On the Inclusion and Placement of Documentation Graphics in Computer Typesetting

Chris Corbett

Dept. of Electrical Engineering Science, Essex University, Wivenhoe Park, Colchester, Essex CO4 3SQ, UK

Ian H. Witten

Man-Machine Systems Laboratory, Department of Computer Science, The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

Computer typesetting systems can produce high quality printed pages suitable for the production of books, journals and newspapers in final form. Little effort to date has been applied to the problem of including and placing arbitrary graphic figures within processed text. In this paper we describe a system, constructed around existing typesetting software, which allows figure definition and placement. Key features of the system are a wide range of figure sources; figure preview facilities; figures-only output and compatibility with non-graphic printing devices, where figures are plotted separately; support of a figure language for in-line figure definition; and figure placement algorithms which are changeable at the user level.

INTRODUCTION

The use of computers for text layout is certainly one of their fastest-growing application areas. Substantial economies in the production of books, journals and newspapers can be achieved if some of the traditional steps in publishing are short-circuited. Computers can help with copy editing, composing, proof-reading, correcting, page make-up, index preparation, distribution, and archiving. All of these tasks are being addressed both by the front-office technology of word processors and office information systems, and by larger and more powerful document preparation systems such as *troff*,[1] TEX,[2] and Scribe.[3]

But one important area has received relatively little attention: the preparation of illustrations and their placement in documents. This is no doubt because computer peripherals which provide high-quality printing of both text and figures have only recently become widely available. Impact printers can produce acceptable text but not graphics; plotters can produce graphics but not text (certainly not in quantity). Isolated attempts have been made to incorporate artwork into documents. The Multics COMPOSE package[4] has a facility which allows crude artwork to be constructed on a VDU and printed on a letter-quality printer. Recently, a language called PIC has been devised[5] which allows a picture definition to be contained within the document's text, expanded by an interpreter which constitutes a preprocessor to the text-formatting operation, and printed by a phototypesetter. However, it does not permit the inclusion of arbitrary half-tone images and graphics from sources outside PIC—this is probably because the phototypesetter is a conventional one with optical images of characters. Document preparation systems such as TEX are designed for use with soft-font output devices but do not have any significant graphical capability.

Real-world documents usually contain figures from many sources, amongst which are hand-drawn artwork, graphical output from computer programs, grey-scale photographic originals, pre-screened images and figures which include formatted text. Once the text is available and the figures have been obtained, the conventional compositor goes through a figure-placement exercise to determine the location of figures on the page which corresponds best with their callouts in the text. Then, text and figures are merged (using a paste-up technique), and the document is printed.

This paper describes a system based upon *troff*[1] and other UNIX utilities which can include documentation graphics from many different sources and place them on the page according to user-defined criteria. Pictures can be specified in various ways in the text. They are sized automatically and the measurements are given to the figure-placement algorithm, which determines what page they should occupy and whereabouts on it they should go. As far as the text-formatter is concerned, such areas simply appear blank and text is placed around them. Then a final merging operation is performed by a raster preprocessor just prior to printing. This may involve expansion of a picture definition into raster form, or the invocation of a transform on a grey-scale image to produce black and white pixels. The complete document is printed on a raster device. To avoid the massive storage requirements that could otherwise occur, the preprocessing is done on the fly as the document is printed.

PICTURE SOURCES

Figures defined within the text

The PIC language for drawing block diagrams has been re-implemented in an expanded processor called *fig*. (Current work is centred on improving the picture-drawing facilities it provides, in particular by the addition of structured sub-picture definitions and graphic transforms.) This allows a figure specification to be included in the text, preceded by a control command which invokes PIC. The PIC interpreter is then applied to the document as a filter before the text-formatting operation (*troff*). The figure definition is extracted and processed into a plot file which has a standard, system-wide plot

CCC-0010-4620/82/0025-0272 $03.00

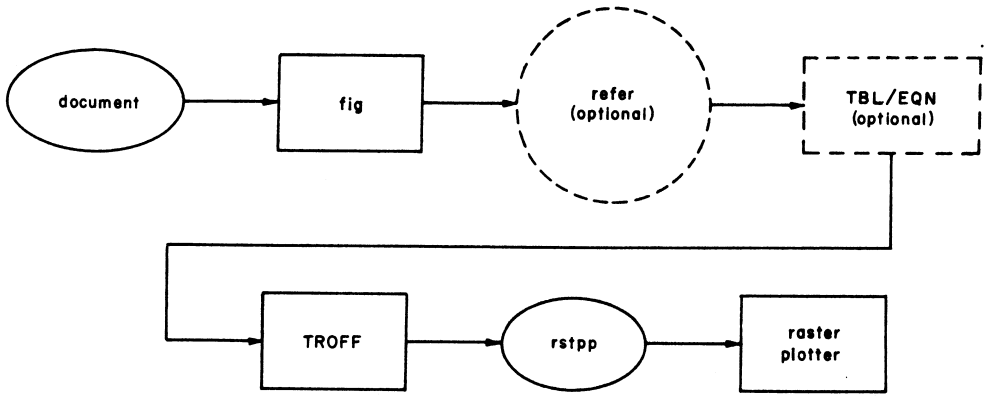

Figure 1 Example of a PIC picture definition

Figure 1.

format. *Fig* selects a file name, sizes the figure, and substitutes an appropriate control command containing size and file name into the output stream. This output stream is then processed by *troff*. The whole operation appears to the user almost exactly as defined by Kernighan (1981).

As an example of the use of PIC, Fig. 1 was created with the input

```
.FI
ellipse wid 0.9i "document"
arrow
box "fig"
arrow
circle rad 0.5i dashed "refer" "(optional)"
arrow
box dashed wid 0.9i "TBL/EQN" "(optional)"
line down
line to 1st ellipse + (ellipsewid/2,-boxht/2-lineht)
line down
arrow right
box "TROFF"
arrow
ellipse "rstpp"
arrow
box "raster" "plotter"
.FE "Figure 1 Example of a PIC picture definition"
```

(taken from Ref. 5). '.FI' and '.FE' are control codes to invoke and terminate PIC, and the whole excerpt appears in the document just after the words 'Figure 1' at the beginning of this paragraph. It is expanded into a graphics specification by *fig*, placed by the figure-placement macros which the user can specify to *troff*, and merged with the text by the raster postprocessor. The string argument to '.FE' provides the figure caption.

Output from graphics packages

The use of a standard plot file format greatly enhances the utility of the system. Standard graphics packages can produce picture definitions, possibly interactively, in this format. For example, we have a set of graphics routines callable from the C programming language,[6] which can utilize real (cm) or user-defined units of measure for coordinates.

The GROPER language for recursive picture definition[7] has recently been ported to our system: its output is in the standard plot format and so can be incorporated into documents directly. One simply specifies the file name in a call to *fig*, which extracts the size information from the file and passes it and the file name into the output stream to be processed by *troff*. Figure 2 shows the picture created by the GROPER commands:

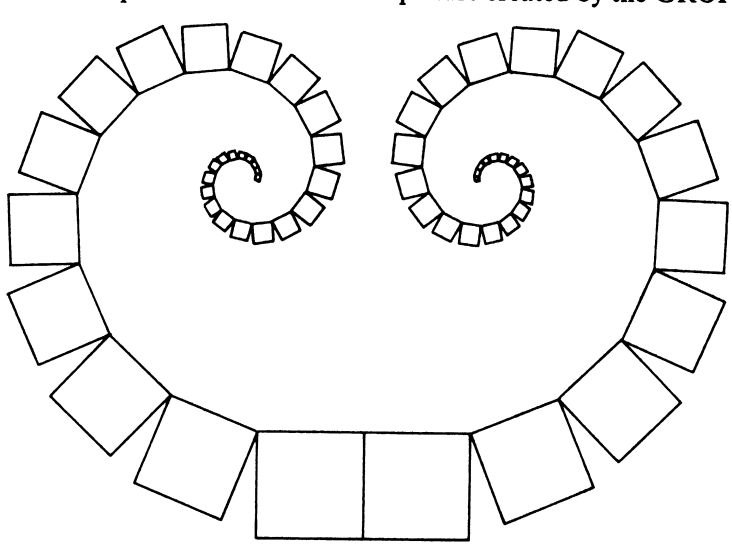

Figure 2 Example of GROPER output

Figure 2.

```
add line sq add sq sq ori 1, 0 rot 90 lim 4
  add sq sp add sp sp ori 1, 0 rot −22 mag 0.9 lim 30
  add sp pik
  add sp pik mag −1,1
  dev plot 5
plot
```

placed and merged into the text by *fig*, using the command

```
.FI plot groper-output
.FE 'Figure 2 Example of GROPER output'
```

placed in the text.

A useful Unix system program called *graph* exists which takes a list of coordinates and produces a standard plot file. Its output can therefore be used to generate figures in text. Figure 3 shows an example, created by the command

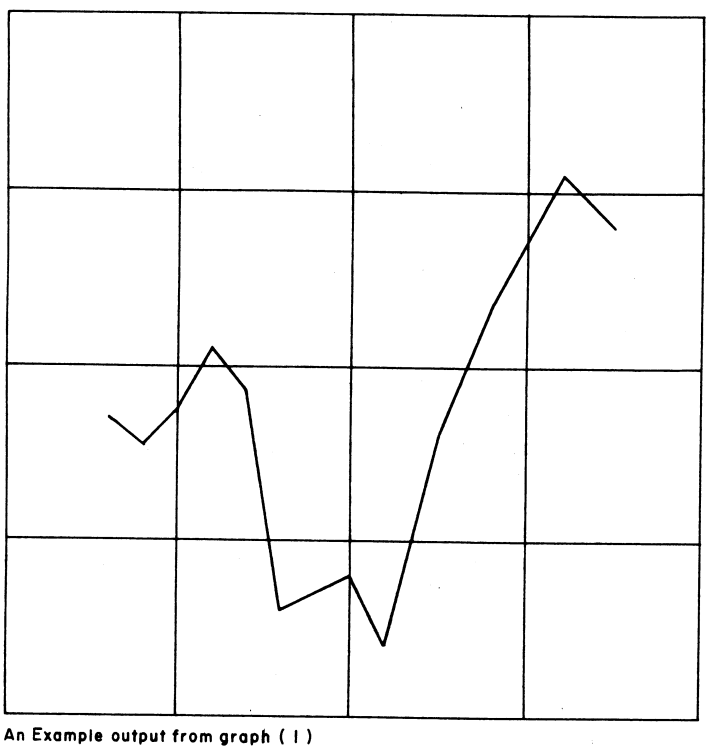

Figure 3 Example of graph output

Figure 3.

```
.FI plot graph-output
.FE 'Figure 3 Example of graph output'
```

Grey-scale images

A file of grey-scale intensity values on a matrix of x, y grid points must be converted to a black-and-white raster image before it can be printed. Several techniques for conversion have been proposed.[8] A new method has been developed recently[9] which uses a recursive space-filling curve as a contour along which the image is converted into incremental representation by the DDA technique.[10] Moire effects are totally eliminated by the use of a sufficiently convoluted path, and there exist theoretical reasons for choosing Peano curves as the discretization contour. This and other methods are presently being evaluated. To assist with this, a rather flexible mechanism has been provided within *fig* to allow escape to an arbitrary Unix program, whose output is directed into a raster file whose name is chosen by *fig*. Figure 4 was generated by the command

```
.FI halftone image_file | peano-flags
.FE 'Figure 4 Example of halftone output',
```

where image_file contains the original grey-scale image, 'peano' is the name of the peano-conversion program, and 'flags' are arguments to it which specify options. The half-tone raster size in Fig. 4 has been chosen to be rather larger than the resolution of the output device, to make the dot structure easily visible.

Two operations that are important in real-life image manipulation are clipping and magnification of photographic pictures. Neither of these have been implemented at present because we are not working in a commercial environment. A syntax like

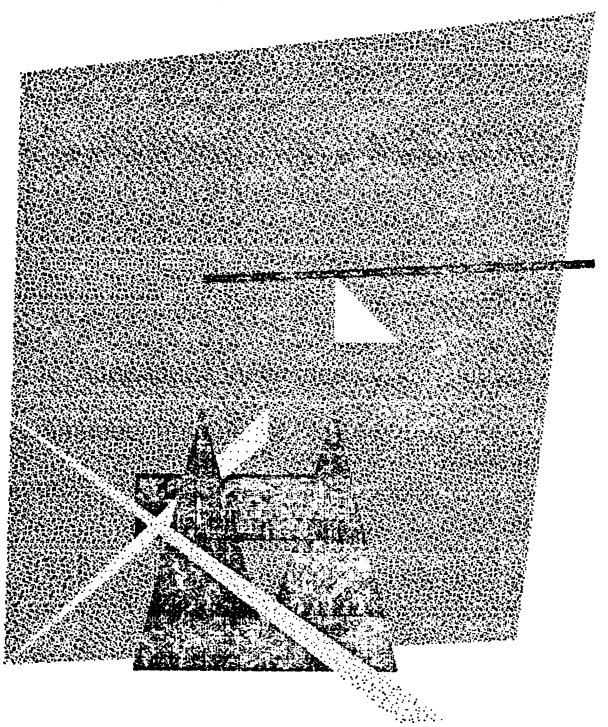

Figure 4 Example of halftone output

Figure 4.

```
.FI halftone image_file | clip -flags |
                scale -flags | peano -flags
```

is envisaged for these operations.

Pre-screened images

It sometimes happens that a publisher has to deal with pre-screened half-tone images. This is common for advertisements in newspapers and magazines, where an illustration may be prepared by an advertising agency, screened into a half-tone image, shown to the customer for approval, and then presented to the publisher for printing. In a computer-based system the pre-screened image corresponds to a raw raster file at the resolution of the output device. Hence we have included a facility in *fig* for such an image to be specified. The file format

contains the horizontal and vertical measurements of the image, followed by the bits that comprise it. It is included as a figure by the command

.FI raster image_file.

Figures which include processed text

One area which creates considerable difficulty is the interaction between figures and the text-formatting operation. Normal typeset characters are defined within our *troff* system as raster dot-patterns for each character, in each font, in each size. It is computationally infeasible to perform certain transformations (such as rotation) upon characters stored in this form, because the transformation must be done on each of the many dots that comprise the character.

In order to allow arbitrary scaled and rotated characters to be included in figures, a facility has been incorporated into *fig* for drawing Hershey fonts.[11] These are specified as line segments and hence are easy to transform. The PIC language as currently defined does not permit text rotations to be specified. However, we intend to make substantial enhancements to it which will include transformations, and envisage offering the user a choice of normal typesetter dot-pattern fonts or line-segment ones, with the proviso that only the latter may be transformed. Figure 5 shows a figure which uses exotic, rotated characters: it was drawn using the above-mentioned graphics routines.[6]

Sizing of text is another interesting issue. The current PIC cannot judge the size of a block of text, and does not allow text filling or margin adjustment within pictures. Hence it cannot, for example, create a box just large enough to enclose a piece of text, nor can it break a line just enough to leave space for an annotation. To provide these facilities, the text-formatting operation must be callable from within the picture language. Then a block of text could be formatted with certain characteristics (like type size and line length), measured, and placed in a figure with its measurements made available to the figure processor for subsequent use. It may be that a coroutine control structure for the figure and text processor is most elegant. These possibilities are presently being explored.

FIGURE PLACEMENT

Once a figure has been sized and converted into either a plot file or a raster file, *fig* inserts a macro call into the output stream which invokes the figure-placement algorithm when processed by *troff*. (Actually, *fig* simply copies the .FE line with the caption, if any.) Among the aesthetic criteria which are commonly used for figure placement are (i) the figure should be close to its callout in the main text, (ii) figures must appear in the order in which they occur in the input document, (iii) figures should only be placed at the beginning or end of a page, (iv) a page should not have figures both at the top and bottom, with intervening text, (v) several figures can appear consecutively on the page and (vi) the caption of a figure placed at the end of a page should be aligned with the lower boundary of the text area. With a basically one-pass document processor, it is not feasible to perform global optimization of figure positions such as is described by Bammel.[12] Hence a figure-placement method is used

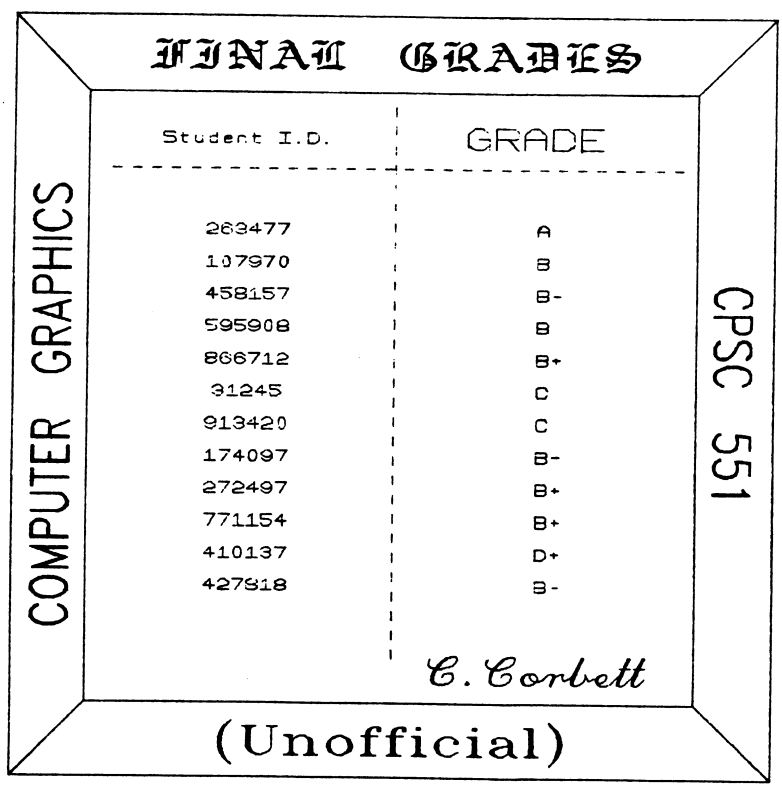

Figure 5. Example with rotated characters produced by graphics package.

which can only site the figure *after* that point in the text where it is specified; it is usually best to place the figure specification just after the first reference to it.

With this proviso, a suite of macro definitions for *troff* which satisfies the above placement criteria has been built[13] and used for some years. It allows many figures to be 'pending' (defined but not yet placed) at once, and sites them as soon as there is an opportunity to do so. Graphic figures are fully compatible with the placement macros, since from the outset it was decided to implement graphic figure sizing and placement with a user-defined troff macro (see next section).

However, the most important point is that such an algorithm can be written and communicated to *troff* at the user level. Sadly, the difficulty involved is abominable, due to poor human engineering of the user interface. Steps are being taken to improve this by defining a high-level block-structured language, based upon an ADA-like syntax, which compiles into the low-level *troff* code. This is intended to enhance the accessibility of the figure-placement process and to give easy control over the whole format of the document.

MERGING PICTURES WITH TYPESET OUTPUT

The basis of figure merging is the ability to build a raster image of the page to be printed. Page composition with figures is in essence a three-dimensional technique. TEX[2] builds pages based on the notion of two-dimensional boxes; box positioning is performed using the properties of *shrink* and *stretch* between adjacent boxes, whether they contain characters or paragraphs. For figure inclusion, we regard a page to be constructed of several layers—a three-dimensional object. Each layer is in itself a two-dimensional slice of the page, containing either text or figures. A layer's position on the page is determined by the x, y coordinate defining its top left corner. Figure 6 illustrates the page make-up process. Starting with a blank page, *troff* will produce the formatted text slice, while *fig* is producing and positioning figures defined in the text. Figure 6 shows the case for two figures being centrally located within the text layer.

Troff produces phototypesetter codes, which are unsuitable for the type of page composition discussed above. A post-processor for *troff* (*rvcat*, developed at the University of California at Berkeley) was available to us. This produces a raster image of the *troff* output for Versatec-like printers. *Rvcat* forms the basis for our figure post-processor, *rstpp*, which performs the page make-up step from the individual raster slices.

The layering technique for page composition works as follows. Textual input with an included figure will take the form of the example below:

```
some text
.FI ⟨figure definition⟩

.FE ⟨figure caption⟩
some more text
```

The syntax for ⟨figure definition⟩ has been described in previous sections. The figure preprocessor *fig* performs several operations when interpreting the '.FI' control request. First, the figure definition statements are used to

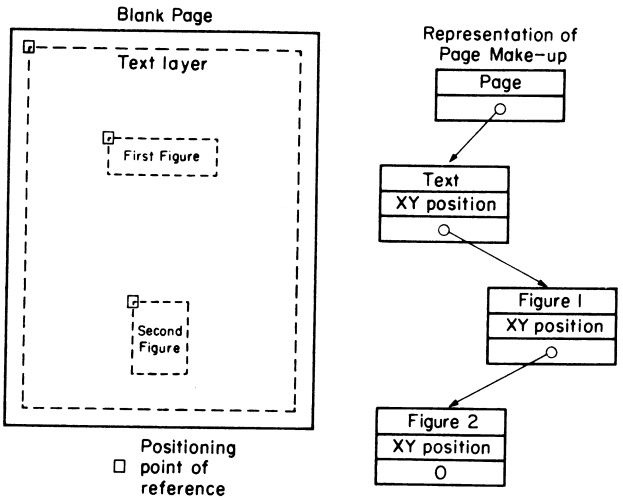

Figure 6. Page structure and its representation.

create a complete picture file describing the figure at one of two levels. The first is Intermediate level 1 code, which is a vector optimized format based on the UNIX plot standard. PIC generates this output, as do the standard graphics packages available under UNIX. The second, Intermediate level 0 code, is a raster format suitable for half-tone or photographic type figures.

The second function in *fig* is to replace the figure definition in the source text by a single line having the following format:

.FI XSIZE YSIZE ⟨filename⟩

XSIZE and YSIZE specify the size of the generated figure, while ⟨filename⟩ is the name of the generated file. If preceded by a '$', the file is in Intermediate level 0 code. (Although this seems ugly, remember that this substituted line is not normally seen by the user.)

The third function is to generate a canned definition of a *troff* macro '.FI' on the first invocation of *fig*. This macro interprets the information in the '.FI' output line discussed above. Currently, the XSIZE and YSIZE information is used to centre the figure on the page, leaving enough white space around it for text. The filename is encoded appropriately so that *troff* will pass it on to the page builder *rstpp*. Allowing figure placement in this way gives the user complete control over the figure placement process—the definition of '.FI' can easily be altered to position the figure in different ways relative to surrounding text.

OTHER SYSTEM CAPABILITIES

A preview facility is almost essential for any hard-copy graphics package. Fortunately the use of the standard plot file format at Intermediate level 1 means that this is already available. Any plot file created by GROPER, PIC, *graph*, or the subroutine library can be viewed by an existing utility called *plot*: a switch on this scales the figure to the maximum size permitted by the output device.

The figures in a document can be processed for viewing without any text being printed. The text file is simply passed through the *fig* processor, and the output—which

normally goes to *troff*—is discarded. This creates a succession of files, one for each figure, in Intermediate level 1 format. These can then be viewed one by one. They could just as easily be plotted on a non-text device such as a plotter, for pasting into a typescript.

Similarly, it is trivial to get the text of a document without the figures on a device which does not have drawing capability. Simply process the figure as normal, specifying the output device! In other words, pass the document through the *fig* preprocessor, then through *troff*, and then to that device. All the illustrations will be sized correctly and space will be left for them. The plot files which are created by *fig* will be ignored by the output device handler, for only the raster device handler knows about the figure-generating process. A mark is left in the top left corner of the white space as a guide to figure positioning for later pasting-up operations.

A complementary requirement is to be able to obtain a galley proof of the figures only. They could of course be drawn one by one, as described above for viewing. However, a switch has been incorporated into *fig* to suppress all textual output, so that the illustrations can be drawn on the raster device by the raster preprocessor as a single operation.

CONCLUSIONS

This paper has described a framework which allows arbitrary figures to be included and placed within processed text. A specific environment was chosen to implement the system, but on the basis of this implementation we regard the following general properties as essential for figure processing:

(i) the user should have complete flexibility of image sources with a standardized syntax for inclusion, and a universal mechanism for placement; (ii) figures, however defined, should be in a format which will allow interactive previewing; (iii) intermediate picture description languages for both raster images and line drawings are necessary; (iv) a textual, in-line, figure definition language like PIC is essential for entering simple images on a VDU; (v) the system must be transparent to output devices which cannot display graphical information.

The wide variety of potential image sources has been discussed above. A glance at almost any technical publication will show that there is a real need for systems which can cope with their diversity. Any scheme which is restricted to only one or two such sources will have serious limitations in practice. The advantage of a uniform syntax for specifying images is obvious. The method described above has the advantage of clearly indicating the type of image in the inclusion command, and transparently invoking the figure placement algorithm.

A preview facility is essential in any hard-copy computer graphics application. The use of existing standards for file formats allows advantage to be taken of ordinary system utilities. Furthermore, any interactive graphics packages can then be used for figure preparation. An obvious extension to the system is to use the presentation-level protocol defined by Telidon[14] as an additional picture source: cheap interactive systems for production of Telidon frames are already becoming available and promise to be a fruitful source of images in the future.

The use of existing standards may, of course, compromise on some desirable features. For example, we would like to see relative vectors and picture subroutines at the intermediate level, for this would improve the compactness of figure files. However, such compromises seem eminently suited to an age where resources for the production of software are scarce.

Similar comments apply to the use of the PIC language. We have capitalized on the fact that PIC is already defined—its re-implementation has proved to be fairly simple (around 3 man-weeks). However, we are far from satisfied with the facilities it offers to the user, and are presently working on the definition of a language which embodies a structured graphical approach to pictures, sub-pictures, and transformations.

Acknowledgments

This research was supported by the National Sciences and Engineering Research Council of Canada.

REFERENCES

1. B. W. Kernighan, M. E. Lesk and J. F. Ossanna, Document preparation, *Bell System Technical Journal* **57** (6), 2115–2135 (July/August 1978).
2. D. E. Knuth, Tex and Metafont: new directions in typesetting, *Digital Press and American Mathematical Society* (1979).
3. B. K. Reid, *Scribe: a document specification language and its compiler,* PhD thesis, Carnegie-Mellon University (1980).
4. Honeywell, *Multics Wordpro Reference Guide,* Honeywell Information Systems (1979).
5. B. W. Kernighan, PIC—A crude graphics language for typesetting, Bell Labs. Internal Report (January 1981).
6. C. Corbett, *A Standard Graphics Package for Use under Unix,* Internal Report, Department of Electrical Engineering Science, University of Essex, February (1980).
7. B. L. M. Wyvill, PICTURES-68 MK1, *Software—Practice and Experience* **7**, pp. 251–261 (1977).
8. J. F. Jarvis, C. N. Judice and W. H. Ninke, A survey of techniques for the display of continuous tone pictures on bilevel displays, *Computer Graphics and Image Processing* **5**, pp. 13–40 (1976).
9. I. H. Witten and R. Neal (in preparation), *Bilevel Display of Continuous-Tone Images Using Peano Curves.*
10. T. R. H. Sizer (Ed.), *The digital differential analyser.* Chapman and Hall, London (1968).
11. N. M. Walcott and J. Hilsenrath, *A Contribution to Computer Typesetting Techniques (Tables of Coordinates for Hershey's Repertoire of Occidental Type Fonts and Graphic Symbols).* National Bureau of Standards, U.S. Department of Commerce, Washington, DC. (1976).
12. S. E. Bammel, Automatic full-page formatting of technical primary journals, *Proc. National Computer Conference,* pp. 825–829 (1975).
13. I. H. Witten, M. Bonham and E. Strong, *On the Power of Traps and Diversions in a Document Preparation Language,* Research Report 81/65/17, Department of Computer Science, University of Calgary (1981).
14. H. H. Bown, C. D. O'Brien, W. Sawchuk and J. R. Storey, *A General Description of Telidon—a Canadian Proposal for Videotex Systems.* Communications Research Centre, Department of Communications, Ottawa, Ontario (1978).

Received October 1981