Concurrent Use of an Interactive TEX Previewer with an Emacs-type Editor

Minato Kawaguti and Norio Kitajima
Fukui University, Department of Information Science, 9-1, Bunkyo-3, Fukui, 910 Japan
kawaguti@i1mps1.fuis.fukui-u.ac.jp

Abstract

A new efficient method was developed for editing (LA)TEX source files. It uses the combination of an Emacs-type editor and a special version of xdvi. Source files may be edited while browsing through the dvi preview screen simultaneously on the X window screen. Whenever a position is selected by clicking the mouse on a page of the document on display on the screen, the corresponding location of the particular (LA)TEX source file is shown in the editor's buffer window, ready for inspection or for alteration. One may also compile and preview (and obviously edit as well) any part of the entire document, typically one of its constituent files, for efficiency's sake. Fundamental characteristics of the document, shaped by the specification of the document style and various definitions found mostly at its root file, are retained even under partial compilation.

The Editor for TEX

Since it is not easy to grasp what a document looks like by simply reading the TEX source files, the efficiency of editing a TEX document file can be enhanced significantly if we can edit the TEX file in close coordination with the viewing capability of the corresponding TEX dvi file linked dynamically to the editor.

There can be two approaches for the realization of this scheme. The first method leads to developing a special editor which is capable of displaying a TEX-processed result. The second method respects the user's preference for a general-purpose editor, opting for its enhancement with the efficient viewing capability of the TEX dvi files in the X window screen.

The advantage of the former is that the designer of the editor has ample freedom to bring in the novel features desirable both for presenting the dvi view on the screen and for editing the TEX source being worked on. The VORTEX project by M. A. Harrison's group is a notable example adopting the first approach.

On the other hand, it may be equally advantageous for many people if they could use an editor with which they are familiar, provided it is equipped with an interactive dvi viewing feature. This paper describes a simple scheme of the second category targeted to those people who prefer Emacs or one of its derivatives as their sole editor for everything, including TEX sources.

This scheme of synchronizing an editor of the finest breed, of Emacs-type to be specific, with an acclaimed previewer will help improve, among others, editing sessions for large TEX documents. A typical document written in TEX, say a book manuscript, may consist of many files. These may form a tree structure through a multi-layered \input hierarchy, based on the logical divisions.

With the traditional editing style using the Emacs editor, particularly when the document consists of many files forming complex \input layers, a laborious cut-and-try search to single out a file from many is almost inevitable before locating the given passage. In any event, the cursor in the editing buffer window would have to be moved to all these places more or less manually.

In contrast, our scheme eliminates most of these time-consuming chores, and a single mouse click is all that is needed.

Outline of the Operation

In short, the editor/previewer combination does the following:

a. Any TEX source file or a chained cluster of them, be it the root, a node, or a leaf of the \input tree, can be previewed without compiling the entire TEX tree.
b. The number of generations of \input files to be included in a partial compilation for previewing can be limited to a user-specified depth, both in the direction of descendants and of ancestors.

In so doing, the fundamental characteristics of the document will still be preserved and reflected on the pages shown on the screen, even if its root file could have been curtailed.

c. Selection of the compiler, TEX, LATEX or LATEX2ε (LATEX3), is automatic.
d. The cursor of the Emacs-type editor jumps to the line of the TEX source file corresponding to the location specified by clicking the mouse on the display screen of the previewer.
e. The editor accepts interactive commands from the user while the previewer is active on the display. That is, both of them coexist, and there is no need to terminate the previewer to regain control over the editor.

Combination of Two Tools

A straightforward way to achieve this scheme is to select an editor and a previewer from among the tools most frequently used. The combination of the Emacs editor (or a close cousin) and `xdvi` would surely be acceptable to the majority of users, particularly those in the academic and scientific communities where we find a heavy concentration of devoted TEX users.

The present paper is based on our experience in implementing this scheme for two kinds of Emacs-type editors: the original Emacs editor, `GNU Emacs`, and one of its derivatives, `Njove`.

The latter, having been the subject of development for some years at Fukui University, is based on Jonathan Payne's `JOVE` (*Jonathan's Own Version of Emacs*). Amongst its many unique editing features not found in the original `JOVE` or in Emacs, `Njove`'s TEX mode is an attractive asset for editing (LA)TEX files. Like `JOVE`, `Njove` is written entirely in the C language.

Except for the ways the new editing commands are added to the main body of the respective editors, the two version are almost identical. For `GNU Emacs` this portion is written in Emacs lisp.

`Njove` has been the primary testbed for new ideas in this project because of the present authors' familiarity with its internal details. As such the `Njove` version is, at the time of this writing, in a slightly more advanced phase of development. Some of the minor implementation details (such as the coding method of inter-process communications) to be described in what follows may reflect, therefore, those of the `Njove` version. Nevertheless it is hoped that the word `Njove` can be read as indicating a generic Emacs-type editor, including `GNU Emacs` itself.

To ease portability, a substantial part of the program consists of modules that can be run as parallel Unix processes, isolated from the editor itself.

The Previewer

`Njove` permits previewing the whole or part of the file being edited using a modified version of the standard `xdvi`. (To distinguish it from the original version, the modified version will henceforth be referred to as `xdvi+`.) `Njove`'s text buffer window and the `xdvi+` TEX viewing window are shown side by side on the screen.

When `xdvi+` is activated, it displays the image of the specified `dvi` file on the X window screen. `xdvi+` scans the `dvi` file sequentially, and places each character glyph or rule on a page one by one, just as any `dvi` device driver does. Simultaneously with drawing each page, however, `xdvi+` keeps track of the locations it encounters by using `\special` commands that have all a valid argument string (*parsing message*) with the following format:

`loc` *source-file-name source-line-number*

A correspondence table is created anew for each update of the displayed page. The table records the correspondence between this locational information (the x- and y-coordinates) of the document page and that for the source file, namely the source file name and the line number. The table can accept a generous amount of `\special` commands (by default, up to 4096 entries per page).

Each time `xdvi+` detects a mouse event for the page and identifies it as the newly implemented `xdvi` instruction to locate the source file, `xdvi+` searches for the closest tagged location upstream in the document from the point the mouse click occurred. The source file name and the line number are identified by consulting the correspondence table for that tag entry.

Upon notification by `xdvi+` about this information, `Njove` switches the displayed content of the editing buffer promptly to that of the (possibly newly opened) target file, and moves its cursor (that is, "point" in Emacs jargon) to the beginning of the line which is most likely to contain the passage the user specified with the mouse on the TEX preview screen. Incidentally, the buffer is ready to accept any editing command all the time.

The coordination between `Njove` (or its "agent", `texjump`, to be more rigorous, as will be discussed in a moment) and `xdvi+` can be outlined as follows:

1. Establish a link between Njove and xdvi+, so that they can communicate with each other in real-time in a typical X-window environment.
2. Let xdvi+ pick up the positional information where the mouse click event took place.
3. Interpret the click position and notify Njove of:
   a. the source file name, and
   b. the source line number.
4. Let Njove "find" the specified file, and position the cursor at the beginning of the designated line.

A New Editor Command

To integrate the interactive previewing capability of TEX's dvi file into the editor, a new command tex-jump was added to Njove.

When the Njove command

Esc-x tex-jump [*option switches*] [*target-file*]

is issued, Njove spawns a separate Unix process texjump, independently of the editor. (The presence or absence of the hyphen in the name texjump is used to differentiate between these two closely related but clearly distinct entities.) If the file name is not specified, the file associated with the buffer of the window, from which the command was issued, is selected as the default *target-file* (to be described later). Optional switches may also be specified. These are identical to the ones for texjump as a shell-executable command.

The standard I/O of texjump is connected to Njove via a pair of *pty*s, and its output stream is eventually sent to and stored in the newly created Njove buffer named "*texjump*".

texjump in turn spawns xdvi+. They communicate with each other through a Unix pipe. For each mouse click in the preview screen, xdvi+ sends back to texjump the locational information of the source file through the Unix pipe. texjump thereupon outputs a grep-like message (*parsing message line*) to the standard error stream, which Njove accepts through its *pty*.

Until texjump is eventually terminated, Njove intercepts all the input streams to its various buffers scrutinizing a stream destined to the buffer *texjump*. If a parsing message for texjump is found, Njove subsequently lets its newly added function ParseErrorOneLine() parse that single line, and displays the pertinent buffer (or opens a new file if it is not yet assigned to any of the existing buffers) in an appropriate working window, and lets its cursor (point) move to the beginning of the line specified by xdvi+ ("*point positioning*"). At the same time, the successive parsing message line is appended to a special buffer "*texjump*".

Positioning on the Screen

Whenever the left button of the mouse is clicked while holding down the control key of the keyboard at the same time, xdvi+ determines the corresponding current location in the source file, and transmits it to texjump. Njove, receiving this information from texjump through *pty*, selects the relevant buffer and advances the point to the beginning of the requested line.

Users user can pick any location at any time asynchronously until they quit xdvi+ with the q command.

When the command tex-jump is issued, Njove switches to the active state of "*error parsing*". Then Njove is ready to accept parsing commands from the keyboard. They are next-error (*C-x C-n*) and previous-error (*C-x C-p*), respectively, which step the point in the buffer *texjump* either one parsing message line downward or upward, followed by a new *point positioning*. (In the case of GNU Emacs, next-error is key-bound to *C-x* `, while previous-error is missing.) This active status persists even after xdvi+ is terminated through its *quit* command. Issuing *C-x C-c* finally lets Njove exit from its error parsing status.

The Tree Structure of TEX Source Files

A typical TEX document may be composed of multiple TEX source files forming a tree structure by means of the \input feature. Let its root file be *root*.tex.

A new tool, textree, analyzes the tree structure of the document by tracing recursively the existence of \input or \include commands. textree expects a single argument in the command line, the root of the document tree.

```
% textree root.tex
```

textree generates a file, Tex_Input_Tree by default, which indicates the mutual input dependency relationship of the document in a format akin to what the Unix make command understands. Therefore, as a byproduct, the created file, Tex_Input_Tree, may also be used to write the dependency rule of a Makefile for all sorts of (LA)TEX compilation in general.

Derivation of the Source Position from the DVI File

Since the TEX compiler does not leave any trace of the locational information about the original TEX source files in the dvi file, ordinary dvi device drivers have no way of correlating an arbitrarily chosen point on a processed page of the document with the specific file among a number of TEX source files forming that document, and the word/line position within that file in particular.

Therefore some means of forwarding the locational information to the device driver has to be incorporated. There are two alternatives:

The most straightforward scheme would be to modify the (LA)TEX compilers such that they either

- include the locational information of the source files within the dvi file they generate, or
- generate an additional auxiliary file which contains the information about the location in the document pages of the items found at the beginning of all the source lines.

One could envisage introducing a parallel to the optional switch -g found in the C compiler, used to add extra information for source level debugging. While there is no doubt as to the technical feasibility of this scheme, and obviously it is the most rational and robust of the two alternatives, in real life the modification had better be incorporated into the official circulating version of all the compilers by their original authors, lest the introduction of yet other variants go astray from the spirit of unification of (LA)TEX.

Although we would very much like to have this feature in future releases of the (LA)TEX compilers, we will look for another alternative that offers a practical solution for the time being. This approach uses the (LA)TEX compilers as-is, without any modification. It generates copies of the source files, and additional information (a "positional tag") is inserted into these files automatically prior to the (LA)TEX compilation. The positional tag is inserted at every "landmark location" of the source files, say at every location where a new paragraph begins ("paragraph mode"). Or it could as well be at the beginning of each non-empty source line ("line mode").

The applicable positional tags must never distort the original content of the document. Two TEX commands, \wlog and \special, satisfy this criterion.

One can insert a \special command with its message text consisting of:

- a unique ID code (default: loc) to distinguish this particular usage of the \special command from others;
- the source file name;
- the source line number.

This is the scheme adopted in texjump.

By comparison, one could insert a \wlog command, instead of \special, as the positional tag. The preprocessor (that is, the equivalent of texjump) would then generate the message text for \wlog as an ASCII string indicating the location as the line number of the source file at the point it inserts the \wlog command. With the help of a simple program that would analyze the log file written by the the (LA)TEX compiler, the page boundaries of the printed document could be identified in the source files.

The advantage, if any, of using \wlog would be that neither the (LA)TEX compilers nor the previewer need to be altered, offering the user a much wider selection of previewers. This benefit would, however, be offset in most cases by the drawbacks, in comparison with using \special:

- The editor can control the previewer page, but not vice versa, because the unmodified version of the previewer cannot communicate back to the editor.
- The positioning resolution one can expect cannot go beyond the page of the document displayed on the previewer screen.

Line Mode versus Paragraph Mode

texjump accepts two options to select the way \special commands are inserted, namely line mode and paragraph mode. When line mode is chosen, a mouse click in the previewer window can locate the source position within the range of a line or so. In paragraph mode, however, we deal with a scope no finer than the size of the paragraphs involved. The main motivation for paragraph mode comes from the need to make texjump much more robust if line mode fails for reasons discussed below.

Line mode. In this mode \special is inserted at the beginning of each non-empty line without otherwise altering the original context of each line. Since the original line number assigned to each line remains valid after the insertion, the dvi driver can identify the correct line number in the original source files, even though it extracts the data from a single file, namely the dvi file created by compiling the modified copy files containing the scattered \special commands.

Paragraph mode. In this mode `\special` is inserted exclusively at the beginning of the first line of each "paragraph". `texjump` recognizes a cluster of one or more empty lines as the paragraph delimiter. (Note that the definition of a paragraph is different from that of TEX or Emacs.)

Problems Associated with Tag Insertion

Even though a `\special` command supposedly causes no appreciable side effect other than merely forwarding a character string to the `dvi` driver as a communicative message, it does not mean we can insert it indiscriminately in any arbitrary position of the given source file.

As a typical example, consider the case of a TEX macro which expects one or more arguments, and there occurs a line break in the source file just in front of one of its arguments. One cannot insert the `\special` blindly at the beginning of the following line which starts with the expected argument.

For instance, within a `\halign` construct, the line with `\noalign` rejects `\special`. If the construct's final line begins with its outermost closing brace (}), `\special` is not permitted.

A more obvious example is LATEX's verbatim environment, or its TEX equivalent. Insertion of a `\special` in the lines belonging to this environment does alter the content of the compiled document because there `\special` is nothing more than a plain character string. Needless to say, `xdvi+` does not identify the "argument" as positional information.

Therefore `texjump` has to know about lines where `\special` insertion should be avoided. This means that `texjump` must be able to, ideally speaking, analyze the syntactical structures.

Realization of a full scope syntax analysis would be equivalent to almost fabricating a new (LA)TEX compiler. This kind of duplicated effort would not be justifiable, because the modification of the compilers mentioned before is clearly the rational way to do it. The current version of `texjump` analyzes, therefore, the syntactical structure of the source files only superficially.

If the (LA)TEX compiler complains about a syntactic error that originated from the insertion of the `\special`, the user may either switch to paragraph mode, which is more robust than line mode, or modify slightly the original source file, as will be discussed below, by adding some directives to `texjump` in the form of comment lines for the TEX compiler.

Where to Attach the Positional Tags

Since (LA)TEX refuses to accept the insertion of a positional tag at certain places, we have to discern these syntactically inappropriate circumstances. The current version of `texjump` interprets the syntactical structure superficially. Therefore it recognizes only the most obvious cases.

Tags are not attached to the beginning of the following lines:

1. a blank line, or a comment line;
2. within a verbatim environment;
3. from the line beginning with `\def` till the following blank line;
4. a line which begins with }, `\noalign`, `\omit`, `\multispan`;
5. the line following a non-blank line ending with %;
6. the preamble and postamble part of each file, if any;
7. lines for which explicit instructions tell `texjump` not to attach a tag;
8. each non-first line of each paragraph when in paragraph mode.

Otherwise the positional tag is attached to the very beginning of each line.

Manual Control of the Tag Insertion

`texjump`'s algorithm for inserting positional tags works reasonably well for relatively simple TEX documents. For documents of a complex nature, however, one can only expect it to be marginally smart.

When `texjump` stumbles into a pitfall, particularly in line mode, some `texjump` directives can rescue it. Inserted manually in the source file by the user, they let `texjump` avoid potential hazardous spots in the file.

Each directive is a TEX comment line with a predefined format. It consists of a line beginning with three % characters followed by a symbol.

%%%<	Enter paragraph mode.
%%%>	Exit paragraph mode.
%%%!	Skip tagging the ensuing single line.
%%%~	Skip tagging until the next blank line.

File Inclusion

`texjump` lets the user specify the range of files to be included through three parameters:

1. the filename under consideration (*target-file*);
2. the number of generations, in the `\input` tree, corresponding to:

a. its ancestors from there upstream (*ans*);
b. its descendants from there downstream (*des*).

The default is *ans* = 0 and *des* = ∞; that is, the *target-file* and all of the files it includes in a cascade downstream.

`texjump` first looks for the file `Tex_Input_Tree` in the current working directory, and obtains from it the tree path which reaches the root file from *target-file*. If `Tex_Input_Tree` is missing, *target-file* is assumed to be the root file.

As for ancestors, inclusion is limited to only those files directly on that path. No siblings of the *target-file* or of its ancestors are included. If *ans* is smaller than the generation number to the root file, some of the files closer to the root, including the root itself, will be out of range for the file inclusion scope. `texjump` inspects the content of each of these files, and if any of them contains the preamble and/or postamble, these portions (not the entire file) are all extracted for inclusion despite the scope rule.

The file inclusion rule for the descendants is much simpler. If *des* is specified as the option parameter to the `texjump` command, up to *des* generations of direct descendants of *target-file* are included. Otherwise, all of its descendants are included.

`texjump` suppresses the file inclusion simply by altering the string `\input` or `\include` to

```
\par\vrule width 2em height 1ex
            \qquad{\tt \string\input}\quad
```

in the same line, which generates a line like

■ `\input` *input-file-name*

on the preview screen, thus making it clear that the `\input` command line is there.

Preambles and Postambles

`texjump` assumes that each file consists of three parts:

1. an (optional) preamble;
2. the main body;
3. an (optional) postamble.

The preamble, if any, is an arbitrary number of lines at the beginning of the file bounded by two lines with the unique signatures:

```
%**beginning_of_header
```

and

```
%**end_of_header
```

Likewise the postamble might be at the end of the file, bounded similarly by the lines:

```
%**beginning_of_tailer
```

and

```
%**end_of_tailer
```

.

The root file is exceptional in that both explicit and implicit definitions of both the preamble and postamble are permitted. For those files which do not have the above-mentioned signature line for preamble initiation, if the line `\begin{document}` is encountered within the first 100 lines then the region from the first line to this implicit preamble terminator line is treated as the preamble. The same is true with the postamble. The implicit postamble in most cases is from a line which contains

```
\end{document}
\bye
```

or

```
\end
```

till the very end of the file.

`texjump` inserts positional tags neither in the preamble nor in the postamble. Therefore any TEX codes, critical for the document but irrelevant for positioning, should be placed inside these regions. TEX macro definitions, variable parameters setting, or inclusion of system files are typical examples.

It should be noted that both preambles and postambles of all the files involved in the TEX tree are always included, irrespective of the scope rule.

Source Recompilation

Since `texjump` keeps showing the very same `dvi` file, and therefore the recent modifications are not reflected on the viewing screen, updating the screen may become desirable after some modification of the source files. Taking into account the time (LA)TEX takes to compile, however, it may hamper efficient editing work if we let `texjump` decide to initiate automatically the recompilation of the latest source files over and over again even at sporadic intervals. Therefore, unless the user instructs `xdvi+` to do so explicitly, recompilation does not take place.

Sending a `C` character to the `xdvi+` window signals it to perform a recompilation. `xdvi+` conveys to `texjump` the acceptance of the user's request and waits for the renewed `dvi` file. When available, it redraws the screen using the new `dvi` file.

Intermediate Files

Since our scheme modifies the content of the (LA)TEX source files, we must work on the copied files. Therefore `texjump` first creates a new (temporary) working directory with the user-specified path-name. It then reproduces there the entire

directory structure of all the files involved, taking into account the file inclusion rule.

texjump identifies the "local root" for those files which fall outside of the clusters stemming from the original current working directory. texjump assigns to each of them a subdirectory in the above-mentioned working directory and gives it an arbitrary name. texjump keeps the entire record of the file mapping between original and copy.

In this manner, the intermediate files are effectively hidden from the user, thus creating the impression that one is dealing directly with the original files. In reality, what the previewer is showing on the screen, and giving the positional information for, corresponds to the copied files, while what the editor is showing in its window are the genuine original files.

In order to take this hiding process a step further, even the (LA)TEX compiler is manipulated by texjump on purpose. When the (LA)TEX compiler detects an error in a source file, the user usually calls for the editor by responding with an "e" character. The compiler then transfers control to the user-specified editor. It instructs the editor to open the temporary file, because this is where it found the error. texjump, however, swaps the shell's environmental variable $TEXEDIT temporarily with a fake editor, texjump_ed, just before (LA)TEX starts compilation of the tagged files. Therefore it is texjump_ed which receives information about the file (path-name) and the line number. Its sole role is to identify the original source file from the received information, and then to call in the real editor the user had requested, acting as if (LA)TEX had performed that job.

In order to enforce integrity we minimize the possibility of confusing the original and the copies by deleting the temporary subdirectory created by texjump each time xdvi+ is relinquished after previewing.

Option Switches

The Unix shell can execute texjump as a stand-alone process. It expects a (LA)TEX source file name, and optional switches may also be specified.

% texjump [-*opt* [*num*] [, ...]] *target-file*

If *target-file* does not specify a filename extension, texjump assumes it to be .tex.

Valid option switches -*opt* are:

-h Displays the entire list of switch options.

-p Instructs texjump to treat all files in paragraph mode.

-n *num* Lets the (LA)TEX compiler repeat the compilation *num* times (default is 1).

-t *num* Start from the ancestor *num* generations upstream in the input tree. *num* = 1 specifies that the parent immediately above the source file (*target-file*) should be included (default is 0, i.e., no ancestor is included).

-b *num* Include the \input files down to *num* generations of descendants. *num* = 1 means only the "child" files, included directly through an \input command in the source file (*target-file*), are to be included.

Generation of xdvi+

The source files for xdvi+, the extended version of xdvi, are generated through applying a patch to version 17 of xdvi. It modifies five files, Imakefile, dvi_draw.c, tpic.c, xdvi.c, and xdvi.h, and adds a new module, jump.c.

imake generates a Makefile, which takes care of the entire process of creating xdvi+. Note that xdvi+ preserves all features of the original xdvi.

Customization

Users can specify some of the critical parameters controlling texjump. They are described in a configuration file, whose default name is .texjumpcfg (this can be altered at installation time). texjump looks for this file successively in the current directory, then in the user's home directory, and finally it uses the parameters in the system default file. It specifies to texjump the choice of:

1. the temporary working directory to be created (and removed subsequently);
2. the previewer;
3. the name of the \input tree file generated by textree;
4. the (LA)TEX compiler;
5. the signatures signaling the end of the preamble part;
6. the signatures signaling the beginning of the postamble part.

An example of the default .texjumpcfg parameters is shown below:

Parameter	*Default*
TEXINPUTS	.:/import/TeX/inputs//\ :/import/TeX/lib//
WORKDIR	texjump-workdir
XDVI	xdvi+
XDVI_OPTION	-s 3

```
TREE            TeX_Input_Tree
TAG             loc ${FILE} ${LINE}
TEX             /import/TeX/bin/tex
LATEX           /import/TeX/bin/latex
LATEX2E         /import/TeX/bin/latex2e
FOILTEX         /import/TeX/bin/foiltex
BOH             %**beginning_of_header
EOH             %**end_of_header
BOT             %**beginning_of_tailer
EOT             %**end_of_tailer
BOT_TEX         \bye
                \end
EOH_LATEX       \begin{document}
BOT_LATEX       \end{document}
BEGIN_PAR_MODE  %%%<
END_PAR_MODE    %%%>
EXCLUDE_NEXT    %%%!
EXCLUDE_BLOCK   %%%~
SEL_TEX         %**plain_tex
                %**tex
                \input eplain
SEL_FOILTEX     \documentstyle{foils}
                %**foiltex
SEL_LATEX       %**latex
                \documentstyle
SEL_LATEX2E     %**latex2e
                \documentclass
```

Conclusions

(LA)TEX source files can be edited using an Emacs-type editor, say GNU Emacs. By clicking the mouse on an arbitrary page of the xdvi preview screen, the cursor (point) moves directly to the interesting spot in the Emacs window, that is displayed next to the xdvi window.

Two prototype versions, for Njove and GNU Emacs, are currently operational on workstations running the 4.3BSD and SunOS operating systems. Porting the software to other Unix platforms is expected to be straightforward. texjump assumes that the presence of the standard GNU development environment on the target machine. The program, written in the C language, can be compiled with GNU gcc.

The program will be available through anonymous ftp at i1nws1.fuis.fukui-u.ac.jp in the directories texjump, xdvi+ and textree under /pub/tex/.

Njove is a bilingual editor, which supports both English (using the single-byte ASCII character code set) and Japanese (using the two-byte Japanese character code set). It can be found in the directory /pub/editor/njove at the same ftp site.

Acknowledgments

The authors thank Takayuki Kato for his contribution in making texjump worthy of real-world applications through improving its functionalities. Jun-ichi Takagi wrote the first rudimentary interface module for GNU Emacs using Emacs lisp.

Bibliography

Harrison, Michael A., "News from the VORTEX Project", *TUGboat* **10**(1), pages 11-14, 1989.

Cooper, Eric, Robert Scheifler, and Mark Eichin, "xdvi" on-line manual, June, 1993.

Payne, Jonathan, *JOVE Manual for UNIX Users*, 1986.

Kawaguti, Minato, "Dynamic Filling with an Emacs Type Editor", *Proc.* **jus** *10th Anniversary Intern. UNIX Symp.*, Japan UNIX Soc., pages 49-58, 1992.