Font News

Dominik Wujastyk

Concrete Roman and Italic

The new book *Concrete Mathematics* by Ronald L. Graham, Donald E. Knuth and Oren Patashnik* is naturally typeset using TEX, and also uses new typefaces. The maths is set in AMS Euler, a typeface designed by Hermann Zapf for the AMS. The text is set in special versions of Knuth's CM family roman and italic, with weights designed to blend with AMS Euler. This has been named Concrete Roman and Italic.

Zapf's design for AMS Euler is intended to suggest the look of mathematics as written on blackboards. This is how maths has chiefly been written by generations of maths teachers and researchers and is the medium in which most mathematics has always been seen by most mathematicians. The face is distinctly calligraphic, as opposed to italic, and in my view achieves the effect it seeks. But it faces the same difficulty as any striking and original new type design: it initially distracts the reader from the underlying text. It would be interesting to hear from anyone who reads *Concrete Mathematics* right through how the typefaces fare after protracted reading.

The Concrete roman face appears to have features in common with the CM typewriter font, although at the time of writing I have not seen the parameter files. It is a face somewhat in the genre of Bigelow's Lucida or Carter's Bitstream Charter, though different from these, of course.

For an example of the Concrete and Euler fonts, see Knuth's article "Typesetting Concrete", page 31 in this issue.

Lucida

In December 1988 Chuck Bigelow informed me that:

> Atari is soon (January 1989) bundling Lucida text fonts with its PostScript clone upgrade for its laser printer, the SLM 804. The Lucida fonts include the TEX text character set. The Lucida math fonts will also be available for Atari systems, but from the Imagen Corp., later in 1989. Also, QMS-Imagen are bundling Lucida fonts in the same character set with a software PostScript clone "UltraScript PC" for IBM PCs and various printers. The Lucida TEX math fonts will also be available from Imagen for that system.

* Reading, Mass.: Addison-Wesley, 1989.

Graphics

Computer Graphics and TEX — A Challenge

David F. Rogers
Aerospace Engineering Department
United States Naval Academy

Of late there has been considerable interest in the inclusion of graphics output within a TEX document. Programs such as PiCTEX, gnuTEX, Fig, and TransFig seek to provide a mechanism for the inclusion of graphics within a TEX document. Each of these programs attempts to provide a nearly complete environment for the design and generation of line art or halftones for inclusion in TEX. All are worthwhile efforts. However, each suffers from a serious problem — device dependence. For example, the PiCTEX macro package is too large to run on a microcomputer or in fact many workstations — it really requires a special large implementation of TEX; Fig and TransFig are graphics device dependent (Sun workstations); gnuTEX only generates LATEX compatible output; etc. Yet one of the strong attractions of TEX is its device independence. TEX itself runs on literally dozens of different machines from Crays, through the latest Silicon Graphics, Ardent, and Stellar superminicomputer engineering/scientific workstations, to a lowly PC XT running MS-DOS.

Systems such as PiCTEX, Fig, Transfig and gnuTEX basically require the user to *recreate graphical output* or to generate it *ab initio*. This is rather inefficient. There are literally dozens of graphics programs that produce better graphical output, more efficiently than any of these systems.

An alternate technique for importing graphics into TEX is to use the `\special` command. Unfortunately, this requires giving up device independence. Further, not all dvi drivers support all `\special` commands.

A Suggested Minimal Set of TEX Graphics Macros

Graphics can be incorporated into TEX documents most efficiently by importing the output of graphics programs directly into the TEX document in the form of Plain TEX commands. The important question is how to do this easily and efficiently. Fortunately, only passive graphics is contained in a publishable document. Consequently, the functional requirements for graphics is quite limited (see Refs. 1 and 2). Specifically, these are the ability to move the writing instrument both invisibly and visibly about the writing surface to create thin lines and small dots (points), to 'plot' finite sized areas called pixels in various colors or monochrome and to generate text at specified size, orientation and location. It is convenient to be able to specify movements and locations in either absolute or relative coordinates. Additionally, it is convenient to be able to initialize the graphics system, to explicitly exit the graphics system and to specify the size and location of the writing surface. Finally, it is convenient to be able to specify the thickness of thin lines.

A small suite of less than a dozen and a half Plain TeX macros can provide this functionality. Only two of these functional requirements create difficulties. Specifically, character generation at arbitrary orientations is bothersome. Initially, character generation orientations should be limited to horizontal and a 90 degree rotation counterclockwise from the horizontal. The latter can be accomplished by using METAFONT to generate a rotated font. The rotated font is positioned and displayed by stacking the character boxes vertically. Additionally, the generation of halftone images is a bit difficult (see below).

Currently color is not normally incorporated into TeX documents. However, with technological advances, this is not far off. Hence, it must be considered.

The Macros

The macros and their suggested calling arguments required to implement this functionality are briefly described below. Unless otherwise specified units are any acceptable TeX values.

`\beginpicture` (Begin picture)

Sets any required initial parameters. Saves all current parameters.

`\endpicture` (End picture)

Resets all parameters to the saved values.

`\viewport#1#2` (Viewport)

- `#1` The horizontal size of the area for the graphics.
- `#2` The vertical size of the area for the graphics.

Default size is `\hsize` by `\vsize`

The origin is in the lower left corner, positive upward and to the right.

Viewport is the common computer graphics name for this function.

`\linethickness#1` (Line thickness)

- `#1` The thickness of a thin line. Default is 1pt.

`\setrsl#1#2` (Set resolution)

- `#1` The number of pixels in the horizontal direction within the viewport.
- `#2` The number of pixels in the vertical direction w

Set resolution applies only to the `\setpxl` command given below. It should be called only once within a picture.

`\ma#1#2` (Move absolute)

- `#1` The distance to be moved invisibly in a horizontal direction (x-direction) in absolute coordinates, i.e., from the origin set by the viewport command.
- `#2` The distance to be moved invisibly in a vertical direction (y-direction) in absolute coordinates, i.e., from the origin set by the viewport command.

Moves the cursor from the current cursor position to that specified by `#1`, `#2` in absolute coordinates.

`\mr#1#2` (Move relative)

- `#1` The distance to be moved invisibly in a horizontal direction (x-direction) in relative coordinates, i.e., from the current location of the cursor.
- `#2` The distance to be moved invisibly in a vertical direction (y-direction) in relative coordinates, i.e., from the current location of the cursor.

Moves the cursor from the current cursor position to that specified by `#1`, `#2` in relative coordinates.

`\da#1#2` (Draw absolute)

- `#1` The distance to be moved visibly in a horizontal direction (x-direction) in absolute coordinates, i.e., from the origin set by the viewport command.
- `#2` The distance to be moved visibly in a vertical direction (y-direction) in absolute coordinates, i.e., from the origin set by the viewport command.

Draws a line from the current cursor position to that specified by `#1`, `#2` in absolute coordinates.

`\dr#1#2` (Draw relative)

- `#1` The distance to be moved visibly in a horizontal direction (x-direction) in relative coordinates, i.e., from the current location of the cursor.

#2 The distance to be moved visibly in a vertical direction (y-direction) in relative coordinates, i.e., from the current location of the cursor.

Draws a line from the current cursor position to that specified by #1, #2 in relative coordinates.

`\pa#1#2` (Point absolute)

#1 The distance to be moved visibly in a horizontal direction (x-direction) in absolute coordinates, i.e., from the origin set by the viewport command.

#2 The distance to be moved visibly in a vertical direction (y-direction) in absolute coordinates, i.e., from the origin set by the viewport command.

Moves from the current cursor position to that specified by #1, #2 in absolute coordinates and creates a dot at that point.

`\pr#1#2` (Point relative)

#1 The distance to be moved visibly in a horizontal direction (x-direction) in relative coordinates, i.e., from the current location of the cursor.

#2 The distance to be moved visibly in a vertical direction (y-direction) in relative coordinates, i.e., from the current location of the cursor.

Moves from the current cursor position to that specified by #1, #2 in relative coordinates and creates a dot at that point.

`\text#1` (Text)

#1 The text string to be plotted.

The current font is used. Default is cmr10.

`\tangle#1` (Text angle)

#1 The angle ccw in degrees from the horizontal at which text is to be plotted. Initially only 0 and 90 degrees are allowed.

`\setrgb#1#2#3` (Set rgb)

#1 The red component of the current drawing color.

#2 The green component of the current drawing color.

#3 The blue component of the current drawing color.

The red, green and blue components of the color are given as decimal numbers in the range 0 to 1 with 0 representing no intensity of the component and 1 full intensity.

Black is indicated by r = 0, g = 0, b = 0.

White is indicated by r = 1, g = 1, b = 1.

The default is black.

`\setpxl#1#2#3#4#5` (Set pixel)

#1 The x-coordinate of the lower left corner of the pixel.

#2 The y-coordinate of the lower left corner of the pixel.

#3 The red component of the color of the pixel.

#4 The green component of the color of the pixel.

#5 The blue component of the color of the pixel.

A pixel is a finite area of the writing surface extending to the right and upward from the location specified by the coordinate given in #1 and #2.

The red, green and blue components of the color are given as decimal numbers in the range 0 to 1 with 0 representing no intensity of the component and 1 full intensity.

For monochrome (gray scale) images the monochrome value is obtained by averaging the red, green and blue components.

`\setgray#1#2` (Set gray levels)

#1 The number of gray levels available.

#2 Parameter that determines the gray level representation scheme.

p – patterning (see Ref. 3 and below)

d – dither (see Ref. 3)

Using patterning the maximum number of available gray levels depends on the resolution of the output device.

With dither the specified maximum number of gray levels sets the size of the dither matrix. It must be a power of 2.

`\setlut#1#2#3` (Set look-up table)

#1 Number of red bits

#2 Number of green bits

#3 Number of blue bits

The number of shades or intensities of red, green, and blue are $2^{\#1}$, $2^{\#2}$, and $2^{\#3}$ respectively. The number of colors is $2^{\#1+\#2+\#3}$.

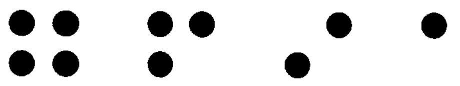
Implementation Considerations

Since normal TeX output is ultimately onto raster scan devices, e.g., laser printers and digital phototypesetters, the line drawing macros must implement Bresenham's line drawing algorithm (or a DDA) (see Ref. 3). PiCTeX, in fact, does this using the period as the plotting character. Although using the period as the plotting character achieves device independence, if variable line thickness is required, using only the period as the plotting character is not sufficient.

Several alternate techniques suggest themselves. Two are mentioned here. The first is to use METAFONT to define graphic plotting characters, e.g., various sized dots, squares or diamonds from which various thickness lines can be constructed. The second is to directly define various size squares using `\hrule` or `\vrule` (Ref. 4 page 64) to use in constructing the various thickness lines. In either case the 'symbols' should be overlapped to decrease aliasing effects.

Inclusion of photographs or continuous tone images in TeX requires dealing with pixels to digitally represent these pictures. Hence the `\setpxl` macro above. Pixels have varying intensity. Printers use a photographic screen process called halftoning to print continuous tone images. There are two principal digital analogs of this process: patterning and dither (see Refs. 3, 5 or 6). Patterning gives up spatial resolution to achieve intensity resolution. Dither introduces randomness into the digitized image to give the impression of multiple intensity (gray) levels without losing spatial resolution. Both, actually print only black 'dots' on 'white' paper. Both are easily extended to color.

In patterning, elements of a small grid are either black or white. For example a 2×2 grid using a single dot size yields 5 intensity levels as shown here



The fifth intensity is, of course, no dots. Multiple dot sizes can be used. With multiple dot sizes, the number of intensity levels is $(\text{width} \times \text{height of the grid})^{(\text{the number of dot sizes}+1)}$. From this it is easy to see that a 2×2 grid with 3 different dot sizes yields 256 different intensity (gray) levels. However, not all these patterns necessarily yield unique intensity levels. Monochrome images are quite adequately represented by 256 intensity (gray) levels. Note also that 256 is precisely the number in a new TeX font. Thus, one method of generating these intensity levels is to create a special graphics font using METAFONT. One word of caution is in order: the number of 2×2 patterns depends on the available spatial resolution. For example, for a 2×2 pattern grid, an image digitized with 512 pixels across its width requires a minimum width on the page of a 300 dpi output device of 3.41 inches. This assumes that one physical output device dot is used for each horizontal grid location. Thus, only 5 intensity levels are available. Higher resolution output devices, e.g., phototypesetters, yield more intensity levels. Minimum acceptable output resolutions are 1000 – 1200 dpi. Consequently, 300 dpi laser printers would be useful for proofing only. One additional subtlety should be mentioned. Unless the patterns are carefully selected, moiré as well as other undesirable patterns appear in the output. These can be minimized by randomly rotating the patterns 90, 180 and 270 degrees. Unfortunately, unless a font rotation macro becomes available, this feature requires four different graphics fonts.

Dither introduces controlled randomness into the digitized image to produce the impression of multiple gray levels. Intensity resolution is increased without loss of spatial resolution. The algorithm is conceptually quite simple:

```
for each scanline in the image
    for each pixel along the scanline
        determine the position in a dither matrix
        i = (x Mod n) + 1
        j = (y Mod n) + 1
        determine the pixel display value
        if image pixel intensity(x,y) <
                    dither matrix(i,j) then
            write black pixel
        else
            write white pixel
        end if
    next pixel
next scanline
```

Details are given in Refs. 3 and 6. The number of apparent intensity levels depends on the dither matrix. Dither matrices are square. Their sizes typically increase by factors of 2, e.g., 2×2, 4×4 and 8×8. The number of apparent intensity levels is the dither matrix size squared, e.g., an 8×8 dither matrix yields 64 apparent intensity levels. The optimal 2×2 dither matrix is

$$\begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

The 4×4 dither matrix, which can be derived from the 2×2 matrix, is

$$\begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Additional intensity levels can be obtained using multiple dot sizes. TEX has quite adequate facilities for implementing a dither algorithm.

Using the Macros

The set of TEX graphics macros described above can certainly be used directly to create both line art and halftones. However, except for touch-up work, that is not their intended use. Their intended use is as the end product of a filter pipe[†] that can be processed by TEX in a device independent way. *Conceptually, the intended pipeline is*

The output of your favorite graphics program is saved into a file.

A program converts this format to a neutral file format here called the standard display file format (.sdf).

A program converts the .sdf format to a TEX file containing only the macro calls described above.

TEX processes this file in the normal fashion producing a .dvi file.

An output driver converts the .dvi file to some device dependent form, e.g., Postscript, HP LaserJet+, phototypesetter, etc.

Assuming that the output of your favorite graphics program is already saved in a file, then the UNIX pipe command line is

```
graf2sdf < graphicsfile ¦ sdf2tex > file.tex
```

Standard Display File Format

The concept of storing a picture in a data file is part of current international standards efforts. The applicable graphics standard is that for the Computer Graphics Metafile (CGM) (see Refs. 7 and 8). However, the CGM is somewhat more detailed and complex than required for the current application. A simple alternative, here called the standard display file, is given in Table 1.[‡]

The first line of the standard display file contains an identification string. Each subsequent line of the file begins with an operation code number, followed by the appropriate information required by that operation. All coordinate values are specified as floating point numbers in the range $0 \le x \le 1.0$, $0 \le y \le 1.0$. All items are separated by commas. A simple standard display file for a square is:

```
0, 0.05, 0.05
2, 0.95, 0.05
2, 0.95, 0.95
2, 0.05, 0.95
2, 0.05, 0.05
```

Using the macros discussed above the corresponding TEX code is:

```
\beginpicture
\viewport{2in}{2in}
\linethickness{2pt}
\ma{0.05}{0.05}
\da{0.95}{0.05}
\da{0.95}{0.95}
\da{0.05}{0.95}
\da{0.05}{0.05}
\endpicture
```

The format is simple enough that a picture can be easily modified or, in fact, generated using a text

Table 1: Standard Display File (.sdf) Codes

Code	Operation	Parameters[†]
0	Move Absolute	x,y
1	Point Absolute	x,y
2	Draw Absolute	x,y
3	Move Relative	Δx, Δy
4	Point Relative	Δx, Δy
5	Draw Relative	Δx, Δy
6	Text	Text string
7	Set color	red, green, blue
8	Set look-up table	red bits, green bits, blue bits
9	Set pixel	x, y, red, green, blue
10	New frame	
11	Pause	
12	Set resolution	xresolution, yresolution
99	Print message	Message string

[†] x,y are floating point numbers in NDC; i.e., in the range $0 \le x, y \le 1.0$. red, green, blue are floating point numbers in the range $0 \le$ red, green, blue ≤ 1 with $0 \equiv$ no intensity and $1 \equiv$ full intensity. Red bits, green bits, blue bits are powers of 2 (see Ref. 1 for a discussion of look-up tables). This table is taken from Ref. 1.

[†] Pipe is used here in the context of a UNIX or MS-DOS pipe.

[‡] A similar concept has been in use at the author's home institute for more than a decade.

editor. The format also allows easy extension to include local or additional functionality.

Conclusions

The TeX graphics macros presented above provide all the capability currently, and for the near future, required to include publication quality line art within TeX source files. They also include all the required capability to seriously begin experimenting with the direct inclusion of halftone images within TeX source files. Including publication quality halftone images within TeX source files requires both further development of digital 'halftoning' techniques and the general availability of higher resolution output devices.

The Challenge

The Challenge is for the macro gurus to implement the above TeX graphics macros and for the METAFONT gurus to generate the required fonts. I'll be happy to consult on the graphics aspects of the development, to test the results and to implement the filter program from .sdf format files to TeX.

Aerospace Engineering Department
United States Naval Academy
Annapolis, MD 21402, USA
E-mail: dfr@usna.navy.mil

References

1. Rogers, David F. and Adams, J. Allan, *Mathematical Elements for Computer Graphics*, 2nd Edition McGraw-Hill Book Co., New York, 1990, Appendix A. (Available in August 1989.)
2. Rogers, David F. and Rogers, Stephen D., A Raster Display Graphics Package for Education, IEEE Computer Graphics & Applications, Vol. 6, No. 4, April 1986, pp 51–58.
3. Rogers, David F., *Procedural Elements for Computer Graphics*, McGraw-Hill Book Co., New York, 1985.
4. Knuth, Donald E., *The TeXbook*, Addison-Wesley Publishing Co., Reading, MA, 1984.
5. Knuth, Donald E., Digital Halftones by Dot Diffusion, ACM TOG, Vol. 6, pp 245–273, 1987.
6. Jarvis, J. F., Judice, C. N., and Ninke, W. H., A survey of techniques for the display of continuous tone pictures on bilevel displays, Comput. Graph. Image Process., Vol. 5, pp 13–40, 1976.
7. Computer Graphics Metafile for the Storage and Transfer of Picture Description Information (CGM), ISO 8632–Parts 1 through 4: 1987; also, ANSI/X3.122-1986.
8. Arnold, D.B., and Bono, P.R., *CGM and CGI*, Springer-Verlag, Heidelberg, 1988.

A Portable Graphics Inclusion

Bart Childs
Alan Stolleis
Don Berryman

One of the serious limitations of current TeX usage is the lack of a portable inclusion of graphics. We propose a means of making this possible. It will require a little discipline on the creation of drivers; the procedures outlined herein should make it straightforward to add this to existing drivers.

Graphics inclusion has been a part of the drivers from the Computer Science Department at Texas A&M since soon after the initial release of the earliest QMS-QUIC drivers. In these cases it required significant positioning by the user and therefore became dependent upon the particular printer the document was destined for. This violated the intent of the dvi file, namely being device independent. Our previous graphics inclusion was much like the "Ph.D. with a screwdriver" concept.

It is my (Bart Childs) opinion that in spite of the beauty, power, and widespread use of PostScript, it is not a suitable answer for TeX output. There are two reasons for this opinion:

- A convenient manner for incorporating fonts from METAFONT is not yet in the public domain. *PostScript downloading of bitmaps is inconvenient at best!*
- The size of PostScript files is inordinately large, and use of the system in networks both clogs the network and makes the printing of documents happen at a fraction of the rated speed of the printer, especially if you use Computer Modern fonts.

These comments should not be taken to imply that the immense contribution of PostScript is not appreciated. I think that PostScript is in severe need of a binary mode. Another question of relevance for *de facto* standards is, will it be good for three-dimensional graphics? Many other questions need to be answered before we should treat it as a standard.

We need only a few elements to enable portable graphics inclusion. These elements are:

1. A standard template for the allocation of the size of the graphical area in the TeX document.
2. A standard means of putting the size of the graphical area in the dvi file.
3. A standard means of communicating the "name" of the file containing the graphics.