Tools

\textit{\LaTeX} innovations at the Louis-Jean printing house

Maurice Laugier and Yannis Haralambous

Abstract

In this paper we will present several \TeX{} innovations, conceived, or currently under development, at ILJ (the Louis-Jean printing house). ILJ was founded in 1804, at Gap (Southern French Alps) and employs 130 people. Due to its specialization in typesetting and printing of scientific books, ILJ has been using \TeX{} since the late eighties. Currently about 30\% of ILJ's book production is done with \TeX{}. New developments in the \TeX{} area sponsored or financed by ILJ are described in this paper.

--- * ---

In exactly ten years ILJ (Imprimerie Louis-Jean) will celebrate its bicentennial. Needless to say, this printing house has followed closely all developments of the printing industry: leaden types from the early x\text{th} century until 20 years ago, photocomposition in the seventies, and since the early eighties, the computer. Almost everything has changed: authors have changed, publishers have changed, even the product a printing house produces is not the same anymore: some years ago one was making books, now they are more and more often accompanied (and perhaps will be replaced in a few years) by those small silver disks, called CD-ROMs.

In the old days, the author would most probably supply a manuscript. Often one had to rival Champsollon's skills to decipher these manuscripts, in order to be able to compose them. Later on one used to receive manuscripts in typed form; no deciphering was necessary any more, but this implied the work of an intermediate person, usually the author's secretary, or the author him/herself.

In the last decennary authors have bought personal computers; together with these engines they bought programs that make them believe they can typeset a book. More and more authors send "ready-to-print" books on floppies, which most of the time are "ready-to-throw-away". Since it is not possible to teach authors the rudiments of typography, one has to invest time and energy in getting the most out of these files, and finally be able to print the book the author had in mind. For this reason, one has to be able to offer the whole spectrum of services, starting with text input, and finishing with industrial printing.

Typesetting a book in \TeX{} is an even bigger challenge, since the printing process requires also a strong know-how in programming: one has to know \TeX{} sufficiently well to either write the \TeX{} code for a book, or modify the code supplied by the author; one has to know SGML if the text is received marked up in that language and has to be converted into \TeX{}, or inversely, if the author and/or publisher wishes to have the book in SGML form; one has to have some knowledge of PostScript in case something goes wrong at the color separation or flashing stage, and so on.

It follows that a minimum number of services ILJ has to supply are:

1. Processing of \TeX{} and \LaTeX{} files, at the input, DVI or PostScript level, that is:
   - writing \TeX{} or \LaTeX{} code, or converting Word, WordPerfect, Mathor, etc., documents into decent \TeX{}/\LaTeX{} code;
   - correcting it;
   - checking the page setup;
   - incorporating illustrations;
   - coloring it;
   - providing \TeX{}nical assistance on the development of the \LaTeX{} style file;
   - producing an SGML representation of it;
   - selecting or creating if necessary the fonts required for it.

2. Making high resolution films (1200 to 2400 dpi).
3. Industrial printing, binding, routing.

To be able to solve reasonably quickly the problems arising in a process as complicated as the one just described, ILJ had to develop a certain number of tools. The fact that \TeX{} is an open system with no commercial maintenance was a risk to take; it also gave ILJ the opportunity of developing auxiliary tools which it would be impossible to make in conjunction with closed "ready-to-use" systems such as PageMaker or Quark XPress.

1 Oriental scripts

The second author, while working at the Institute of Oriental Languages and Civilizations in Paris, wrote a typesetting system for Oriental languages, based on \TeX{}. ILJ has contributed to these projects both technically (by providing the necessary back end for Oriental typesetting and printing), and financially.

Together with John Plaisce (Université de Laval, Québec), the second author is also developing \Omega, an extension of \TeX{} internally based on ISO/IEC 10646/UNICODE. ILJ is ready to adopt ISO/IEC 10646/UNICODE as the fundamental encoding for text processing, in order to solve once and for all the problem of
encoding ambiguities, a problem which can be very painful for texts with special needs (texts using symbols and/or non-Latin characters).

2 The TradTeX→SGML program

2.1 The principle

The TradTeX→SGML program, developed by Franck Spozio, is an assistance to conversion of TEx or DTEx files into SGML. The user of TradTeX→SGML must have a fairly good knowledge of both TEx and SGML, since the process of conversion involves a stage of analysis.

TradTeX→SGML creates a database containing all TEx codes it has encountered as well as their structure; this database file is accumulating information on TEx commands, from several runs: primitives, standard macros, or user-defined ones. Furthermore, it checks the syntactic validity of these commands, in case the same command is defined with different structures in different files or contexts.

The file “DicoLexico” contains the TEx commands and their structures. As the reader can see in fig. 2, this database file can be edited and modified on the fly. The file “DicoTraduction” contains the SGML entities corresponding to the TEx commands, as well as their structure. The TEx codes which have to remain unchanged in the SGML file, in a <NOTATION> entity (for example, those describing math formulas) are stored in a special file, called “liste tableaux”.

When reading the TEx code, TradTeX→SGML analyzes all TEx tokens; whenever a token has not been defined or contradicts the description contained in “DicoLexico”, the user is prompted to specify the structure of the command (by a dialog as in fig. 3);

the data requested are (a) the number of arguments, (b) the nature of delimiters, (c) if it is acting on one or more tokens coming after or before, (d) special characters, etc. Whenever the TEx command affects the formatting of the text, the corresponding informations are stored into the file “DicoDTD”; they will be used to create the DTD file which will mimic the original formatting specifications of the file.

Once this informations is stored, the user is prompted for the translation of the token in SGML (names of elements, attributes, processing instructions, entities, etc.). This information is then stored in file “DicoTraduction”.

Once all tokens have been read and verified, the translation of the TEx file into SGML begins, using information from all four database files mentioned above; at the same time a DTD file is created. An example follows of a B3TEx file (with two abstracts, a section and a subsection, an array of equations,
a table, a figure, a list, and a bibliography) and its
SGML conversion:

```xml
<document>
  <title>Essai nouvelle figure</title>
  <figure>
    Essai de composition sur plusieurs colonnes
    <biblio>
      <bib>Essai de bibliographie {lit avec de
      l'italique, avec du \bf gras dans le texte}
      \bib{2}\bib{1} la représentation scalaire distribuée
    \end{biblio}
    \end{multicols}
  </figure>
</document>
```

The SGML file produced by \TeX{}→SGML is:

```
<DOCTYPE "document.dtd">
<!--documentstyle[multicol,ts]{book}-->
<!--application/pgp[signature]{}-->
<!--application/x-gzip{}-->
<doc>
  <title>Essai nouvelle approche</title>
  <abstract>
    Essai de composition sur plusieurs colonnes
    \begin{multicols}{2}
    \section{Titre de section titre de section}
    Essai de composition sur plusieurs colonnes
    \subsection{Titre sous-section.}
    Essai de composition sur plusieurs colonnes
    \begin{eqnarray}
    x & = & 17y \\
    y & > & a + \ldots + j + \nonumber \\
    & & \nonumber K + 1 + m + n + o + p
    \end{eqnarray}
    \end{equation}
    Essai de composition sur plusieurs colonnes
    \begin{eqnarray*}
    & \&
    &
    \end{eqnarray*}
    \end{equation}
  </abstract>
  <file>
    \begin{multicols}{2}
    \begin{tabular}{|l|l|l|}
    \hline
    haut gnats & gram & \$13.65$
    \bas
    \end{tabular}
    \end{multicols}
    \begin{figure}
    \vglue3cm
    \caption{Essai premi\'ere figure}
    \end{figure}
    \begin{multicols}{2}
    \begin{tabular}{|l|l|l|}
    \hline
    gnus & stuffed & 92.50
    \bas
    \end{tabular}
    \end{multicols}
    \end{figure}
  </file>
</doc>
```

Interactive and produce quick and efficient SGML code. Interactive SGML editing software is still too expensive for the average author, and often requires working stations or high-end personal computers; on the other hand, \TeX has become a de facto standard of document preparation system in several branches (especially mathematics and computer science): although it may not seem very elegant to a purist, generation of SGML code out of post-treatment of \TeX code is an efficient low-cost solution, whenever (and this happens more and more often) the publisher requests the source code of the book in SGML form.

Trad\TeX\rightarrow SGML is presently implemented on Macintosh, and is used in real-life production by ILJ.

3 \texttt{edViitor}

\texttt{edViitor} is a program developed by Philippe Spozio (brother of Franck Spozio). It allows interactive editing of a DVI file, using a mouse-driven cursor to move blocks of text, insert illustrations, change colors, etc.

People are often shocked when they see a DVI file edited with tools similar to those of graphical programs. It seems that \TeX users consider a DVI file as something immaculate. It is most often created by \TeX (actually a DVI file can also be created by DVICopy, DVIDVI, GfToDVI and other similar utilities), and a lot of information is pumped out of it; it can be printed on any device, previewed on any screen, it can be faxed, or converted to PostScript (and hence to PDF format). But none of these drivers and utilities change the text formatting in a DVI file; DVIDVI will perhaps change the order of pages, DVIcopy will replace characters by other characters with the same metrics, and drivers do not modify a DVI file.

There is a reason behind this: according to \TeX ideology, \TeX does the ultimate text formatting, it would be vain to modify it manually. This of course is true, if we consider \TeX’s line breaking algorithm, or the typesetting of math formulas.

But what about titles, figures, or horizontal lines? We are forced to admit that these depend on the taste of the... human typesetter, rather than on \TeX’s skill. After all, to place a horizontal line or an illustration, we give millimetric instructions to \TeX, concerning both the size of the object and the size of the surrounding white space. And these instructions can very well be wrong, or \textit{slightly} wrong.

In the best of all worlds, one would run \TeX on a file as many times as necessary, until the file is perfect from all points of view. In a real-life production world this is unfortunately not possible: a 600-page
book can be run only a limited number of times. \textsc{edVITor} allows us to make "those small last-minute changes", directly on the (otherwise perfect) DVI file.

Of course, the same changes have to repeated every time a DVI file is created anew (unless the corrector is smart enough to report the changes also to the \texttt{TeX} code file). In fig. 4, the reader can see a DVI file progressively modified by dragging blocks of text around.

\textsc{edVITor} has been implemented both on DOS and on Macintosh. It allows also insertion of illustrations, by simple copy and paste operations, coloring of text and color separation of the whole document. A second version of this program, currently under \texttt{alpha}-testing, executes commands placed into the DVI file by the means of \texttt{\special} commands. Here are some commands which can be executed by \textsc{edVITor} v2:

- Color processing:
  \begin{verbatim}
  \special{Color color1}
  text in color 1
  \special{Color color2}
  text in color 2
  \special{Color color1+color2}
  text in color 1+2
  \end{verbatim}

- Vertical column adjustment:
  \begin{verbatim}
  \special{Post Equalize \small Column1,Column2,Column3 \small Height=25cm Pages 17-27}
  \end{verbatim}

This command will spread paragraphs so that columns 1, 2 and 3 of pages 17-27 will have the same height, namely 25cm. To use this function one has to specify first the parameters of stretching and shrinking of interparagraph blank space.

\begin{verbatim}
\special{Post Expand sup=1pc inf=-2mm}
\end{verbatim}

This command will modify interline space.

- Positioning and moving around figures:
  \begin{verbatim}
  \special{Post InsertAt x=10cm y=2.4cm Pages 1, [76] illustration figure.eps}}
  \end{verbatim}

EPSF figure figure.eps is placed at coordinates $x = 10cm$, $y = 2.4cm$ on page 1 (folio 76).

- Inclusion of graphical commands: These commands allow framing of a block of text, or positioning of simple geometrical figures (rectangles, circles, ellipses), eventually filled with a certain color or gray density.

\begin{verbatim}
\special{Post SetFillDensity 30}
\special{Post FilledZoneFrame}
\end{verbatim}

- Inclusion of external DVI files:

\begin{verbatim}
\special{Post Import fichier.dvi}
\end{verbatim}

The DVI file fichier.dvi will be merged into the current DVI file. Suppose you are writing a book on \texttt{TeX} and want to include an example of \texttt{TeX} file output: for example a beginning of chapter page. Up to now there were two solutions: either simulate the beginning of chapter by writing the corresponding \texttt{TeX} code, or take the real \texttt{TeX} file you want to show, produce a DVI file, run dvips with the \texttt{-E} option and obtain an EPSF file, and include the latter in the original DVI file as an illustration. Of course the latter solution has the disadvantage that it is not device independent anymore: the EPSF file unavoidably contains bitmap fonts in a fixed resolution. \textsc{edVITor} allows you to include a DVI file into another DVI file.\footnote{The second author always wondered how D.E. Knuth did volume E of Computers \& Typesetting, where \texttt{GFtoDVI} printouts are mixed with \texttt{TeX} code.}

It should be noted that modifications not involving PostScript are applied to the DVI file when processed by \textsc{edVITor}: in that way, (a) one obtains a new DVI file, modified according to the \texttt{\special} commands, and processable by any \texttt{TeX} driver, (b) since the \texttt{\special} commands are in the \texttt{TeX} code, these modifications are automatically applied whenever \textsc{edVITor} processes the DVI file (so one can produce new DVI files without fear of losing precious information added manually, as in the case of version 1 of \textsc{edVITor}).

It is the hope of the authors that \textsc{edVITor} will be the first step to a "WYSIWYG" \texttt{TeX}, in the sense of more effective DVI file manipulation and possibility of last-minute changes.

\begin{itemize}
  \item Maurice Laugier
  General Director of ILJ,
  Imprimerie Louis-Jean, B.P. 87, 05003 Gap Cédex, France
  Email: louijean@ciig.grenet.fr
  \item Yannis Haralambous
  187, rue Nationale, 59800 Lille, France.
  Email: haralambous@univ-lille1.fr
\end{itemize}
Camera Calibration from Spheres on a Grid Images

Figure 4: Editing a DVI file with eDVItor