots \{(z_{1r} - z_{2r})$ xscaled 3$\} z_{1r}$ -- cycle;
penlabels(1, 2, 3); endchar;
```



The width of this left parenthesis is $7u^\#$, where $u^\#$ is an ad hoc parameter that figures in all the widths of the Computer Modern characters. The height and depth have been calculated in such a way that the top and bottom of the bounding box are equally distant from an imaginary line called the *axis*, which is important in mathematical typesetting. (For example, TEX puts the bar line at the axis in fractions like $\frac{1}{2}$; many symbols like '+' and '=', as well as parentheses, are centered on the axis line.) Our example program puts the axis midway between the top and bottom of the type by saying that '$.5[ht^\#, -dp^\#] = axis^\#$'. We also place the top at position '$ht^\# = body\_height^\#$'; here $body\_height^\#$ is the height of the tallest characters in the entire typeface. It turns out that $body\_height^\#$ is exactly $7.5pt^\#$ in `cmr10`, and $axis^\# = 2.5pt^\#$; hence $dp^\# = 2.5pt^\#$, and the parenthesis is exactly 10 pt tall.

The program for '(' uses a **filldraw** command, which we haven't seen before in this book; it's basically a combination of **fill** and **draw**, where the filling is done with the currently-picked-up pen. Some of the Computer Modern fonts have characters with "soft" edges while others have "crisp" edges; the difference is due to the pen that is used to **filldraw** the shapes. This pen is a circle whose