♢ ♢ ♢

Visual TEX: texlite

Igor Strokov

abstract. A prototype of a visual TEX is implemented by means of minor modifications of canonical TEX. The changes include the ability to start compilation from an arbitrary page, fast paragraph reformatting, and retaining the origin of visual elements. The new features provide direct editing of the document preview and correct markup of the source text.

keywords: visual, TEX

The need for visual TEX

A good feature which TEX traditionally has lacked is visual editing, or the ability to typeset a document in its final (print preview) form. Though one can manage without visual editing, there are certain cases when it really helps, mainly when tuning the appearance of a document, especially for novice and occasional users.

There are two ways to deal with this problem. The first one, implemented in lyx and SciWord, represents the source text in a form which logically resembles the resulting output. Although this way proved to be a good compromise, the logical preview is often quite far from the printed result. Besides, these tools use latex with special macros, so arbitrary TEX documents are beyond their scope.

The other way is synchronously running a source text editor, TEX the compiler, and a viewer, so that changes made in the source editor are compiled and displayed in the viewer without explicit invocation of these tools in a command string or a menu. In textures, in addition, the cycle is closed by means of two corresponding pointers in the source editor and the viewer. The approach of Jonathan Fine, presented in these proceedings [2], can be related to this way. Its main problem lies in the need to compile a whole document to receive the visual response to an editing action. On slow computers, large documents, or complicated macros this compilation could cause at least a noticeable delay.

How to TEX part of a document

So, if even a single letter is changed, TEX needs to process the whole document from scratch. However, TEX already solved a similar problem by loading precompiled macros (so called ‘formats’). This way TEX saves the time required to compile standard

macros which can be considered a part of a document. With minor changes one can extend this method to arbitrary stages of the document compilation. One only need to choose particular stages and probably improve the storage method.

The natural decision is to make a core dump each time a typeset page has been completed and removed from TEX's memory. At this moment the memory is relatively empty and contains essentially values which do not change from page to page, which allows the dump to be more compact. A page dump is always stored as a record of differences from some reference dump. Every 8-th dump refers to dump zero (the 'format' itself) while the others refer to the preceding 8-th one (see [3] for details).

In comparison to a common format the core dump has to store additional data: the input and semantic list stages, open file pointers, and, above all, the source line number reached when the page was completed. Thus, if the source is changed at some line, one can derive the last page number affected by the change. Then the corresponding dump is loaded and the compilation starts from the given page.

In texlite, the visual TEX prototype, the compilation is allowed to run up to the end of the document. Moreover, if any output file to be read again is changed, then the compilation starts again, this time from the very beginning (this happens, for example, when a LATEX section title is edited after the table of contents). However, this already does not matter, as the page of interest is obtained quite quickly (in constant time, regardless of the page number and the document size). Although in some mentioned cases the page is updated a second time, in practice this event does not disturb a user because of buffered output and preservation of the current input position.

Besides, if another source change will occur before the natural end of the compilation, it will be interrupted to let a more actual compilation run. As a result, all recent editing actions are reflected in time, while the remote consequences (if any) appear after a pause.

How to reformat a paragraph

Selective compilation is a necessary but not sufficient feature for visual editing. It just reduces the response time from the source to the preview (making it constant instead of linear). Let us consider the inverse problem — how to bring preview changes into the source text.

In general, one needs to keep track of the relation of source characters to visual elements. In many cases the reciprocal relation (say, a character — a glyph) may be established and used to synchronize current positions in the source and the preview. Fortunately, no vast interference in TEX is needed to make it remember the origin of its output: it is enough to extend the format of memory nodes in a way similar to that used for breaking the 64K barrier.

So, a user may mark the current position in the preview and type something there. Texlite applies the corresponding changes in the source text and initiates the compilation starting from the current page. In most computers (starting from a 200MHz Pentium) the delay before the visual result is virtually unnoticeable. However, one cannot ignore slow computers and various decelerating factors such as parallel pro-

cesses, complex page formatting, complicated macros, etc.

Therefore, before starting the true compilation, texlite tries to reformat the current paragraph using the native TEX algorithm for this purpose. Canonical TEX, however, does not keep parameters it used to set up boxes and paragraphs. That is, one cannot correctly rebuild a box or a paragraph only from its contents. Texlite resolves this problem by storing the necessary data in special *whatsit* nodes. This does not take too much extra space, as many parameters (penalties, glues, *parshape*, etc.) remain the same throughout a document and thus can be omitted.

Let us see what happens when a user edits the preview. First of all, texlite decides (with the aid of whatsit nodes) in which paragraph, if any, the current position falls. If no paragraph is recognized (for example, it may be within a `\halign`), then only the enclosing box is rebuilt and the selective compilation starting from the current page is initiated. Otherwise texlite locates the current paragraph and unwraps it back into a *hlist* by inserting lost glues and repairing hyphenations. The unwrapped list is subjected to the changes following from the user's input (insertion of a character-and-glue node list or deletion of several nodes from the current position) and the *linebreak* routine is called to rebuild the paragraph. Then the preview (along with the current position) is updated.

After this 'emergency repair' texlite enters the source text, performs parallel changes there and starts the selective compilation, which runs from the current page to the end of the document or until the user presses another key. If the compiler manages to build the current page before the next key stroke (usually it does) and the new page happens to be different from the repaired one (usually it does not) then the preview is accurately updated.

Implementation

Texlite, currently implemented under Win32, provides a visual shell for the TEX core tangled from TEX: The Program [1] and modified in the way described above. Let us enumerate the main changes:

1. It can make detailed core dumps and read them back at specified points.
2. For every paragraph it stores all the data required to unwrap the paragraph back and break it into lines again.
3. It relates nodes in memory to their origin in the source text.

Except for these additions the compiler remains the canonical TEX, able to process an arbitrary TEX document.

The shell consists of two windows. The preview window represents the current page in its typeset form, which is common to all DVI viewers. The main difference is an editing cursor (a flashing caret). The user may mark, type, or delete pieces of formatted text as in any WYSIWYG text processor, without delays or other inconveniences. However, the requirement to keep a correct document structure still imposes some limitations. For example, one cannot select a part of a heading and several words in a following paragraph and delete them at once. Texlite prevents such inconsistent

changes by tracking grouping levels.

Texlite preview supports graphics in several bitmap formats and in PostScript (installed GhostScript required). Standard LATEX coloring schemes and emTEX extensions (popular in the DOS/Windows community) are supported too.

In addition, texlite provides an extension for internal and outer links in a document. For example, a specific macro makes a common table of contents into an interactive directory which opens the corresponding page when a section reference is clicked. An outer reference brings a user to a different document located elsewhere on Internet. This feature facilitates browsing of TEX documents and makes it similar to web surfing.

Though a user may choose to work with a document in the preview window only, there remains a window for the source text. All the changes made in the preview are automatically reflected here. However, if a user edits the source text, he should invoke the synchronization explicitly, because manual correction of the source text allows the existence of inconsistent clauses.

The source text view benefits from visual editing too, as it uses information from the compiler to mark up the text according to the TEX syntax. Moreover, the markup (presented in different colors) is always correct, even if symbol categories are changed during the compilation.

A small improvement concerns error corrections. The shell sets the cursor to the position in the source text where an error occurs and displays its description. However, if a compiler is used just to mark up the text, errors are only marked by a color.

REFERENCES

[1] Knuth, D. E. Computers & Typesetting. Volume B, TEX: The Program. *Reading, Massachusetts: Addison-Wesley, 1986.*

[2] Fine, J. Instant Preview and the TEX daemon. *These proceedings.*

[3] Strokov, I. I. A WYSIWYG TEX implementation. *TUGBoat*, December, 1999.