needs to `input` this file to have access to the data. Fortunately, Metafont does possess the ability to read text files.

```
\newwrite\out=file.mf
  ⋮
\write\out{z1=(w, h); %Text for Meta-
font!}
```

Figure 1: *TEX talks to* Metafont.

Going the other way, from Metafont to TEX, is somewhat problematical. One way I have used is to have Metafont create a dummy font and encode numerical information as the numeric value of a kern between `char0` (say) and `char1`.

Let's make this more precise. Suppose the result of a Metafont calculation is -6.18 pt. Then the Metafont statement

```
ligtable 0:  1 kern -6.18pt#;
```

(don't forget the sharp symbol!) records this value in the `tfm` file for this dummy font as the kern between `char0` and `char1`. Here's how TEX can retrieve this value.

```
\font\foo=dummy \newdimen\result
\setbox0=\hbox{\foo\char0 \char1}
           \result=\wd0
\setbox0=\hbox{\foo\char0{}\char1}
           \advance\result by-wd0
```

Now, `\result` holds the value `-6.18 pt`. In practice, additional hackery can strip the '`pt`' away leaving a pure numeric value. By the way, TEX boxes can have negative widths, so it *is* possible to store negative quantities this way.

MF:

```
ligtable 0:  1 kern -6.18pt#;
```

TEX:

```
\font\foo=dummy \newdimen\result

\setbox0=\hbox{\foo\char0 \char1}
           \result=\wd0
\setbox0=\hbox{\foo\char0{}\char1}
           \advance\result by-wd0
```

Figure 2: Metafont *talks to TEX.*

In the old Metafont, we were restricted to 256 kern pairs per font, and it's easy to use these up in this kind of scheme. In the new Metafont, each font may have up to 32k kern pairs. It is hardly likely that we will run out now, so I recommend that you upgrade your TEX and Metafont if you haven't done so already.

4 Labelling Figures

What would labelling a figure involve? At the time Metafont creates the figure, it should take note of the positions of each label. Then, it should pass that information to TEX.

1. Create the figure.
2. Identify the anchor points.
3. Store the coordinates of the labels.
4. Have TEX typeset the figures in the usual way.
```
\font\fig=figfont
 ...
{\fig\char0}
 ...
```

Figure 3: *How to label figures using TEX and* Metafont.

More precisely, here's the sequence we must follow.

1. Create the figure with Metafont as a character in a special figure font.
2. Identify the points in the figure which will be the anchors for the labels. In Metafont we use notation z_1 or `z1` to refer to key points used for drawing; we might use the notation lab_1 or `lab1` for these anchor points. The tag "`lab`" refers to "labels."
3. Metafont will store the coordinates of the lab_i as kerns between `char0` and successive characters of the figure font. The components of each of the lab_i measures the horizontal and vertical offsets from the reference point (origin) of the figure.
4. TEX typesets the figures in the usual way:
```
\font\fig=figfont
...
{\fig\char0}
```
A set of TEX macros allows placing the labels using the trick mentioned above. One might say `\point{This is a label}` and so on.

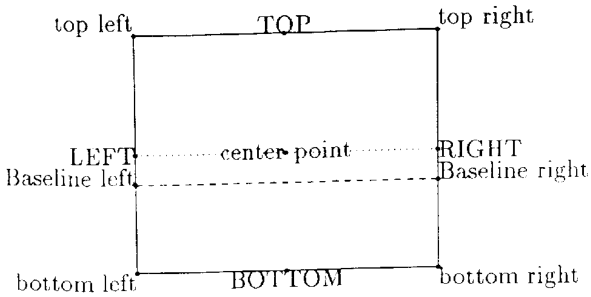

Figure 4: *References for all pointing commands*

These macros are enclosed in two sets of macros, the so-called `labtex` macros (one for TEX, and one for Metafont). `labtex.tex` imagines that each label is enclosed in a rectangular box. Figure 4 shows the eleven different pointing commands for positioning the label box with respect to the label point in the Metafont diagram.

`\up3pt`	`\down`	`\right`	`\left`
`\extradx`	`\extrady`	`\everylabel`	
`\tlpoint`	`\ltpoint`		
`\tpoint`			
`\rtpoint`	`\trpoint`		
`\rpoint`			
`\Brpoint`	`\rBpoint`		
`\brpoint`	`\rbpoint`		
`\bpoint`			
`\blpoint`	`\lbpoint`		
`\Blpoint`	`\lBpoint`		
`\lpoint`			
`\cpoint`			

Figure 5: `labtex` *pointing commands and "tinkering" commands.*

This figure also shows how tight the labels are in the absence of some tinkering. We can tinker either by hand or semi-automatically. To adjust a label placement by hand, we can use the commands `\up`, `\down`, `\right`, or `\left` to move the label by a certain amount. For example, instead of saying `\cpoint{A label.}`, we can say

```
\cpoint{\down3pt\right2pt A label.}
```

We can also adjust all labels automatically by means of three parameters which work in the background. `\extradx` and `\extrady` place extra space to the right/left or above/below the label text. A token `\everylabel` (akin to `\everypar`, `\everyhbox`, `\everymath`, and so on) allows some global adjustments to the labels themselves. For example, if you set

```
\everylabel=\ninepoint
```

you would get smaller labels automatically.

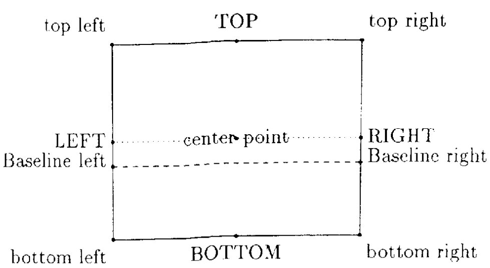

Figure 6: *Automatic tinkering with label placement.*

Figure 6 show how these labels look using these automatic enhancements. Both `\extradx` and `\extrady` have been set to 3 pt. By the way, although this particular figure could have been produced with TEX (after all, it consists only of horizontal and vertical line segments, which TEX can draw), it wasn't. In particular, observe that Metafont can also draw dotted and dashed lines.

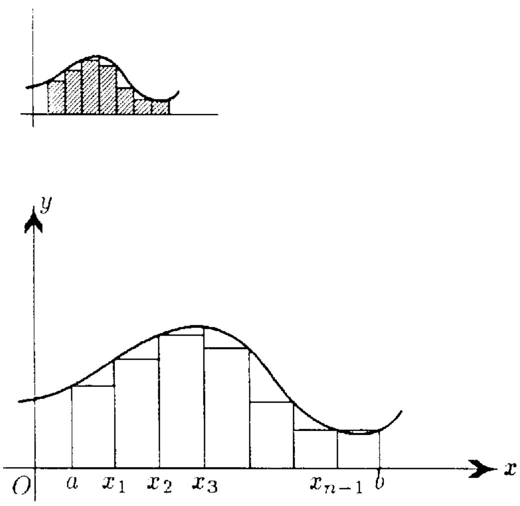

Figure 7: *Developing the theory of integrals.*

One of my hidden agendas in this project is to remind people of the power and utility of Metafont as a figure and diagram generator. With Metafont, you can do all the things you expect in a figure—and more, now that it is possible to anchor labels in the diagram.

Less well known is Metafont's ability to add shading to a drawing. Figure 7 displays a figure that was copied from the book I learned calculus from. (Published by Addison-Wesley, Knuth's publisher, it seems to use Monotype Modern 8a, the inspiration for Computer Modern, as its text face.) There are two versions of this figure for a reason. The larger version has labels but no shading—standard Metafont cannot handle shading for such large pictures. The smaller version was the largest I could generate with the shading, but alas, there was no room for labels in this case.

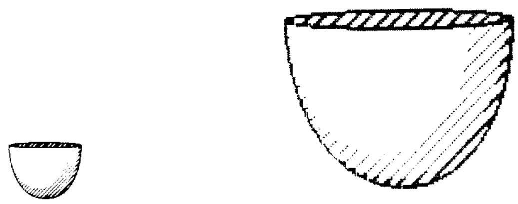

Figure 8: *Special effects with* Metafont.

Figure 8 shows one way that Metafont can draw a three-dimensional bowl. (If it doesn't look sufficiently 3-D, blame the author and not Metafont.) It's fun generating shading, but you quickly run up against the memory capacity of normal Metafont, as I did in figure 7. If the user community vocalizes a need for this kind of Metafont work, I hope the implementer community will respond with versions of "Really Big Metafont."