Notes:

1. Charles Karney states, "...I haven't fully explored the parameter space. If anyone knows of a better (or 'authorized') solution, I'd appreciate hearing about it."
   [Karney%PPC.MFEnet@LLL-MFE.Arpa]
2. John Gourlay has diagnosed an unexpected modification to the pen path as *blacker* increases, causing the diameter of such letters as "o" to decrease; the details are discussed in his article in TUGboat **8**#2, page 20. The parameter values given here are a compromise, allowing most characters to keep their original sizes, although the value of *blacker* "is not quite enough to compensate for the thinning inherent in the printer." There is still "an inconsistency in the weights of characters. Nevertheless, [Gourlay] feel[s] that this set of parameters is considerably better than the ones that result from the 'conjectural' parameters, and also better than the 'am' fonts they replace."
   Gourlay.Ohio-State@csnet-relay
3. Charles LaBrec's comments: "I have twiddled the parameters a bit, and this seems to produce good 12 point cm fonts. I am a bit unsure because changing *blacker*, *fillin*, or *o_correction* seem to make no difference for quite a large range of values. I can't remember exactly, but you will get the same results as [these] for $.4 < blacker < .9$, $-.8 < fillin < -.1$, and $0 < o\_correction < .7$. But this probably makes a good starting point."
   [crl@newton.physics.purdue.edu]
   [Editor's note: The value given in TUGboat 8#1 for decln *fillin* should have been −.2, not +.2.]
4. Stan Osborne: "The decln mode [Mr. LaBrec] suggested did not *fillin* correctly and was too black for the smaller point sizes. His choice of settings produces small sized fonts that are much blacker than the small cmr's found in the cmr book (Vol E).... I found the [above] values of *blacker* and *fillin* to produce readable small fonts for an LN03.... These values were not carefully tested for larger point sizes. (I stopped experimenting when I got something I liked and I had verified that larger sizes were also usable.) [...!ucbvax!dual!dbi!stan]
5. Janene Winter has found these settings "to be optimal for the IBM printers". This information was transmitted by Dean Guenther along with his site report (TUGboat **8**#2, page 10).
6. Matthias Feyerabend: "Fonts tested are CMR5, CMR10, CMR12 and CMSSI17 for a full range of settings for *blacker* and *fillin*."
7. Doug Henderson: Preliminary reasonable looking fonts produced for these three printers. Since the Linotype typesetters are fairly expensive I cannot do extensive testing. Anyone have one and want to donate some use for testing? Please let me know.
   [dlatex@cmsa.berkeley.edu]

Halftone Output from TEX

Adrian F. Clark

Don Knuth's article in TUGboat volume 8 number 2 described the development of a number of fonts which allow halftone output—pictures—to be incorporated into TEX documents. This article chronicles the author's experiments into halftone production on a particular computer/laser printer combination, VAX/VMS and the LN03. It is important to understand that the picture is actually *typeset*, not just inserted into the final output by some printer-specific `\special` command; the following results can, in principle, be achieved on *any* output device using a perfectly normal implementation of TEX.

In the image processing field, where the author works, technical reports are invariably crammed with halftone output. The conventional method of reproducing pictures is photographically. This is slow and expensive, particularly for internal reports with small distributions. Moreover, unless great care is taken over the photographs—using a flat-screen CRT, calibrating films, standardising the processing, and so on—much of the visual impact can be lost. Hence, the possibility of incorporating imagery into TEX document without recourse to a dark room is very attractive.

A great deal of work has been carried out into the properties of the human eye. One result is that the eye is only really capable of distinguishing about 64 grey levels, although it is very good at detecting boundaries between regions of slightly differing grey level (see, for example, "*Digital Image Processing*" by R. C. Gonzalez and P. Wintz, published by Addison-Wesley in 1977). Another result is that the eye is much more sensitive to boundaries in dark regions than in light regions.

The halftone font used here is more or less the same as the 'double-dot' font described by Knuth. It has some 65 different grey levels, represented by the ASCII characters '0' (white) to 'p' (black). In principle, all one needs to do is to convert the grey levels of the individual pixels ("picture elements") of an image to the appropriate characters of the halftone font and sprinkle in a few TeX commands to ensure that the lines of the image are lined up in the output.

The only minor complication is that this sequence of characters includes '\', '^' and '_', which have special meanings to TeX. These must be treated specially. Knuth's approach was to delimit the picture data between macros, `\beginhalftone` and `\endhalftone`, which disable the special characters in a similar way to the 'verbatim' macros in Appendix E of *"The TeXbook"*. The approach developed by the author is much less elegant and builds larger disc files, but does not require special-purpose macros. Each line of the image is built up as a single `\hbox`. These lines are stacked into a `\vbox`, with the inter-line skip turned off. Finally, the `\vbox` is enclosed in another `\hbox`, which makes it easier to handle the picture in constructs such as `\centerline`. The scheme can be summarised as:

```
\hbox{ \vbox{ \halftone
   \offinterlineskip
   \hbox{...}
      ...
   \hbox{...}
}}
```

The `\halftone` command is used to select the halftone font, which must have been loaded with a command such as

```
\font\halftone=hf300
```

assuming the TFM file is called `HF300.TFM`.

A FORTRAN `SUBROUTINE`, `TEXPIC`, was written to output images to files in this format. The image is represented as a `REAL` array dimensioned as `(M,N)`, where `M` is the number of pixels per line and `N` the number of lines. (The use of a `REAL` array to hold data which are usually 8-bit may seem a little strange, but this representation has many advantages—for example, when Fourier transforming an image.)

Since we would normally like our pictures to have the best contrast, `TEXPIC` scans through the image to find its minimum and maximum, then scales the output to make full use of the grey levels in the halftone font. For most purposes, a single

```
      CALL TEXPIC( PIC, M, N, FN )
```

is sufficient. `FN` is a `CHARACTER` variable or quoted string holding the output filename.

Of course, there are occasions when we would like to compare pictures, so fixing the contrast is sometimes desirable; hence, `TEXPIC` has associated routines to fix the range of intensities (`ZRANGE`) and re-select automatic intensity scaling (`ZAUTO`), which must be invoked before `TEXPIC` to have an effect. Similarly, `TEXPIC` can plot negative pictures as well as positive ones: `DONEG` tells it to output subsequent pictures as negatives and `DOPOS` returns it to the default state.

Inserting the picture into a document prepared with plain TeX is quite simple, using commands to generate a 'float', such as

```
\midinsert
   \centerline{\input picture}
\endinsert
```

for a picture in the file `PICTURE.TEX`. To draw a border around the picture, as for the examples presented here, one would define a macro `\border`

```
\def\border#1{\vbox{\hrule\hbox{
   \vrule\kern3pt\vbox{\kern3pt#1
   \kern3pt}\kern3pt\vrule}\hrule}}
```

The picture would then be set with

```
\centerline{\border{\input picture}}
```

The procedure with LaTeX is somewhat different. The most sensible approach is to use the `figure` environment (*not* the `picture` environment)

```
\begin{figure}
   \centering
   \mbox{\input picture\relax}
   \caption{...}
\end{figure}
```

This generates a 'floating' figure, which usually surfaces at the top of the next page of output. The `\relax` following the filename in the `\mbox` command ensures that LaTeX knows where the filename ends. To draw a border around the picture, replace the `\mbox` with a `\fbox`.

It is traditional to test out new image processing techniques on the 'girl' picture from the image database of the University of Southern California's Signal and Image Processing Institute. She is shown in Fig. 1 (64×64 pixels). The output was plotted on a standard `LN03` laser printer using version 10 of Flavio Rose's `DVI2LN3`. For those unfamiliar with the `LN03`, it is a 300 dpi, white-writing laser printer

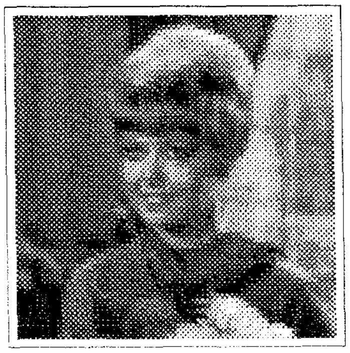

Figure 1: The Ubiquitous 'Girl' Image

based a Ricoh mechanism, supporting the downloading of fonts into on-board and plug-in RAM cartridges. The quality of Fig. 1 may not appear to be particularly good, but this is due to the comparatively low spatial resolution of the image data: approximately 256 × 256 pixels are needed to give a visually satisfying result—as we shall see.

Unfortunately, the standard `LN03` will not output images of much greater than 64×64 pixels: if one tries to do so, it generates "band too complex" errors and produces broad white bands in the output. The actual cause of this is not known; however, it seems to be because the `LN03` buffers plotting commands internally rather than writing their resulting glyphs into a bitmap. When the print operation actually starts, the driving microprocessor cannot translate the commands sufficiently quickly.

However, the `LN03+` device (a field-installable hardware and firmware upgrade) has a full-page bitmap, and is quite capable of printing off large pictures. (However, a little care is needed in setting up the terminal line to which the printer is attached.)

There is another problem in producing these large pictures, and it concerns TeX itself. Since TeX was designed for typesetting text rather than pictures, its memory capacity is too small. Increasing the size of the memory (i.e., `mem_size`) is obviously feasible, at least on VAXen, but there is a snag: TeX was written to use 16-bit integers for subscripts into the memory arrays. However, the change file mechanism of WEB and the careful way in which TeX was written makes the conversion of 16-bit integers to 32-bit integers quite straightforward. (It is also necessary to disable some of TeX's initial consistency checking.)

When the author did this, producing a "big TeX", he found that the 16-bit and 32-bit versions of TeX were identical in almost every respect. The executable file was a few percent bigger, probably due to the increased memory space rather than the different integer representation. Likewise, the string pool and format files were slightly larger. However, there is *no* perceivable impact on execution times. (In fact, the author replaced the 16-bit version with big TeX without telling users—and no-one noticed any difference!)

This may seem a little surprising at first, but an examination of the (pseudo-) assembler generated by the PASCAL compiler provides the answer. The machine code generated for variables declared as `0..65535` (or, indeed, `0..255`) is *identical* to that for, say, `0..262144`: 32-bit integers are used in all cases. (This does, of course, not apply to **packed arrays**.) Moreover, TeX is very frugal in the way it handles its memory arrays, always re-using the same region if possible; this keeps the page fault rate low. Since the VAX initialises all memory to be 'demand-zero' when a program is loaded, there is no real increase in the system overhead due to unused regions of TeX's memory.

The version of TeX at the author's site has a large enough memory capacity for four 256×256 pictures (or one 512 × 512 picture!) in addition to the usual text, fonts and macro definitions. This allows users to put a few images into floating figures, as described above, without overflowing TeX's memory. For example, a 256 × 256 picture is shown in Fig. 2.

Indeed, to a certain extent, the physical size of a picture on the printed page determines the maximum number of pixels which can be plotted. Images of 512 × 512 pixels are more or less standard in the image processing community, while satellite images used in remote sensing applications have several thousand pixels on a side! Hence, if the image size exceeds a proscribed maximum (256 pixels, say), `TEXPIC` must *interpolate* between pixels to reduce the size of an image. Another associated `SUBROUTINE`, `TEXMAX`, is used to tell `TEXPIC` the maximum number of pixels which can be output. If the `M` dimension of an image exceeds this value, the image is interpolated down to this plottable maximum number of pixels.

There are many ways to perform the interpolation. The theoretical optimum is to use a $\sin x/x$ interpolation function (usually achieved via Fourier transformation), but this is slow. Cubic or linear interpolators tend to be used in practise. Recognising that TeX output of a reduced 4000 × 4000 pixel image will inevitably be inaccurate, `TEXPIC` uses a linear interpolation scheme. However, since linear interpolators usually blur edges (a particularly undesirable effect), it attempts to reduce the blur by using a *context-sensitive* interpolator. This in-

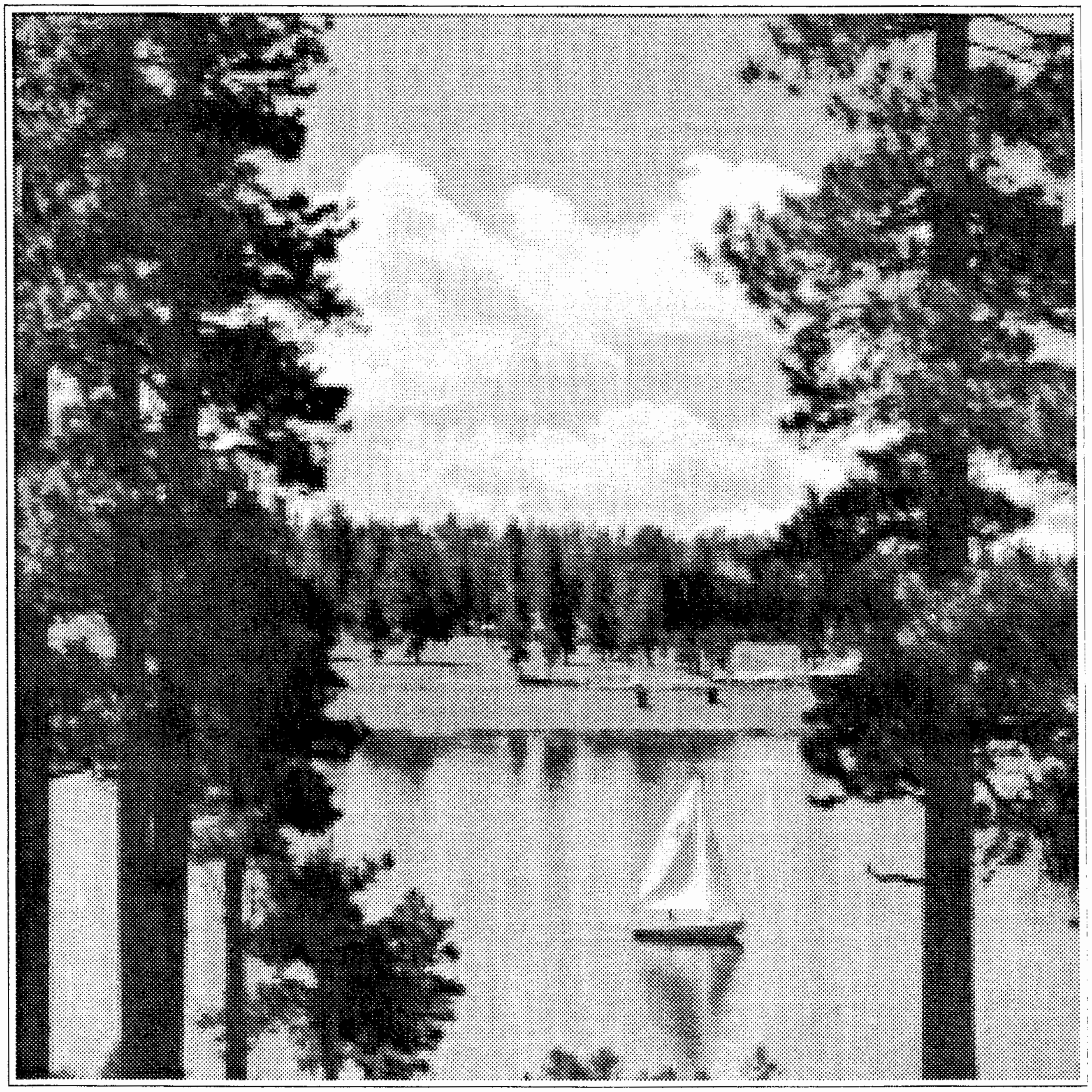

Figure 2: A 256×256 Lake Scene

terpolates between two sets of triplets of pixels at right angles and selects the value on the line with maximum gradient. For example, Fig. 3 is a 200×200 pixel image, reduced from a 512×512 image in this way.

All the software described here is available. TEXPIC and supporting routines exist in both standard FORTRAN and VAX FORTRAN; the VAX version does clever things with filenames and channel numbers. The big TEX change file incorporates an editor interface and automatic determination of batch or interactive use (see TUGboat vol. 8 no. 2 p. 177). It is, of course, specific to VMS, but may be useful for people making similar enhancements on other machines.

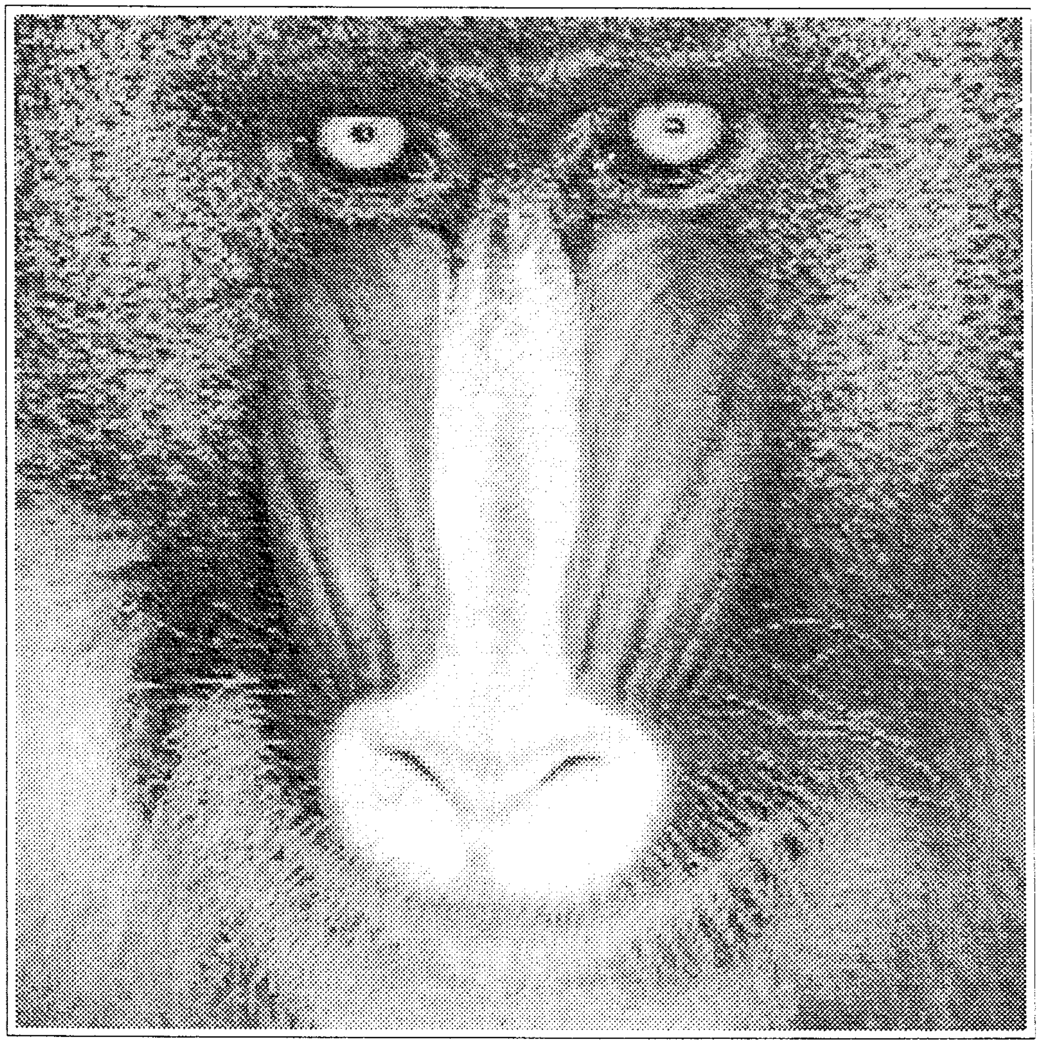

Figure 3: Mandrill Image, Reduced to 200×200Pixels from 512×512 Pixels