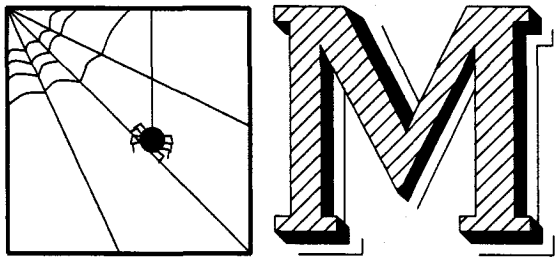
METAPLOT

Machine Independent Line Graphics for TEX

Patricia P. Wilcox

Last winter my husband and I set out to use AmigaTEX to document a collection of FORTRAN mechanical engineering programs. We had a stack of drawings to include: some generated directly by the FORTRAN programs, and some created using the Aegis Draw 2000 program on the Amiga.

It would be a big help to be able to print those drawings with TEX! I decided to give it a try, using METAFONT to do most of the work, and within a few days I had TEX drawing pictures like these two:



From that beginning has grown a set of METAFONT and TEX macros collectively referred to as "**METAPLOT**". Here's how it works. The **METAPLOT** macros enable METAFONT to simulate a line plotter, so that it can turn a (suitably pre-formatted) line plotter command file into a picture variable. The picture is chopped up into rectangular tiles which are shipped out as "characters" that can be typeset by TEX.

This is not a particularly efficient way to do things; there is extra overhead because METAFONT runs as an interpreter, and even more overhead due to the diabolical deviousness of METAFONT's mental processes. METAFONT run time for a single drawing is likely to be several minutes. On the other hand, this approach is virtually system and device independent. Slowness notwithstanding, **METAPLOT** is proving to be a simple and useful way to add line drawings to TEX documents.

We present an account of our journey on the road to illustrated TEX, in the hope that the reader will find some interest therein, and perhaps benefit from advance knowledge of some of the landmarks and pitfalls to look for in his own excursions into the wilder regions of METAFONT and TEX.

Scope of the Project

Our graphics needs were modest, compared with some of the ambitious work being undertaken by other members of the TEX community. For one thing, there was no need to worry about line color. If colored printing should ever be required (unlikely!) we will use the CAD software to separate the colors; what a human printer wants to see is a separate *black* line drawing for each color.

I also chose to ignore dot fill patterns, at least initially, because METAFONT is terrible at dealing with closely-spaced fill patterns over large areas. This is because METAFONT encodes edges, not points; a dot fill pattern has a lot of edges! (I can envision some desperate workarounds for this problem, but basically, if you need to do this sort of thing, you should probably be using a PostScript drawing program with a PostScript output device.)

METAPLOT does not attempt to process half-tone photographs, which have much the same problems as dot fill patterns. Besides, I have enough trouble getting acceptably printed photographs when I am working with an experienced printer who uses a superb copy camera and metal printing plates on a high-precision press! Most TEX output devices just aren't good enough yet to print half-tones cleanly.

By limiting our scope to the representation of black lines and solid areas, we thought to have a useful project that could be accomplished quickly, so that we could get back to the original task of publishing documents.

In Search of Graphics Standards

My first step was an informal look at standardization (or lack thereof) in the engineering graphics field; **METAPLOT** could be much more generally useful if it did not depend on a drawing format specific to one drawing program or one computer manufacturer.

I looked first at standards for the logical representation of graphical objects. The possibilities were IGES (a standard adopted by the U.S. in 1981) [1], GKS (see [1] as well as discussion by Bart Childs et al. in the April TUGboat [2]), and DXF, the AutoCAD drawing exchange format [3], which is something of a *de facto* standard in the industry. Quite a few of the common CAD programs attempt to support IGES, DXF, or both, although what I have been hearing is that you can expect about an 85% success rate in transferring a "standard" drawing between unrelated software packages with IGES or DXF, and that the standards change from week to week. Not good enough!

Although Bart Childs *et al.* (reference [2] again) say that "Most vendors deliver reasonable support for a GKS environment or it is available from third party vendors for common systems," at the time I was designing METAPLOT it was not evident that the GKS standard was supported by *any* CAD programs for any of the common personal computers. It may be that Dr. Childs is talking about large mainframe computers.

The final blow is that not one of the four computer aided drafting programs I use on the Amiga is smart enough to understand IGES, DXF, or GKS.

The next place to look was on the output side: display standards. There are two major philosophies at work in the structured graphics software out there today. For want of a better term, call them "traditional" vector graphics and PostScript graphics. PostScript is rapidly winning the field, because it's more powerful than straight vector graphics, and vector graphics capabilities can be handled as a subset of the functions supported by PostScript.

In a traditional structured graphics program, lines are drawn by moving a pen or an electron beam along the shortest path from here to there. There are no true curves, only straight-line approximations. If fill patterns are used, they have to be something that can be drawn with line segments.

PostScript graphics programs allow you to generate curves from their Bezier control points and fill areas with arbitrarily fine and complex pixel patterns. Many TEX implementations, including AmigaTEX, already have PostScript "\special" commands which allow you to integrate PostScript graphics with TEX documents for output to a PostScript printer. There's just one little problem — if you speak PostScript, you can speak only to something that understands PostScript. Since I want to print TEX documents on "dumb" lasers and dot matrix printers, and I have considerable investment in traditional vector graphics software and data files, PostScript will not work for me, yet.

Things are changing rapidly. It is encouraging to hear about the good work of people much braver than I who are working on PostScript interpreters like the one described in "News from the VORTEX Project" in the April TUGboat [4]. If such interpreters were universally available, the task of importing vector plot files into TEX documents would be reduced to writing a simple program to translate vector commands to PostScript commands.

But, let's face it, PostScript was designed as a "write-only" standard. It's straightforward to write a program that produces PostScript output, but tricky to write a program that does a 100% correct job of interpreting PostScript code and turning it back into a bit-mapped image to drive a non-PostScript device. The "real" PostScript exists only in the microcode of PostScript display devices, and is not generally available to developers. The problem is exactly analogous to what we would face if we were asked to re-create TEX and METAFONT from an external description of their behavior, without the benefit of access to the original code and without "trip" and "trap" tests to ensure adherence to the standard.

Instead of waiting around until there was a PostScript interpreter that could do my work for me, I looked for something less elegant, but simple and general, along the lines of the "standard display file format" described by David F. Rogers [5] in the last TUGboat. It didn't take long to find what I was looking for. A sort of lowest common denominator between all of these CAD software packages is that they all know how to drive pen plotters, using a very small set of graphics primitives. This "standard" has the great advantage of being enforced by a machine. Deviations from standard are punished by the fact that the plotter simply won't work!

If you look at Hewlett-Packard Graphics Language (HPGL), which is understood by HP plotters (and a lot of other plotters on the market) you find the following set of actual drawing commands:

```
Pen motion:
  PU pen up
  PD pen down
  PA plot absolute
  PR plot relative
  CI circle
  AA arc absolute
  AR arc relative
  LB label (draw text)
Line specification:
  LT line type (dot/dash pattern)
  SP select pen color

--------------------------------
Special purpose commands, mostly
for graphs & pie charts:
  FT specify fill pattern
     [type [,spacing [,angle]]]
  EA,ER,EW outline rectangle/wedge
  RA,RR,WG shade rectangle/wedge
  XT,YT draw X and Y tick marks
```

along with a collection of auxiliary commands to do things like plot scaling and plotter initialization and cleanup. The specialized commands below

the dashed line do not really belong in a standard command set. Of the commands above the line, if we omit the "relative" commands, we haven't lost any functionality. This leaves, for a "standard" set of line plotter commands:

```
Pen motion:
  PU pen up
  PD pen down
  PA plot absolute
  CI circle
  AA arc absolute
  LB label (draw text)
Line specification:
  LT line type (dot/dash pattern)
  SP select pen color
```

Could I omit anything else? Looking closely at the actual HPGL plot commands used by CAD software, you'll find that not everything in the list is required by every graphics application. Ignoring plotter setup and scaling, the three programs I use on the Amiga (Aegis Draw, IntroCAD, and mCAD) use just four commands: "move", "draw", "line type", and "pen color". Generic CADD (on the IBM PC) makes do with even fewer commands; it uses only "move", "draw", and "pen color".

However tempting it was to pare down the list further, I had one application (John's FORTRAN plot package) that was going to need CI and LB commands, and I later found another (VersaCAD) that also used CI. Better leave them in.

Plot scaling would be done by scanning the data (after rotation) for min and max x values, and multiplying all coordinates by the ratio of printed width (specified by the user) to width of the data ($x_{max} - x_{min}$). This is scaling from a printer's point of view, where the important final dimension is column width.

METAPLOT — Initial Implementation

Including the commands in the standard command set didn't mean I had to implement them right away. Color was at the bottom of my list; dashed lines were near the bottom; label was too complicated to deal with on the first pass. I chose to start the implementation of METAPLOT by writing METAFONT macros analogous to plotter "move" and "draw" routines.

You may notice that something is missing. METAFONT is not very good at character string manipulation. Surely we don't want to write an HPGL language interpreter in METAFONT! How do we get plotter commands translated to a form that METAFONT can understand?

There were three answers to that. One of the first things we did was to write a version of the FORTRAN plotting routines with output in the form of METAPLOT macro calls instead of plotter commands. This took care of the first stack of drawings.

Translating the second category of drawings (plots from the CAD program) depends on a sneaky trick with Aegis Draw — watch closely! Aegis Draw has the virtue of allowing the user to supply his own plotter configuration file containing an initial string, a separator, and a terminator for each plot command. I created a configuration file defining a rather strange imaginary plotter called "META". When plotting to META, Aegis Draw emits METAFONT macro calls instead of physical plotter commands. Here's a small plot in HPGL, with the equivalent META plot commands as first implemented back in January:

```
HPGL:              META:
-----              -----
IN;DF              beginplot;
SP 1               pencolor(1);
LT 5               linetype(5);
PU;PA 100,100      moveto(100,100);
PD;PA 200,100      drawto(200,100);
PD;PA 150,167      drawto(150,167);
PD;PA 100,100      drawto(100,100);
IN;DF              endplot;
```

Later on, I changed the most frequently-used META plotter commands to more efficient 2-character codes, and added an explicit "-1" for each unused HPGL parameter. (Line type has an optional second parameter indicating pattern size.) The same drawing, revised, looks like this:

```
beginplot;      % EXPLANATION:
sp(1);          % pen color
lt(5,-1);       % pattern,spacing
pu(100,100);    % move to x,y
pd(200,100);    % draw to x,y
pd(150,167);
pd(100,100);
endplot;
```

The third way to convert plot commands to META commands is a preprocessor called VGtoMF, which is just now (May) becoming a reality. I'll save VGtoMF to talk about later, because a lot of things happened to METAPLOT between January and May.

Once the META plot file exists, we need a METAFONT driver file to generate the plot font. This looks a lot like any METAFONT font generation file, with a few extras needed for handling line

drawings. (This sample anticipates the tile/mosaic scheme described in the next section.)

```
mode_setup;
font_size .80 in#;
numeric current_char;
input plotmacs;          % METAPLOT macros
print_width:=1.5;        % Inches!  Using
max_tile_width:=.80;     % dimensionless
max_tile_height:=.80;    % numbers here is
                         % a design flaw.
print_rotation:=-90;
first_letter_code:=1;
plotter_pen_weight:=7;
     %pen weight in plotter steps
mosaic ("myplot")(first_letter_code);
     %characters generated here!
font_slant 0;
font_normal_space 0pt;
font_normal_stretch 0pt;
font_normal_shrink 0pt;
u#=.1in#;                % These values
font_x_height 5u#;       % don't mean much
font_quad 2u#;           % for a line plot
font_extra_space 2u#;    %        ...
bye.
```

The resulting font could be used in TEX by explicitly typesetting the characters:

```
\font\plotfont=myplot50
....
{\offinterlineskip\plotfont
  \centerline{\char1\char2}
  \centerline{\char3\char4}
}
```

or by invoking the "\plot" macro in `plotutil.tex` (the third component of the **METAPLOT** package):

```
{\offinterlineskip\plot 1 {myplot50}}
```

The Evolution of Modern METAPLOT

Dealing with Finite Memory. The biggest problem with our first test plots was that they weren't big enough. They were small for two reasons: METAFONT memory limitations and device driver limitations. A complex plot (with lots of vertical structure) will run out of memory sooner than a very simple plot, but in any case, it doesn't make sense to expect to keep a bit image of an entire plot in memory at one time.

(Another reason for keeping character sizes reasonably small is that some printers are limited to characters 255 pixels on a side; this does not happen with my DeskJet printer.)

It is not very easy to increase memory allocation for AmigaMETAFONT — it uses the maximum memory addressable by 16-bit pointers. Adding more memory would require doubling the size of all address pointers.

Discussing TurboMETAFONT in the last TUGboat [6], Richard Kinch says, "We do not now see the need to include the virtual memory simulator in the TurboMETAFONT programs ... the enterprise of generating fonts does not seem to encourage the use of enormous macros or tables ... ." **METAPLOT** may provide an incentive for including virtual memory in METAFONT, since **METAPLOT** could process a large picture in a fraction of the time it takes now, if the complete picture could be kept in memory at once.

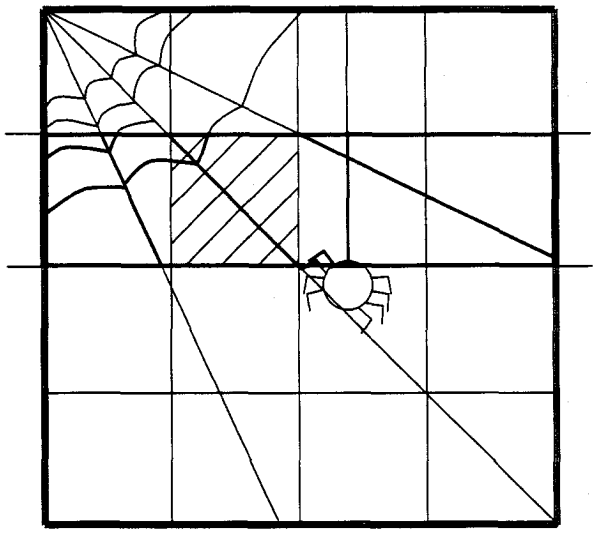


Figure 1. Mosaic tile layout.

Here's our initial solution for dealing with limited memory (Figure 1). A one-dimensional clipping scheme is used, since 1-D clipping is faster than true 2-D clipping. Two new variables (`max_tile_width` and `max_tile_height`) were added to the font generation commands to specify how the picture should be subdivided. The drawing is divided into n horizontal strips, the plot file is scanned n times, and on each pass, all lines are clipped against the top and bottom of the current strip (remembering to add half the pen width at the top and subtract it at the bottom, so as not to lose the edge of a line whose center just misses the box limits); after clipping, any visible parts of the path are added cumulatively to a picture variable. Then (and this part is pretty fast) **METAPLOT** moves from left to right across the strip, and-ing the picture with a

black box exactly the size of one tile, and shipping out the resulting piece of the puzzle as a character.

Note that the clipping is used only to reduce the amount of memory used to store the picture. The final character edges are determined by the and-ing operation.

I had assumed that arbitrarily large pictures could be processed by reducing the tile height, thus requiring fewer square inches of picture to be stored at one time. This works, up to a point, but there is some tile height for which the method breaks down. The figures in this article were well within AmigaMETAFONT's memory limits, even at TUGboat's 723 dpi resolution, but some of our FORTRAN-generated plots exceed memory capacity at large size or high resolution.

Streamlined plots. Having, after a fashion, resolved space problems, I started looking for improvements in the time domain. A 2:1 performance improvement resulted when I stopped culling the picture variable after each path was added. (One wonders what other easy speed-ups might still lurk in the code ...) One thing that would certainly make things faster and reduce the size of plot files would be to dispense with line-segment approximations to circles, curves, and filled areas, and let METAFONT generate the curves mathematically.

It was frustrating — here was METAFONT, which could solve all of our problems, acting like a dumb line plotter. How could the CAD program send circle, spline, and fill commands to METAFONT? Well, we weren't using the pen color command for anything ...

And here was born the first of several "graphical escape sequences". Let's say (assigning arbitrary colors to pen numbers for purposes of discussion) that we use pen 1 (black) for plain lines, pen 2 (blue) for "fill", pen 3 (green) for "filldraw", and pen 4 (red) for "erase". Then if we hop over and appropriate one of the line types on the DRAW menu, and, by convention, call it a circle-drawing pen, and use another spare line type for a spline-drawing pen, we should be able to transmit some fairly useful requests to METAFONT.

Circle-drawing pen conventions: (These are METAFONT near-circles and super-ellipses)

- A rectangle drawn with the circle pen is converted to the ellipse bounded by the rectangle.
- A single line segment defines a circle — leftmost point in print coordinates is the center, rightmost point is on the circumference. (Don't count on your CAD package not to flip lines end-for-end. Saying "first point is center" didn't work at all well.)
- Alternatively (thanks to Bill Hawes for this idea), use a square box to specify a square ellipse, which is, of course, a circle.
- A triangle (which is a 4-point path with beginning and end superimposed) is used to represent an arc — point 1 = point 4 = center of arc. Pick the shorter of the first and last sides; this will be the radius. The arc is drawn counterclockwise around the circle, from shorter side toward longer side.

Spline pen conventions: A splined path consists of $3N + 1$ points, defining Bezier curves, four points per segment, with the center two points of each segment being control points. (It may take a bit of practice to develop the knack of defining a curve by its control points, if your CAD program doesn't display the curve.)

Pen type and path type are defined so that they can be paired up in any combination; you can, for example, draw a green ellipse, and METAFONT will use filldraw to add the ellipse to the picture; or draw a red circle and METAFONT will erase that circular area of the drawing.

This is a bizarre-looking way to enter data! What you see on the screen has very little relationship to the **METAPLOT** picture you are creating. It helps to use brown (invisible to **METAPLOT**) to draw a temporary copy of a line-segmented ellipse or arc as a visual indication of the figure symbolized by the rectangle or triangle you're sending to **METAPLOT**. To keep plot files small, erase the brown temporary copies before writing out the plot for META. (I haven't told you about brown. After trying out the red/green/blue lines, we added pen 5 (brown) for lines that will be invisible to METAFONT except for computing min and max x and y — good for bounding boxes and construction lines. And we added pen 6 (purple) for half-weight lines and pen 7 (orange) for half-weight filldraw.)

Figure 2 demonstrates the use of graphical escape sequences to fool Aegis Draw into generating an assortment of things that are "impossible" to do with Aegis Draw.

Now one last thing would be *really* useful, and that is a way to graphically specify typesetting commands with the CAD software, and have METAFONT pass the typesetting requests along to TEX for final realization. OK, let's see ... to be a legal splined curve, a path must consist of $3N + 1$ points. A rectangle is a 5-point path, so it can't be a spline. We'll specify the position of a typeset

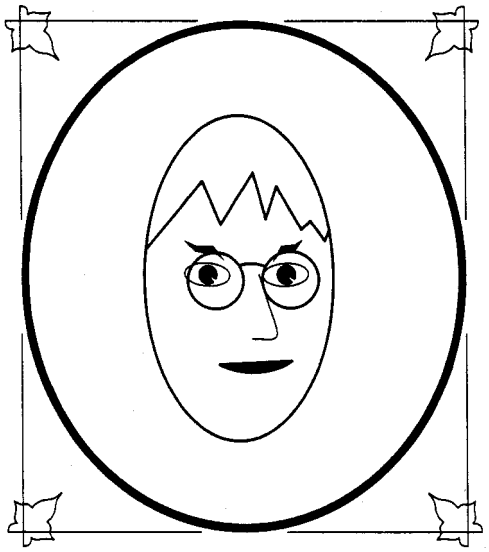


Figure 2. "Self Portrait"
Demonstration of circles, arcs, and splines.

label by drawing a brown rectangle with the spline pen. We can start at the lower left corner for left-justified text, start at the lower right corner for right-justified text, and start at one of the top corners for text centered in the box.

We have METAFONT compute the corner coordinates for each text box (in true inches on the printed page, down and across from the upper left hand corner of the plot), and write a TEX typesetting command to the METAFONT log file, complete with position information and a label number to print on the first draft. After the user sees the first TEX draft of the plot, he can replace label numbers with the appropriate text and move labels by adjusting x and y coordinates in the TEX file, without incurring the overhead of running METAFONT a second time.

While we're on the subject of typesetting, I should mention that complete typesetting information for each mosaic of plot characters has been included in the typeface itself, in the form of `\fontdimen` parameters (thanks to Tom Rokicki for nagging me to do this). METAFONT writes the one-line macro call that reassembles the plot mosaic, along with the rest of the typesetting information, in the METAFONT log file. After METAFONT has created your plot type face, extract the typesetting commands from the METAFONT log file and insert them in your TEX file at the point where you want the picture printed, and you're done.

Using little invisible typesetting boxes, I did the typesetting for Figure 3 at least ten times as fast as I could have done it with pencil and ruler. What a relief!

(Note for dingbat enthusiasts: the `\fontdimen` parameters now include line weight in printer coordinates, so that fancy METAPLOT characters can be joined with TEX rules of the right thickness.)

VGtoMF: A Universal Vector Graphics Interface?

According to what I've told you so far, the Aegis Draw program on the Amiga and a mysterious FORTRAN program are the only programs in the world that can generate command files for the META plotter. If you look at the sample plot listing, you can see that it would not be very difficult to convert HPGL commands to META format using nothing more than a text editor: but this is hardly an elegant solution!

The trick to making METAPLOT portable to all systems is to have a nice simple easily ported C program that reads plotter configuration files describing 1) the syntax of your existing plot file and 2) the command syntax of the plotter you want to convert to. This is intended to convert *from* something else *to* META, but in theory it ought to be able to convert from any plotter to any other plotter.

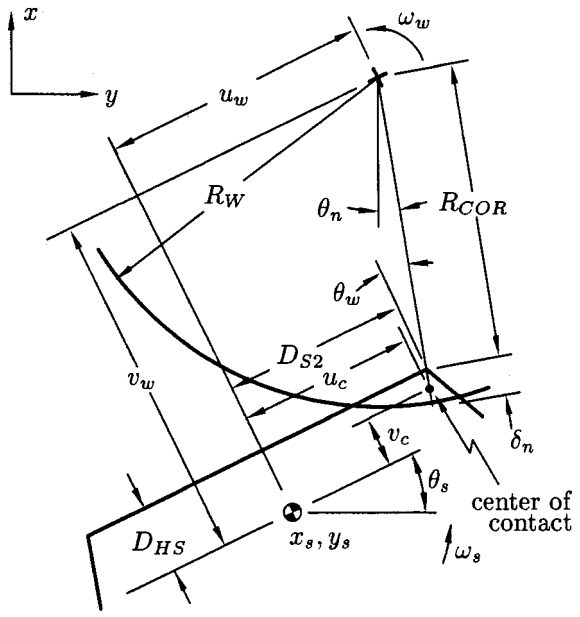


Figure 3. A typical engineering drawing, using METAPLOT typesetting boxes to position labels.

There are a couple of traps here that I should warn you about. Remember the 4095+ limit on numeric values in METAFONT? Given that coordinates in HPGL are usually written as integers, this creates a rather narrow range of plot coordinates where the plot program is not losing accuracy and **METAPLOT** is not blowing up with illegally large numbers in its transforms. A related issue is that, in the present version of **METAPLOT** at least, the x and y dimensions of a picture, measured in printer steps, may not exceed 4095. This is not a problem at low resolutions, but it would limit picture size on a 2000 dpi printer to just over two inches.

Furthermore, mCAD, my favorite shareware graphics program on the Amiga, needs a little help in fixing up its x-to-y aspect ratio. And, as I discovered in preparing the drawings for this article, if you wish to plot at high resolutions, you had better be prepared to center your plot directly over the x, y origin, again to prevent numeric overflow of transforms. Things would be much more comfortable if coordinates were decimal numbers pre-scaled to reasonable limits: say x and y coordinates ranging from about -1000.000 to about $+1000.000$.

The HPGL language permits a path to be represented by a single draw command followed by a list of x, y pairs — yet another syntactic variation that the conversion program must be able to handle.

A quick look at some HPGL output from the Macintosh version of VersaCAD revealed that it was using CI (circle) commands as well as the move, draw, pen color, and line type commands we were expecting. OK, we'll add CI to the list of required commands.

It's becoming apparent that `VGtoMF` has to be more than the simple string-substitution editor I originally set out to write! Required functions are:

- Command string substitution
- Aspect ratio correction
- Coordinate transformation
  - Translation
  - Scaling
- Special work-arounds

It's just about working now, with code to take account of all the quirks I know about, but it's clear that for every new CAD program someone wants to use with **METAPLOT**, we can plan on having to tweak up the code in `VGtoMF` to handle a new set of idiosyncrasies. And, even though I am attempting to write `VGtoMF` in simple straightforward C code, it will take a bit of work to get it to compile and run whenever we try it on a new system.

Of course, with what we've already said about Generic CADD, the graphic escape sequences for curves and fills would have to work with just colors (they have 256 of them), since CADD does not use "line type" commands. I'm beginning to see that my "graphic escape sequences" are simply a way to implement a set of PostScript graphics commands for a CAD program that does not understand PostScript. Since generic CADD *does* understand PostScript, it would make sense to read the PostScript file directly. It's not much of a design change to enhance the `VGtoMF` design to handle a small set of PostScript commands, specified (along with the vector graphics commands) in the configuration file.

What should this PostScript command set include?

I've discussed this at some length with Scott van der Linden, who handles the technical support part of the Generic CADD bulletin board on BIX; I've also studied PostScript output from several other commercial CAD programs. Here are the PostScript features supported by Generic CADD:

- Arcs
- Circles
- Bezier curves (4 points per segment)
- Lines
- Ellipses
- Fills (not supported by Macintosh CADD Level1)
- Conversion of colors to greyscales (not supported by Macintosh CADD Level1)

This is pretty close to the list supported by Gold Disk's Professional DRAW on the Amiga. (Professional DRAW also allows the user to import bit-mapped drawings, but **METAPLOT** will ignore them.)

What has to be added to the list of META commands to support this list? Not much! It looks to me like all we need is to add an "ellipse" command and a "fill" command, and generalize the line type command a little bit.

Ellipse. Suggested ellipse command:

```
el(r1,r2,theta);
```

where `r1` is the length of the semi-major axis, `r2` is the length of the semi-minor axis, and `theta` is the angle of the semi-major axis measured counterclockwise from the positive x axis.

Line type. The syntax of the HPGL line type command is:

```
LT pat_no[,pat_len]
```

where `pat_no (0..6)` specifies a dot/dash pattern and `pat_len` specifies a scale factor for the dashes.

A pattern number 7 (line erase) would be handy. I could add an eighth pattern number indicating Bezier curves, but I'd rather not—this would preclude the possibility of specifying a smooth dashed line. Instead, let's add a third parameter "path_spec", which is 0 for straight joins (the default) and 1 for Bezier curves (4 points per segment). This permits future extensions like path type 2 (free join), 3 (bounded join), and 4 (tense join). I like it! Here's how the `lt` command in META language looks after the change:

```
lt(pat_no,pat_len,path_spec);
```

Fill types. Earlier we rejected the HPGL "FT" command as merely part of a special-purpose pie-chart and bar-graph complex. Let's resurrect it and look at it:

```
FT [type[,spacing[,angle]]]
```

where "type" can be 1 (solid; bidirectional), 2 (solid; unidirectional), 3 (parallel lines), 4 (cross-hatch), or 5 (ignored).

To include black and white PostScript fills, we need to add type 6 (gray scale) and a fourth parameter (percent) to specify the percent black, so the META command looks like this:

```
ft(type,spacing,angle,percent);
```

"Area erase" is "`draw`" with fill-type 6 and 0% fill; solid fill could be "`draw`" with fill type 6 and 100% fill, or possibly just fill type 1 or 2. The PostScript files I've looked at implement "`filldraw`" by breaking it down into separate fill and draw commands: this lets you draw a solid outline with a dot fill.

More generally, it would be nice to draw a sample and say "Fill area with this pattern."

Text. This part is deliberately left vague. HPGL does text by specifying direction (DI) and size (SI) in separate commands and then issuing a label (LB) command. It will take some juggling to define a syntax incorporating METAPLOT's typesetting boxes, HPGL's stick letters, and PostScript typesetting commands (if any—the entry-level CAD programs do not seem to do PostScript typesetting.)

META commands for PostScript Graphics. Leaving text for a future article, here is the list of META commands augmented with the tools for doing PostScript graphics:

```
Pen motion:
  pu(x,y);          %move
  pd(x,y);          %draw
  ci(r);            %draw circle
  aa(x,y,theta);    %draw arc
  el(r1,r2,theta);  %draw ellipse
Line specification:
  lt(pat_no,pat_len,path_spec);
  sp(color);
Fill specification:
  ft(type,spacing,angle,percent);
```

Standard Graphics subsets for PostScript. I would like to propose a nomenclature for talking about PostScript graphics. I steered clear of PostScript for months because the choices seemed to be either 1) no PostScript or 2) writing a full-featured PostScript interpreter.

In fact, METAPLOT includes a well-defined set of PostScript functions, even though it does not call them PostScript. Let's have some formally-defined subsets of PostScript for Graphics!

METAPLOT is capable of doing "Subset 1 PostScript", which consists of the set of functions supported by Generic CADD Level1 for the Macintosh. (Arcs, circles, Bezier curves, lines, and ellipses in black and white only; fills not supported.)

The META language defined above will support a "Subset 2 PostScript"—same as Subset 1, but add gray scale fill capabilities.

To make it support "Subset 3 PostScript" which has the option of color as well as gray scale fills, we may need to add a "color" parameter to `ft`; in other words, fill color and outline color for a filled area are typically two different things, and specifying line color does not affect fill color. "Subset 3 PostScript" corresponds to Generic CADD's IBM Level3 product, and also, I believe to Gold Draw's Professional DRAW program for the Amiga.

METAPLOT will probably support "Subset 2" PostScript eventually, but there are no plans to support "Subset 3" PostScript (colored fills).

Future Directions

The chief items on the menu are

1) Formalized METAPLOT support for reading and writing "Subset 1 PostScript". We should soon be able to translate any vector graphics file to PostScript, using an enhanced `VGtoMF` with the appropriate configuration tables.
2) Making METAPLOT work for new users and new CAD programs on new systems. It may take some work to get `VGtoMF` to compile with non-Amiga C compilers; the METAFONT and TEX macros have so far run perfectly on every system we've tried.

On page 306 of this issue of TUGboat is a METAPLOT order form. Be sure to specify diskette size and format in your order! I've tried to set

the price low enough that it won't be a barrier for any of you who wish to join this adventure into the unknown. I look forward to a challenging group effort to see just how many systems we can get METAPLOT to work on; and I'm excited about the prospect of illustrations bursting into bloom in TEX documents all over the world. I'll try to keep the TUGboat readership up to date on future developments.

Afterward

As we go to press, I've just received my copies of the ANSI *Graphical Kernel System* and *Computer Graphics Metafile* standards. Look at the foregoing paper as a historical treatise on "How Pat Wilcox attained enlightenment on the reasoning behind the inner workings of the Computer Graphics Metafile standard." The standard defines a set of graphical objects very similar to my HPGL-derived list. Two changes are needed: add "elliptical arcs" to my list, and add support for Bezier cubic splines to the CGM standard (splines would be supported as Generalized Drawing Primitives). Add to "Future Directions": incorporate CGM support into the `VGtoMF` program.

References

1. Encarnação, J., R. Schuster, and E. Vöge, eds., *Product Data Interfaces in CAD/CAM Applications: Design, Implementation and Experiences.* Springer-Verlag, Berlin Heidelberg New York Tokyo, 1986.
2. Childs, Bart, Alan Stolleis, and Don Berryman, "A Portable Graphics Inclusion." TUGboat, Vol. 10, No. 1, pp. 44–46, April, 1989.
3. Johnson, Nelson, *AutoCAD: The Complete Reference.* Osborne McGraw-Hill, Berkeley, CA, 1989.
4. Harrison, Michael A., "News from the VORTEX Project." TUGboat, Vol. 10, No. 1, pp. 11–14, April, 1989.
5. Rogers, David F., "Computer Graphics and TEX — A Challenge." TUGboat, Vol. 10, No. 1, pp. 39–44, April, 1989.
6. Kinch, Richard J. "TurboMETAFONT: A New Port in C for UNIX and MS–DOS." TUGboat, Vol. 10, No. 1, pp. 23–24, April, 1989.
7. Tobin, Georgia K. M., *The Elements of* METAFONT *Style.* Preliminary Version, 4 August 1985.

Acknowledgments

Tomas Rokicki gets a large part of the credit for METAPLOT — first, for his outstanding implementation of TEX and METAFONT on the Amiga, and second, for being a constant source of inspiration, information, bug fixes, and reassurance as I pushed his software to its outer limits and beyond.

Thanks to my office neighbors at OCLC, Georgia K.M. Tobin and Rick Tobin, for teaching me about TEX and METAFONT, by a very successful policy of benign neglect coupled with coming instantly to the rescue when I got in trouble. Georgia's instruction manual *The Elements of Metafont Style* [7] was the start of my addiction to METAFONT.

Many thanks also to all the friends and acquaintances who have cheerfully helped out when descended upon by an apparition bearing computer diskettes — "Here, let's see if this will run on your system. Show me your instruction manuals. Can I watch all your graphics programs run? Now can you dump the data files for me? Send me some PostScript!" Some of these long-suffering helpers are (again) the Tobins, who first tried METAPLOT with Personal TEX and showed me MacDRAW II; Bill Hawes (the famed wizard of ARexx on the Amiga); Tim Mooney (the author of mCAD and IntroCAD for the Amiga); Dave Haas of Dartmouth's Northstar Project, who ran METAPLOT for me on the Unix system at Dartmouth and is gearing up to be the number one Atari ST beta test site; and Andrea Ardito and Jack Somerville at Foremost Computer Systems, Inc., who opened my eyes to a whole world of Macintosh wonders in a lightning late-night office tour, and dumped their VersaCAD plot files for me. Thanks to Willy Langeveld who sent me PostScript files from VLT, and, most recently, to Scott van der Linden, who has answered a steady stream of questions about Generic CADD for the IBM PC and Macintosh, and has convinced me that they are Doing Things Right.

And, of course, I need to thank John Wilcox, who is putting up with all this nonsense when I really should be working on his program documentation.

◇ Patricia P. Wilcox
The Coolspring Banjo Works
6617 Home Road
Delaware, Ohio 43015