* * * * * * * * * * *

# Output Devices

* * * * * * * * * * *

## OUTPUT DEVICES AND COMPUTERS

Table I: Proof-Quality Devices

| | Diablo 630 | Epson MX-80 | Facit 4542 | Fla. Data OSP | GE3000 | HP2680 | Imagen Imprint 10 | Laser-grafix | Perq/ Canon | Qume Sprint 5 | Symbolics LGP-1 | Varian | Versatec | Xerox Dover | Xerox 9700 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Amdahl (MTS) | | | | | | | U. British Columbia | | | | | | | | Univ. Michigan |
| Apollo | | | | | COS Info. | | OCLC | | | | | | OCLC | | COS Info. |
| Ethernet | | | | | | Stanford | Imagen, Inc. | | | | | | | Stanford | |
| DEC10 * | | | | | | | Vanderbilt* | Talaris* | | | | | Vanderbilt* | | |
| DEC10 | | | | | | | Stanford (Sail) | Talaris | | | | | G A Technology | | Univ. Delaware |
| DEC20 | | | | Math Reviews | | | SRI; Columbia | Talaris | | | | AMS | | CMU | |
| DG MV8000 | | | | | | | | Texas A&M | | | | | | | |
| HP1000 | | JDJ Wordware | | | | | | | | | | | | | |
| HP3000 | TeXeT | | | | | TeXeT | | | | TeXeT | | | | | |
| IBM(MVS) | | | | | | | | | | | | | | | CIT |
| IBM(VM) | | | | | | | SLAC | | | | | | SLAC | | |
| Prime | | | | | | | | Texas A&M | | | | | Livermore | | |
| Siemens BS2000 | | | | | | | | | GMD Bonn | | | | | | |
| Sun | | | | Textset | | | Sun Inc. | | | | | | | | Textset |
| VAX (Unix) | | | | | | | UC Irvine | Talaris | | | UC* SantaCruz | | Cal. Tech.*; Univ. Wash. | Stanford | |
| VAX (VMS) | | Inter-graph | INFN CNAF* | | | Inter-graph † | Argonne* | Texas A&M | | | Calma* | Sci. Appl.* | Inter-graph † | | |

*Notes*:
* Still running TEX80
† Graphics supported

Most of the interfaces listed here are not on the standard distribution tape. Some of them are considered proprietary. Information regarding these interfaces should be obtained directly from the sites listed.

Output device data is being maintained by Rilla Thedford. Anyone desiring more information or relaying new information can now send it to her on the Arpanet:

`Rilla_Thedford%UMich-MTS@MIT`

Table II: Typesetters

| | Alphatype CRS | APS-5/Micro-5 | Compugraphic 8400 | Compugraphic 8600 | Harris 7500 | Linotron 202 |
|---|---|---|---|---|---|---|
| Amdahl (MVS)* | | Washington St U* | | Washington St U* | | |
| Apollo | | COS Info. | | | | |
| CDC Cyber * | | | | RECAU* | | |
| DEC 20 | AMS | Textset | | | | Adapt, Inc.* |
| HP3000 | | | Univ. of Sheffield | | | |
| IBM 370 * | | Info. Handling* | | | | |
| Sun | | Textset | | | | |
| Univac 1100 * | | | | Univ. of Wisconsin* | | |
| VAX (UNIX) | | | | | SARA | |
| VAX (VMS) | | Intergraph † | | | | |

### Index to Sample Output from Various Devices

Camera copy for the following items in this issue of TUGboat was prepared on the devices indicated, and can be taken as representative of the output produced by those devices.

The bulk of this issue, as usual, has been prepared (using TEX82 for pages 4–11 and TEX80 otherwise) on the DEC 2060 and Alphatype CRS at the American Mathematical Society.

- Autologic APS Micro-5: Description of Metafont Generic Font file format and Gray fonts for Metafont proofs, p. 31; DEC 2060.
- Compugraphic 8600: Dean Guenther et al., TEX at Washington State University, p. 24; Amdahl V/8 (MVS).
- Compugraphic 8600: Goffredo Haus, *How to Tame Your Phototypesetter by TEX*, p. 17; Univac 1100/80.
- Harris 7500: Han Noot, *DVI-Code to the Harris 7500*, p. 18; TEX on CDC Cyber 175-750, typesetter driven by VAX 11/780 (Unix) and/or PDP 11/45.
- HP 2688A Laser Printer (300 dpi): Susan Daniels, *The HP TEX Macros*, p. 49; HP 3000.
- Imagen Imprint-10 (240 dpi): G.M.K. Tobin, *The OCLC Roman Family of Fonts*, p. 36; Apollo.

## HOW TO TAME YOUR PHOTOTYPESETTER BY TEX

**Goffredo Haus**

**C.I.L.E.A. & Istituto di Cibernetica & Te.Co.Graf.**

**viale Pisa, 10**

**I-20146 Milano (Italy)**

I have implemented TEX-1100 from MACC on a UNIVAC 1100/80 mainframe computer at C.I.L.E.A. (Segrate, Milano).

I have used a Compugraphic 8600 phototypesetter as a quality output device and various low-cost graphic devices to edit documents.

I have read the CG8600 User Manual and I have found a lot of features which seem to be very useful to produce my documents but, unfortunately, TEX is not able to control them.

So, I have successfully tried to deceive TEX: I have built (by means of the FONTPRE program from MACC) a TFM file in which ASCII codes correspond to particular CG8600 commands to be treated as zerowidth, zerodepth, zeroheight characters by TEX.

The CG8600 driver program SETTEX (from MACC) recognizes the font code and produces a command string suitable for the phototypesetter according to the character code.

For example, I can write white-on-black or *I CAN GET* SMALLCAPS*ITALIC* FROM SMALLCAPS FONT.

This simple trick, allows me to completely control CG8600 directly from the TEX source of my document by means of TEX processor and SETTEX driver program using my special TFM-commands file.

# DVI-CODE TO THE HARRIS 7500

Han Noot

## Introduction

In 1981 it was decided that at SARA (Stichting Academisch Rekencentrum Amsterdam) TEX would be installed on a Cyber 175-750 under NOS/BE 552. SARA is a computing center which provides its service to Amsterdam's two universities and to Stichting Mathematisch Centrum (the author's institute) among others. TEX's DVI-code was to be processed at SMC, where a Harris 7500 phototypesetter, driven by a VAX 11/780 and a PDP 11/45 are available. (On both computers the UNIX operating system runs.) The filter which converts DVI-code to Harris 7500 code as well as the various programs for the creation and maintenance of font metric files would be implemented and run under UNIX at SMC.

At SARA, one obtained a Cyber version of TEX from Erik Bertelsen, RECAU, Aarhus, Denmark. We at SMC obtained a TEX version for the VAX under UNIX from Robert Morris, University of Massachusetts, Boston, USA.

## The Harris 7500 phototypesetter

The Harris 7500 phototypesetter is a CRT machine with typefonts stored in digital form on an internal system disc. It produces output on photographic paper or film. Its setting speed highly depends on character size and the number and size of moves between characters; the estimated speed is between 200 and 300 characters/second when the typesetter is driven through a 9600 Baud serial interface. Characters can be explicitly positioned with an accuracy of 0.05 points in both the horizontal and vertical directions.

When used in so-called 'slave mode', the Harris 7500 accepts a set of commands which is an almost ideal target language for the translation of DVI-code. Its main features that interest us here are:

- Horizontal- and vertical move commands which perform displacements relative to the current position in the film plane. They come in two flavors: utilizing absolute machine units (0.05 points) or relative units (1/72 of an 'em' at the current point size).
- Horizontal and vertical position commands relative to the origin. (A user defined point in the left margin.)
- Characters can be typeset with- or without automatic updating of the writing beam position.
- Two character size definition commands. The first fixes the overall pointsize of the characters (in units of 0.1 points), the second can override the pointsize in the horizontal direction as determined by the first. As a result, horizontal- and vertical pointsize can be set independently.
- A rule command for hardware generated rules.
- Slanting hardware to produce slanted versions of non-slanted characters. There are three possibilities: rotation through 9, 12 and 15 degrees. (This feature only gives typographically good results with sans-serif characters.)
- Characters are grouped into fonts of at most 128 characters each.

## A naive approach

For every TEX font we need two different font information files: the familiar TEX font metric file and a file to be used by the DVI-code translator. This last file contains information on the correspondence between TEX characters and Harris characters and the width of every character. (Width information is included because the code translator must be able to update the horizontal position after typesetting a character.)

We have designed a symbolic human readable font description from which both font information files can be generated by program. These programs use a data base of height-, depth-, and width values for all the characters that are available on our typesetter. (We obtained this information through the help of the Harris corporation.)

The symbolic font description consists of:

- a header section,
- the character mapping section,
- ligature specifications,
- kern specifications,
- lists of characters of ascending size,
- extensible character definitions,
- a font parameter part.

The header section just contains the font name, design size, character coding scheme and whether the font is a 'NOS-TEX' font or a 'UNIX-TEX' font (see below).

The character mapping section contains one line for every character in the TEX font. Such a line has the form:

N*int* F*name* C*int* S*dig* I*frac* P*int* H*int*

in which N, F, C, S, I, P and H are fixed key-letters. The meaning is the following:

N*oct:* We are talking about the character with octal number *oct* ($0 <= oct <= 177$) from the TEX font being defined here. (We use octal numbers because they are used in the font examples in appendix F of the TEX manual too.)

F*name:* This character is represented by a character taken from the Harris font indicated by *name*.

C*dec:* We use character number *dec* ($0 <= dec <= 127$) from this Harris font. (That we have a decimal number here is for easy correspondence with Harris documentation.)

S*dig:* The character must be set slanted by 9, 12 or 15 degrees (*dig* = 0, 1 or 2).

I*frac:* The italic correction is *frac* * *character-width*, where *frac* is a decimal fraction.

P*int:* The typesetter must set this character at pointsize *int* (specified in units of 0.1 points).

H*int:* The horizontal pointsize must be *int*, instead of the value specified by the P-field.

The ligature specification consists of one line of the form:

N*oct1* N*oct2* L*oct3*

for each ligature, indicating that TEX chars *oct1* and *oct2* are to be combined to ligature *oct3*.

Kern specifications are of the form:

N*oct1* N*oct2* W*frac*

indicating that the kerning value for TEX characters *oct1* and *oct2* is decimal fraction *frac* of the width of char *oct1*.

Next come lists of characters of increasing size (e.g. a list of left braces). There is one list per line.

Extensible characters are specified by lines of the form:

N*oct1* T*oct2* M*oct3* B*oct4* E*oct5*

saying that TEX character *oct1* is extensible, having as top part character *oct2*, as middle part character *oct3*, *oct4* as bottom part and *oct5* as extension component.

This symbolic TEX-font description together with the data base of Harris character dimensions was thought to contain all the information needed to generate TEX font metric files by program, but a few somewhat unexpected problems cropped up. They required an extension of the character mapping definition. We deal with them in the next section. Meanwhile we mention that while generating font metric files, our program performs various consistency checks. For instance, it is checked whether a character is not both declared to be the first of a ligature pair and the first of a kern pair. (By the way, why does TEX not allow such -albeit theoretical- possibilities?)

The device font files for the DVI-code translator are generated using only the header- and character mapping parts of the symbolic font specification. They contain a machine readable form of the N, F, C, S, P and H fields of the character mapping lines. Furthermore, they contain the width of the characters, both in TEX's rsu's and in typesetter units. These twofold width values are used to eliminate rounding errors. As soon as the theoretical (correct) position on a page, measured in rsu's differs from the physical (rounded) position measured in typesetter units (0.01 point) by more than one typesetter unit, the typesetter is instructed to make a compensating move.

A complication with device font files is that we have to generate different files depending on whether we use TEX running under NOS or TEX under UNIX. This is so, because our 'NOS-TEX' and 'UNIX-TEX' use different rsu's, namely a $2^{-16}$ point rsu respectively a $2^{-20}$ meter rsu. Hence our DVI-code translator unfortunately has to be aware of which type of DVI-code it is translating. This is moreover so because the two TEX versions produce slightly different postamble formats in their DVI-code.

**Complications**

The font description scheme discussed so far would work fine, if not for a number of practical complications, which we will discuss now.

In the first place, Harris characters are generated by digitizing hand drawn art work. The result is that even in a straightforward typefont like Times Roman, not all capitals are of exactly the same height nor do they have equal depth. The same applies to lower case characters: 'p' does not always have its height exactly equal to that of 'r' or its depth equal to that of 'g'. As a consequence, font metric files cannot be generated by directly using character dimensions from the data base (after some unit conversions). That way we would exceed by far the limit of sixteen different depth- and height- values per font as allowed by TEX. We considered automatic rounding to sixteen different values but had to discard this for the following reason: In the extension font, extension components should have an (almost) exact depth, while the height and depth of for instance the '+' operator may be wrong by quite a lot. A symbol like 'Σ' is an intermediate case. So rounding must be done under explicit human control.

To make that possible, we introduced into the character mapping lines an optional A-field (A comes from as) which can take one of the following forms:

A*arg*, Ad*arg*, Ah*arg*, or Ah*arg*d*arg*

in which *arg* is either an octal number or a period. If a line reads:

N157 ... A141

it means: take the height and depth of character 157 equal to the height and depth of character 141 from this same TEX font. Ah*i*d*j* means: take the height equal to that of character *i* and the depth equal to that of character *j*. Ad*j* means: give the character its own height but take its depth equal to that of character *j*. Finally, a period instead of a character number (e.g. A. or Ah. etc.) means: take the character dimension indicated by the period equal to zero.

The next complication arose from the fact that in TEX's extension font, components of extensible characters must have a height of zero while the corresponding Harris characters intersect the base line. * Furthermore, symbols with limits (like $\Sigma$) only come out right when they have a height of zero too. As a remedy, we introduced an optional U (**u**p) field in the character mapping lines. It is more general than is strictly needed, but can be turned to good use in other circumstances. This U field comes in four forms:

Ud, Uu, Ud*frac* or Uu*frac*

meaning the following: Ud pushes the Harris character representing the TEX character so much down, that its top touches the base line. Ud*frac* pushes the character down by an amount equal to *frac* * (*character-height* + *character-depth*), in which *frac* is a decimal fraction. Uu and Uu*frac* work like Ud and Ud*frac*, but now in the upward direction.

When all this was done, two problems remained: there were (just visible) gaps between occurrences of some extensible character components and between objects like root signs and rules connected to them. The problem with extension characters seems entirely due to the fact that there apparently are minor discrepancies between character height and -depth as given in the typesetter documentation and their real physical dimensions. We introduced an E (epsilon) field for that. It tells the font metric file generating program to take the height and depth of a character a fraction 'epsilon' (in our case 0.02) smaller than the values contained in the character dimension data base. This produces just enough overlap between extensible character components to make the extensible character come out fine.

The problem with roots (and horizontal braces) is due to the fact that there is sometimes a bit of white space at the left and right of certain typesetter characters where we do not want it. Solution: the R- (reduce white space) field. It has the form:

R*lfrac*1*rfrac*2

(The order of the l*frac*- and r*frac* parts may be reversed and one of them may be omitted.) The meaning is: reduce the white space to the left of a character by *frac1* * *character-width* and to the left by *frac2* * *character-width*. (*frac1* and *frac2* are signed decimal fractions.) This is done by having the typesetter perform small horizontal moves before and after setting the character. It solves the problem with roots and furthermore it can be used more generally to make differently spaced fonts out of existing ones.

To sum up: apart from the fields discussed in the previous section, a character mapping line may contain:

- an A field to specify that character dimensions must be taken equal to those of other characters,
- an U field to specify vertical displacement of the character,
- a R field to specify changes in the amount of white space surrounding the character,
- an E field to specify that the character height and -depth must be taken somewhat smaller than the physical values.

A character mapping line **must** contain a N, F and C field; it **may** contain S, I, P, H, U, R, E or A fields. When font metric files and device font files are generated from the symbolic font description, the order of processing of the various fields of a character-mapping line is crucial. It is the order in the summary of the fields just given. First the character height, -depth and -width are obtained from the data base. Height and depth are updated according to the ratio of the character pointsize (P-field) and the pointsize for which the data base values are applicable. The same is done for the width, using the horizontal pointsize if a H field is there, otherwise using the P-field. Next, a width value may be affected by a R-field; height- and depth values are modified by U-, A- and E- field values, in that order. Finally all values are converted to proper units and inserted in the device font- and font metric files.

The rationale for this processing order is the following: First the precise dimensions of the physical character as a result from pointsize specification,

* By the way, we wonder if this fact might not be mentioned explicitly in appendix F of the TEXmanual where the extension font is described. We had to deduce it from TEX's behavior when big brackets turned out far too tall. Only later on, a short reference to this phenomena, contained in an appendix of the METAFONT manual, was brought to our attention.

up/down moves etc. are calculated. Only thereafter, deliberate errors may be introduced by E- and A-field values.

The specification of an symbolic font should proceed in corresponding stages. First, the font is specified without R-, E- and A-fields. After inspection of a human-readable version of a font metric file (probably still containing too many height or depth values), A-fields are introduced as needed. Thereafter, on evidence from the typeset output, R- and E fields can be added. E-fields may only be added to character specifications **not** containing A-fields or to **all** characters which have the same height and depth value (taking into account their A-fields). If this restriction is violated, extra height- and depth values may again appear.

Finally, it may be useful to sum up which information from character mapping lines is reflected in font metric files and which in device font files. First the font metric file: The F and C fields together with the character data base determine initial character dimensions. These are then acted upon by eventual P, H, U, R, E and A field values, so all these fields may influence the contents of a font metric file. Furthermore, the I- (italic correction) field is used in this file too. On the other hand, only the F, C, P, H, I, U and R fields determine the contents of the device font file. F, P, H and I specify the state in which the typesetter must be while typesetting the character, the U and R fields describe moves to be performed in the film plane before and after setting the character.

**Miscellaneous topics**

To assist in the task of creating and maintaining symbolic font description files, Gertjan Vinkesteijn has implemented a kind of special purpose editor. This program interactively asks for the values of the various font description fields, checks the response for correct format, range in which values may lie, etc.

The translator program, which generates typesetter code from DVI code can be called with a number of optional arguments which are used to specify:

- whether 'NOS-DVI'- or 'UNIX-DVI' code must be processed,
- which pages from the document must be typeset (all of them, only individual ones, ranges of pages) and in which order,
- whether some special symbol must be typeset in the margin in order to demarcate pages (to assist in cutting them later on) and if so which one,
- whether all pages must be set to the length of the largest one or to their individual length (which saves some quite expensive photographic material but which may annoy a printer).

As already stated, all programs are implemented under UNIX in the programming language C. There is no fundamental reason for this however. Everything could have been equally well coded in say PASCAL, but we quite heavily use the UNIX system call which makes possible to move freely forward or backward to any position in a file.

**Conclusions**

The claim made by some that DVI-code can be used to generate code for almost any reasonable output device has not been falsified by our experience. On the contrary, it was quite straightforward to transform DVI-code to code for the Harris 7500. On the other hand, there is quite a lot of 'device dependence' (METAFONT dependence?) hidden in the limitations on font metric files and especially in the peculiarities of TEX's extension font. In particular, the 'zero-height' requirement for certain characters considerably expands the amount of code to be generated for our typesetter. To typeset **one** extension component **seven** bytes must be sent across the typesetter-interface. The first three generate a downward move (one opcode, two operand bytes), the fourth byte is the actual character and next come three bytes to move up again. (Some of these moves could be optimized away, but only at the cost of complicating the device font files with otherwise unnecessary height- and depth- values of characters.) We wonder whether the reason (unknown to us) for the zero-height requirement is compelling enough to justify the complications it introduces in driving output devices and generating font metric files. Furthermore, could there not be two types of font metric files: one of the current compacted kind and one allowing 128 different values for every character dimension? In spite of its cost in memory, we certainly would have used the latter type for the extension font. Meanwhile we are curious to know, whether there are others who have encountered problems analogous to the ones described here.

To end with a note of optimism, when everything finally worked, we indeed got typeset output which looked quite attractive!