Experiments in TEX and HyperActivity

L. Carr, S. Rahtz and W. Hall
Department of Electronics and Computer Science
University
Southampton S09 5NH
email: L.Carr@uk.ac.soton.ecs

Abstract

Publishers are starting to explore ways of making their traditional books available on screens, using techniques grouped together as "hypertext," and have tended to concentrate on the problems of linkage and navigation. We believe that both structured writing and traditional typesetting skills have an important part to play in the screen-oriented books to come, and we present the results of various experimental systems which use TEX in the process of creating a hypertext document. Problems tackled include the use of TEX to format text being displayed on screen in variable sized windows, embedding generic hypertext navigation tools in the source and using a single source to generate both printed and hypertext versions of a document.

Intro

Ted Nelson (*Literary Machines* [1987] is a good introduction to his ideas) coined the term *hypertext* to describe "non-sequential writing," or works which could not be expressed as a simple linear sequence of their contents. This includes:

branching texts where the reader is presented with a choice between several paths to follow through the work

interconnected texts which make reference to other parts of themselves or parts of other, separate works

active texts which modify themselves according to some particular criterion, e.g., by recomputing one of its own tables of figures from the latest available stockmarket information.

multimedia texts which are composed of material from disparate information media (such as text, graphics, sound and video). These works are known more generally as "hypermedia" documents.

Hypertexts predate computers — novels frequently have concurrent threads of action (branches) as well as flashbacks (local interconnections) and literary cross-references (external interconnections; for example, we cannot read David Lodge's *Small World* without reference to a huge body of older literary forms) — but the computer has made hypertexts much more practicable in three ways:

presentation A reader may follow branching pathways and connections to other documents by selecting the appropriate material (usually with a mouse) and pressing a button.

speed The computer may retrieve information several orders of magnitude faster than a human can turn pages in a book or fetch a new book from library shelves.

information unity The computer acts as a control centre, allowing the many media available to it to be manipulated in a common fashion.

Hypertext systems are characterised by the kind of information that can be stored and the ways in which the information can be interconnected. Information is usually divided into discrete chunks (or *nodes*) which are connected through *links*. Some systems such as HyperTIES use fixed-sized nodes which only contain textual data, others (such as InterMedia, described by Yankelovich [1988]) allow nodes to contain arbitrary amounts of textual and graphical information. The linking capabilities vary widely: some link from a specific point within a node to a destination node, others allow a connection to be established between stretches of text within nodes. Some links also have information associated with them: a name, a type or a title.

The similarity between creating a document using a generic markup system like LATEX and creating a hypertext document is quite pronounced. In both activities the author is committing a set of ideas

and thought processes to a medium and attempting to arrange the information within the constraints of the structure provided by that medium. In LaTeX, the structure is that of technical authorship and is essentially *hierarchical* (chapters of sections of subsections and subsubsections) whereas a general hypertext document has no controlling structure, and is usually referred to as a *network*.

The aim of LaTeX is to allow the author to concentrate on the logical structure of the text and not to be concerned about details of its physical representation. Various different physical representations can be achieved by the use of the `\documentstyle` command. The LaTeX document is, in fact, a hypertext because it exhibits both branching (sections are composed of many subsections which are not necessarily meant to be read in sequence) and interconnection (through the use of `\ref`, `\label` and `\cite` commands), and so by the application of a different document style it should be possible to view a technical report or book as a hypertext network.

The next sections describe several attempts to use LaTeX as a tool for producing hypertext documents.

Two Approaches

The function of LaTeX is to take a logical structure (the document's contents with embedded markup) and from it to produce a physical structure (the device independent description of a set of pages). If a hypertext network is required instead then two options are available: to use LaTeX as either

1. a structural markup language which is to be interpreted by an independent system
2. a formatting engine for preparing the individual nodes of a hypertext network

The first alternative views LaTeX as a document interchange standard in competition with SGML (e.g. Bryan, [1988]) or Microsoft's document interchange Rich Text Format (RTF). The document is processed by a *pseudo*LaTeX which is mainly concerned with mapping the continous stream of text onto a network of discrete nodes which will be managed by an independent hypertext system.

An example of this process is Southampton University's Computer Science Prospectus as shown in Figures 1, 2 and 3. The prospectus was originally written in LaTeX so that printed copies could be sent to applicants but a requirement grew for an "interactive guide" to the Department (like that successfully developed for Glasgow Computer Science, and several other sites) and so information from the

CM348 **Text Processing**

Year 3 Semester 2 Slot 2

Course lecturers: Mr S P Q Rahtz, Prof D W Barron
Method of assessment: Examination 50% and Coursework 50%

Text processing is one of the commonest applications of computers; it covers simple word processing, computerised typesetting, digital font design and pure electronic manipulation, as well as the tools needed for manipulating unstructured text. This course will cover basic concepts of document creation.

Topics covered

- the distinction between generic markup and visual layout, with particular emphasis on the SGML standard
- software tools, such as intelligent structure editors
- the layout of printed text, and the 'rules' of page design
- the computerised design of typefaces, with a study of the METAFONT system
- page description languages, especially PostScript
- typesetting languages, concentrating on TeX
- the design of documents for the screen, and the principles of hypertext and database publication

Coursework will consist of work on software tools for manipulating structures, writing TeX macro packages and a group project on designing a font with METAFONT.

Figure 1: A Page From the Computer Science Prospectus

prospectus had to be merged into a HyperCard stack along with photographs and maps.

To do this a simple *pseudo*LaTeX interpreter was written in HyperCard's programming language HyperTalk (see Figure 4 for a brief extract). The interpreter scans through the document filling nodes with text and creating new nodes whenever a new structure (chapter, section or subsection) is encountered. Subsequent processing allows a table of contents to be built from the titles of each node and reference/label pairs to be resolved.

This approach is advantageous when the document in question already exists in LaTeX format, otherwise it would be more convenient to use a simpler markup scheme. The disadvantage is that specialised markup (such as tabular and math environments) cannot be dealt with automatically. Because the current version of HyperCard does not allow for the use of multiple fonts or font sizes within a field then none of the visual cuing provided by typographic differentiation is possible. This restriction is removed in competitors to Hypercard like Asymetrix *Toolbook* for Microsoft Windows 3, and is about to be remedied in new releases of Hypercard.

A successful example of a relatively complete and advanced pseudoTeX interpreter is the Gnuemacs "info" processor for producing help systems within the Emacs editor; this takes the source of software documentation written in a TeX-based (or LaTeX) generic markup (prescribed by the

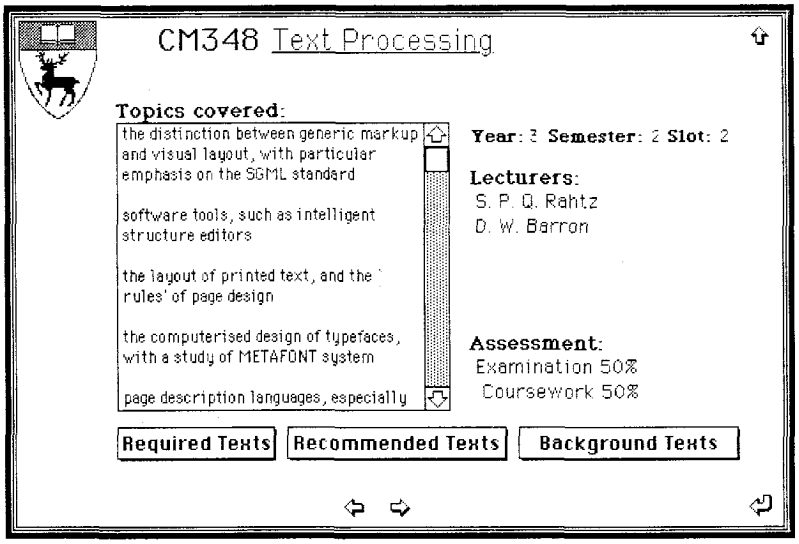

Figure 3: A Screen from the HyperCard Prospectus

```
\code{CM348}\year{3}\semester{2}\slot{2}
\coursetitle{Text Processing}
\headingsection
\lecturers{Mr S P Q Rahtz, Prof D W Barron}
\assessment{Examination 50\%
            and Coursework 50\%}

\topics

\begin{enumerate}
\item the distinction between generic
markup and visual
layout, with particular emphasis
on the SGML standard
\item software tools, such as
intelligent structure editors
\item the layout of printed text,
and the ``rules'' of page design
\item the computerised design of
typefaces, with a study of
the \METAFONT\ system
```

Figure 2: LaTeX source for Figure 1

```
repeat
    read from file "LaTeX input"
         until return
    if it is empty then exit repeat
    if char 1 to 9 of it is "\section{"
      then
        delete char 1 to 9 of it
        delete last char of it
        doMenu "New Card"
        put it into field "Title"
        put sectionNumber into field
             "sectionNumber"
        add 1 to sectionNumber
        put 1 into subsectionNumber
        put subsectionNumber into field
             "subsectionNumber"
    else if char 1 to 12 of it is
             "\subsection{" then
        ...
    else
        put it&space after field "Text"
    end if
end repeat
```

Figure 4: Simple PseudoLaTeX HyperTalk Script

Gnuemacs team) and reformats it to produce a set of "help" nodes in a hierarchical form comparable to a printed manual. Navigation tools are provided to move up and down and sideways in the tree structure, using the normal Gnuemacs editing commands in "read-only" mode. Alternatively, the same source can be put through LaTeX or TeX to produce a standard printed manual (and in this case effects like figures, tables or maths are activated, rather than being ignored by the "info" system).

The second of the listed alternatives makes use of "true" LaTeX to interpret all the markup in the usual fashion, making use of a modified document style to specify layout for an appropriately sized node in a standalone hypertext system. Once the device-independent file has been produced three different paths have been explored:

1. Use the `dvi2tty` program to produce the page as an ASCII plain text which is fed into a HyperCard field by a simpler HyperTalk program. In this case the logical structure has already been interpreted, so that the HyperTalk script only needs to turn the pre-existing page breaks into new nodes. The table of contents has been created by LaTeX, and all the cross-references have been resolved. Buttons and links may be added in a semi-automatic fashion based on the contents of the text.

   This approach has been used for *The Electric Rough Ground Farm* (Rahtz and Allen [1990]), implemented on an IBM PC under Windows 3 using Toolbook; an example page is shown in Figure 5. It is necessary, of course, to have a special document style to cope with the following problems:

   (a) page width; we can either decide how wide the field is to be, and format to that width, or we can assume the text will be wrapped by the application. But in the latter case, we lose LaTeX's careful layout.

   (b) we need to suppress running headers and footers, as they are irrelevant in this context;

   (c) it is almost certainly necessary to format using a fixed width font, so that the results will be predictable when converted to ASCII;

   (d) the ASCII codes are missing many useful ligatures and accented letters; getting a 100% useable result from `dvi2tty` is not easy, and requires some modifications to the source;

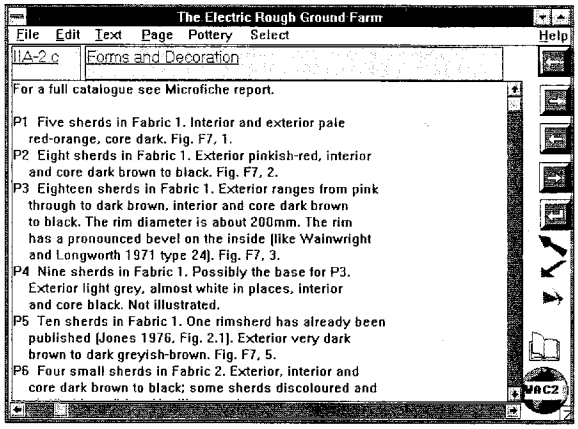

Figure 5: A page from *The Electric Rough Ground Farm* in *Toolbook*, formatted by LaTeX

   (e) tables and mathematics cause hopeless problems, to which there is no obvious solution.

   A variant approach adopted in one experiment with archaeological excavation reports (Rahtz, *et al.* [1989]) was to run the LaTeX source through a program to convert it to `nroff` input, using the *mm* macros. The generic structures can be translated unambiguously, and `nroff` can be used to create ASCII output, from which we can proceed in the same way as if we had followed the `dvi2tty` route described above.

2. Save the images from a `dvi` previewer, and use the results as a graphic to cover each node in the hypertext document. Thus we can use `dvipict` to create a bitmap for a HyperCard stack, or Eberhard Mattes' `dvimsp` to create a Windows Paint file for Toolbook (as in Figure 6). This has the advantage of faithfully reproducing the page design (including fonts and spacing) but loses both the structure and the actual text itself. No editing or searching can take place, and buttons and links have to be inserted manually. A solution demonstrated by Wilkins [1990] was to keep the generic LaTeX source behind the scenes in the Hypercard stack, and use a parallel bitmap to show the reader. The underlying ASCII text was used for searching and determination of location, but was never presented to the user. This approach is elegant, in that the source and the "compiled" `dvi` version are kept together, but suffers from the major drawback that TeX has to be invoked to reformat pages if any edits are made (and it is not clear what model to adopt

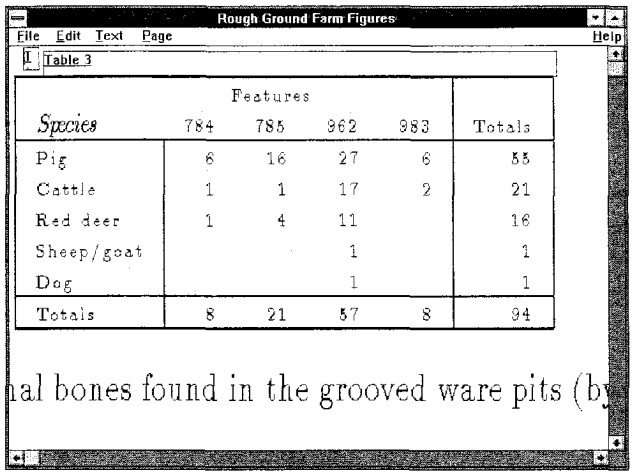

Figure 6: Part of table from *The Electric Rough Ground Farm* in *Toolbook*, showing a bitmap from `dvimsp`

for allowing the reader to choose the text to edit).

3. Use a dvi previewer that has been enhanced to deal with hypertext nodes and links. In this case no conversion process is necessary to accommodate the document within a foreign system. This is the route which we have used in the LACE system (described in further detail in the next section) by developing a `hyperdvi` program which converts TEX's device independent format into a form suitable for manipulation as part of a hypertext network.

LACE

At the simplest level, LACE[1] provides a TEX previewing service. Unlike the other experiments described above, LACE is implemented on a SUN workstation using the NeWS (now OpenWindows) windowing software. Based on the PostScript language, this allows compatibility between the formats of documents and pictures that are previewed on-screen and printed on a LaserWriter. A very similar environment exists on the NeXT workstation, but requires greater effort for a proposed implementation for the X Window system, as the X "page description" is lower level than PostScript.

[1] LACE is *not* an acronym for anything; it is partly a pun on the author's initials but more importantly is supposed to reflect the beauty of a structure of thousands of tiny knots of thread forming a coherent and structured whole.

One of the main goals of the LACE project is that there should be little (if any) alteration to the *authoring* process so that a large body of existing documentation can be used unmodified. This goal has been achieved by using a "hypertext" document style option which makes the following changes to the standard LATEX environment:

1. The definitions of all structuring commands are modified to add markers to the `dvi` file, so that they can be extracted from the formatted output.
2. A new command `link` is defined. This takes two parameters—the first is a piece of text over which a button will be placed, the second is the LACE address of a document part. For example, the command `\link{see section 3}{me:section 3}` will make a new window in which this section will "pop up" when the user clicks on the (transparent) button over the text "see section 3."
3. The table of contents, list of figures and list of tables all have buttons over each of the lines, so that clicking on any line will bring up a window with that part of the document in it.
4. The definitions of table, figure and footnote have been changed so that they take up *no space* on the page, but instead inhabit a separate window that is brought up when pressing a button over a reference to them. For example, a footnote window is brought up when the reader presses the button over the footnote marker in the main body of the text. Similarly a figure window is displayed when the reader presses a button over some text like "see figure 3."
5. A new environment *aside* is defined. This behaves very much like a footnote: the LATEX fragment

```
\begin{aside}{Click Me For More
                Information}
Green Hypertext allows texts to
be re-used and has no long term
effect on the environment
\end{aside}
```

   will produce a button over the words "Click Me For More Information" which, when pressed, will bring up a window with the text that is in the body of the environment.
6. The `label` and `ref` commands have been extended in the form of the `LACElabel` and `LACEref` pair. These two commands differ from the standard LATEX forms in that they

save not only the *number* of the current environment, but also its *type e.g., section 3.4* or *table 2*. As well as doing this, the reference text has a link to the item it references.

7. Any use of the `cite` command to produce a bibliographic citation in the main body of the text has a button over it that brings up a window containing the full bibliographic entry.

When the authoring cycle is completed the document is *published* by entering it into a host-wide database of documents which is presided over by a librarian process, the LACE daemon. The database holds information about each document: its title, keywords, access permissions and location in the host's filestore. Each document is considered to be a database of logical elements (in the structural markup sense). The LACE daemon and the `hyperdvi` process cooperate to provide access to any addressable element of any document published on the local host. Requests for document parts are of the form `<document specifier>:<element specifier>` where the document specifier is either the document's title or a shorter nickname and the element specifier is a string such as `chapter 1`, `section Vehicle Repairs` or simply `Introduction`. In this way a LaTeX document can include the command,

```
\link{See the User's Guide}{guide:section
        Introduction}
```

to reference the introduction from a section of a different document or

```
\link{See Table 2}{me:table 2}
```

to cross-reference a table in the current document.

The `hyperdvi` process is responsible for resolving references to structures within a document and displaying text from the dvi file, while the LACE daemon is responsible for resolving references to documents (potentially across local- and wide-area networks). The LACE daemon also integrates access to other kinds of textual, graphic and video material which may be appropriate to the document being perused.

Figure 7 shows a LACE session. The main window (to the left) was displayed in response to a command-line invocation of `lace Humanities Computing`. The menu (appearing in response to the user's mouse-click) has the usual previewing options of page selection and the common hypertext facility of stepping backwards through the list of elements that have been browsed. The *Contents* and *Tables* submenus are derived from the document's Table of Contents and List of Tables, also seen in this window. The *Add to Trail* item allows the reader to add the current element to a list of "interesting" elements for later browsing. (The trail is a plain text document containing references to each element which can be browsed by the LACE daemon or edited with a text editor. An *involuntary trail* document is also maintained which holds a list of every element that the users ever browses.)

Buttons are transparent patches which cause the cursor to change shape as it is moved over them. Buttons have *actions* associated with them which are invoked when a mouse-press occurs while the cursor is inside them. The actions may be an arbitrary computation (expressed in PostScript), but are usually requests to the LACE daemon to resolve a `document:element` pair.

Since the buttons are transparent it is the document style's responsibility to provide some visual cueing for the reader, usually by putting the button's text in a distinctive font. Some buttons may also be deduced by semantic context, for example, "See table 6 for further details".

The remaining windows in Figure 7 have been brought up from buttons in the main window.

Conclusion

One of the problems of producing a hypertext from pre-existing documentation is partitioning the information between separate nodes and finding the links between them. LaTeX allows authors to express the logical structure of their documents and the relationship between the different elements of that structure: this provides a useful basis for constructing a hypertext network. It is also has the crucial advantage that we know we can generate very high quality printed versions of what we write; until screen-reading comes of age, a great many people will much prefer to do sustained reading from a printed page.

Any scheme of logical markup would be suited to the job: SGML is in some ways a more obvious choice (Niblett and vanHoff [1989] deal with work using SGML as a the source language for hyperdocuments; the Perseus Project at Brown University, and the Oxford English Dictionary at Waterloo are well-known examples of large-scale projects based on SGML; it is also relevant to note that the international *Text Encoding Initiative* is proposing an SGML tagset, and if this is adopted it will mean an ongoing supply of SGML-compatible documents). However, LaTeX has the advantage of producing a high quality physical representation (on paper or screen) especially for technical and scientific docu-