TUGboat

Volume 19, Number 2 / June 1998

91 Addresses

General Delivery
93 From the President / *Mimi Jett*
94 Editorial comments / *Barbara Beeton*
Copyright protection for typefaces; More on PS fonts; CyrTUG membership now free of charge; IBM's techexplorer; New Omega for Mac; EuroTEX 98 — The Tenth European TEX Conference
95 April Fool's Hoax
97 CTAN CDROM series, compliments of DANTE

Typography
98 Typographers' inn / *Peter Flynn*

Graphics Applications
101 pst-fill — a PSTricks package for filling and tiling areas / *Denis Girou*

Book Review
113 *TEX Unbound*, by Alan Hoenig / *Michael Doob*

Fonts
115 An overview of Indic fonts for TEX / *Anshuman Pandey*
121 Diversity in math fonts / *Thierry Bouche*

Hints & Tricks
135 'Hey — it works!' / *Jeremy Gibbons*
Smart spaced macros everywhere (Robert Tolksdorf); Dashed lines (Pedro J. Aphalo); Double-headed arrows (Jeremy Gibbons)

LATEX
137 Default docstrip headers / *LATEX project team*
139 LATEX News, Issue 9, June 1998 / *LATEX project team*

TEXNortheast
140 Conference Program
142 Final report: TEXNortheast / *Stephanie Hogue*
144 TEXNortheast: Workshops and additional papers
147 mathscape — Combining Mathematica and TEX / *Michael P. Barnett*
157 TEX and LATEX on the Web via IBM techexplorer / *Robert S. Sutor and Samuel S. Dooley*
162 Real Life LATEX: Adventures of a TEX consultant / *Amy Hendrickson*
168 Typesetting with TEX and LATEX / *Alan J. Hoenig*
176 Alternatives to Computer Modern Mathematics / *Alan J. Hoenig*
188 Developing database publishing systems using TEX / *Jeffrey McArthur*
195 Presenting mathematics and languages in Web-pages, using LATEX2HTML / *Ross Moore*
204 BIBTEX 101 / *Oren Patashnik*
208 One-document scientific publishing for print and Web/CD / *Peter Signell*
214 TEX to HTML translation via tagged DVI files / *Michael D. Sofka*

Abstracts
223 Les Cahiers GUTenberg, Contents of issue 28–29

News & Announcements
227 Calendar
92 TUG 98 – Torun, Poland, 17–21 August 1998, Preliminary program
231 Volunteers needed for LaTeX2rtf coordination and development / *Wilfred Hennings*
229 TUG'99 Announcement

Late-Breaking News
228 Production notes / *Mimi Burbank*
228 Future issues

TUG Business
230 Institutional members

Advertisements
231 TEX consulting and production services
232 Y&Y Inc.
c 3 Blue Sky Research

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the TEX Users Group, 1466 NW Front Avenue, Suite 3141, Portland, OR 97209-2820, U.S.A.

1998 dues for individual members are as follows:

- Ordinary members: $60.
- Students: $40.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $70 a year, including air mail delivery.

Periodical-class postage paid at Portland, OR, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TEX Users Group, 1466 NW Front Avenue, Suite 3141, Portland, OR 97209-2820, U.S.A.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group. For further information, contact the TUG office.

Printed in U.S.A.

Addresses

All correspondence, payments, parcels, etc.
TEX Users Group
1466 NW Front Avenue
Suite 3141
Portland, OR 97209-2820
USA

Telephone

+1 503 223-9994

Fax

+1 503 223-3960

Electronic Mail (Internet)

General correspondence:
`TUG@tug.org`

Submissions to *TUGboat*:
`TUGboat@tug.org`

World Wide Web

`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

No matter how many palettes of buttons and how many menu options are offered, users of a program will always want to do something the author has not foreseen. Adding still more buttons and menus is not the answer.

B. Hayes
"Pleasures of Plication",
American Scientist **83**(6)
(November–December 1995)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 19, NUMBER 2 • JUNE 1998
PORTLAND • OREGON • U.S.A.

TUGboat

During 1998, the communications of the TEX Users Group will be published in four issues. The September issue (Vol. 19, No. 3) will contain the Proceedings of the 1998 TUG Annual Meeting.

TUGboat is distributed as a benefit of membership to all members.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

Submitting Items for Publication

The next regular issue will be Vol. 19, No. 4. The deadline for technical items will be November 1; reports and similar items are due by November 15. Mailing is scheduled for early December. Deadlines for other future issues are listed in the Calendar, page 227.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, Barbara Beeton, or to the Production Manager, Mimi Burbank (see addresses on p. 91).

Contributions in electronic form are encouraged, via electronic mail, on diskette, or made available for the Editor to retrieve by anonymous FTP; contributions in the form of camera copy are also accepted. The *TUGboat* "style files", for use with either `plain` TEX or LATEX, are available "on all good archives". For authors who have no network FTP access, they will be sent on request; please specify which is preferred. Send e-mail to `TUGboat@tug.org`, or write or call the TUG office.

This is also the preferred address for submitting contributions via electronic mail.

Reviewers

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to `TUGboat@tug.org` or to the Editor, Barbara Beeton (see address on p. 91).

TUGboat Advertising and Mailing Lists

For information about advertising rates, publication schedules or the purchase of TUG mailing lists, write or call the TUG office.

See page 91 for addresses.

Other TUG Publications

TUG publishes the series *TEXniques*, in which have appeared reference materials and user manuals for macro packages and TEX-related software, as well as the Proceedings of the 1987 and 1988 Annual Meetings. Other publications on TEXnical subjects also appear from time to time.

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee in care of the TUG office.

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which appear in this issue may not be complete.

MS/DOS is a trademark of MicroSoft Corporation
METAFONT is a trademark of Addison-Wesley Inc.
PC TEX is a registered trademark of Personal TEX, Inc.
PostScript is a trademark of Adobe Systems, Inc.
techexplorer is a trademark of IBM Research.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.
Textures is a trademark of Blue Sky Research.
UNIX is a registered trademark of X/Open Co. Ltd.

TUG 98 – Toruń, Poland, 17–20 August 1998 "Integrating TEX with the Surrounding World" Preliminary Programme

The Programme Committee has provided the following preliminary information concerning the 19th annual meeting of the TEX Users Group. Additional information will be posted to

`http://www.tug.org/tug-98/`

as it becomes available.

Friday 14th August 1998:
- ■ Registration for tutorial attendees

Saturday 15th:
- ■ Registration
- ■ Tutorial: Hans Hagen, "Actually making an electronic document", Part I

Sunday 16th:
- ■ Registration
- ■ Tutorial: Philip Taylor and Jiří Zlatuška, "Document design, document markup and the converging worlds of computer typesetting and electronic publishing"
- ⋆ Welcome reception and buffet

Monday 17th:
- ■ Formal opening and grand welcome
- ■ Lectures (see attached list for likely speakers)
- ⋆ Free evening, suggest walk around old Toruń

Tuesday 18th:
- ■ TUG Annual General Meeting
- ■ Lectures (see attached list for likely speakers)
- ⋆ A bonfire at the Teutonic Knights' Castle of Golub-Dobrzyń

Wednesday 19th:
- ■ Lectures (see attached list for likely speakers)
- ■ Free afternoon, except for TUG Board and LUG reps
- ⋆ Gala dinner, preceded by a string quartet concert at the beautifully renovated Artus' Hall

Thursday 20th:
- ■ Lectures (see attached list for likely speakers)
- ■ Formal closing and farewell
- ⋆ TUG Board meeting

Friday 21st:
- ■ Tutorial: Bogusław Jackowski, "TEX & PostScript integration"

Saturday 22nd:
- ■ Tutorial: Hans Hagen, "Actually making an electronic document", Part II

■ – day; ⋆ – evening

Tentative list of talks:

Sasha Berdnikov et al.: The paradigma of encodings in LATEX and the Cyrillic encodings X2/T2

Piotr Bolek: MetaPost and patterns

Włodek Bzyl: NLS/WEB/GNU/CWEB

Laurence Finston: Generating a concordance from TEX input files

Hans Hagen, Erik Frambach, Gilbert van den Dobbelsteen and Taco Hoekwater: TEX in the next millenium

Hans Hagen: Visual debugging in TEX

Hans Hagen: MetaTEX: How TEX does graphics

Hans Hagen: TEX, PDF, forms and JavaScript

Hàn Thế Thành: Improving TEX's typeset layout

Taco Hoekwater: MetaFog conversion of MetaFont fonts to Adobe Type-1

Bogusław Jackowski et al.: More TEX-PostScript links

Richard Kinch: Developing new TEX math fonts for the public domain

Miroslava Misáková: Typesetting with varying letter widths

Timothy Murphy: TEX, Java and the World Wide Web

Dick Nickalls: TEX in the operating theatre!

Janusz M. Nowacki et al.: Antykwa Toruńska – An electronic replica of a Polish traditional type

Karel Píška: Georgian scripts

John Plaice and Yannis Haralambous: French guillemets in Omega

John Plaice and Yannis Haralambous: Preparing for 16-bit math fonts with Omega

John Plaice: A TEXtoMathML translator

John Plaice: Fonts for Omega

Chris Rowley (Keynote speaker): LATEX and the future of TEX extensions

Karel Skoupý: NTS – A new typesetting system

Petr Sojka: Adapting TEX's hyphenation to the needs of the surrounding world

Piotr Strzelczyk & Bogusław Jackowski: CEP – A Ghostscript-based utility for compressing PostScript files

Daniel Taupin: ltx2rtf: Export LATEX documents to Word addicts

Marcin Woliński: PretPrin: A LATEX 2_ε package for prettyprinting texts in formal languages

As of 30 June 1998

General Delivery

From the President

Mimi Jett

Greetings TUG Members!

The growth in our membership between 1997 and 1998 confirms a positive trend for TUG. After several years of decline, our numbers have been increasing steadily for two or three years now. There are many contributing factors — an energetic, proactive board; outstanding volunteerism in the membership; ongoing R&D; and of course, the strong TEX community. At this time, we count 1856 members representing 48 countries. Such strong membership has allowed us to return the support to our members in many ways. For example, we publish *TUGboat* quarterly; distributed TEX Live 2 in 1997 and TEX Live 3 in 1998; and 4allTEX and the CTAN CDs in 1998; hold outstanding meetings such as TUG'97 in San Francisco, TEX/NE in New York and TUG'98 in Torun (Poland); provide the Technical Council; bursary funds; and training/workshops. Benefits of membership are numerous, but the biggest benefit of all is the community. The ability to share, learn, and teach each other outweighs all the software distribution we might consider.

Growth has not been without trouble. Issues of office staff and organization have hampered our ability to provide great service to our members. Just recently, we have hired an outstanding office manager who is already making a difference. Dick Detwiler has a strong background in not-for-profit organizations, publishing, and fundraising. Dick understands the importance of responsiveness from the office, and has received raving compliments from people trying to work with us (or join us!) in the past months. One advantage to having Dick on staff is the instant gratification of having a live person answer the telephone. Even if the caller gets our voicemail, Dick is paged with the message and returns calls quickly. The biggest mess we have had is the Institutional Memberships renewals. I apologize to all of the fine organizations that had trouble with their TUG renewals, and hope that we have finally straightened out all of the kinks.

Our annual conference and meeting is coming up this Summer in Poland, hosted by our good friends at GUST. TUG'98 will be held in Torun, at the Nicholas Copernicus University. How fitting that we meet at the birthplace of the man who first realized that the earth was not the center of the universe, just as it is becoming obvious that, in the universe of mathematics, TEX is the center! OK, maybe I am a little zealous, maybe TEX is not the core of science, but it certainly gives us a language in which to communicate. The conference will include many important presentations and provide hands-on workshops in the week surrounding the meeting. There are exciting events planned, both cultural and recreational. Please check the Web site, `http://www.gust.org.pl/TUG98/`, for more details. We look forward to seeing you in Poland!

In the damage repair department, I would like to stress that TEX remains freely available, as it was intended by Professor Knuth when he created it many years ago. One of our long-time members, and resident jokesters, posted an April Fools' Day notice about Knuth selling out to Microsoft. Please understand that this is only a joke, and although it was published in MAPS (by NTG), with color photos of Don and Bill, it remains a joke. What would Microsoft want with such a powerful piece of software anyway? For more about this, please see the article on page 95.

And finally, we would once again like to let you know that the CTAN CDs in this issue are a gift from DANTE e.V. and a valuable repository of the CTAN archives. Our thanks to DANTE!

◇ Mimi Jett

`mimi@iccorp.com`

Editorial Comments

Barbara Beeton

Copyright protection for typefaces

In the last issue this column included an item concerning the decision, finally, to allow some copyright protection in the U.S. to computer programs which define typefaces.

An article containing extensive background on this topic, "Protection for Typeface Designs, A Copyright Proposal", by Terrence J. Carroll, can be found on the Web at `http://www.aimnet.com/~carroll/copyright/typeface.html`. It was originally published in the *Santa Clara Computer and High Technology Law Journal*, Volume 10 (1994), No. 1.

More on PS fonts

A tutorial concerning the use of PostScript fonts with LaTeX 2ε has been created by David Wright and is available at `http://www.phys.washington.edu/~wright/texfonts/`.

The tutorial covers the LaTeX font model, the preparation of `tfm` files for PS fonts, the construction of encodings and font families, and the configuration of dvips to use PS fonts. It's well worth a look.

CyrTUG membership now free of charge

In May, Irina Makhovaia announced that CyrTUG membership is now free of charge. Information can be obtained from the CyrTUG Web page, at `http://www.cemi.rssi.ru/cyrtug/`.

A working group under the sponsorship of CyrTUG has been actively developing a new LaTeX-compatible character layout for Cyrillic fonts; reports on various facets of this work were made at EuroTeX 98. The work is all but done, and the resulting fonts and LaTeX support should be available soon.

IBM's techexplorer

Shortly before this issue went to press, Bob Sutor of IBM's Interactive Scientific Publishing group announced the imminent availability of the new "professional edition" of the techexplorer Hypermedia Browser. Development of this tool began as an experiment to see whether a subset of LaTeX could be extended to support interactive viewing of documents for a computer algebra system.

Experimental versions of techexplorer have been available for about two years, and earlier this year a version of the product was stabilized as the "introductory edition", a no-charge version that is functional in a browser environment, but doesn't permit certain useful options such as printing. These additional features will be available in the "professional edition".

The big surprise is the suggested retail price of the "professional edition" — $29.95. At that price, it shouldn't be a hardship even for students and other potential users with limited means. Congratulations to IBM on their enlightened policy.

Details should be forthcoming at `http://www.software.ibm.com/enetwork/techexplorer/`.

New Omega for Mac

In April, Tom Kiffe announced that Omega 1.5 had finally been ported to the Macintosh, including the entire suite of programs. However:

The programs are for PowerPC only and won't run on older 68k Macs. MPW tool versions will be forthcoming later. To use these programs you will need a complete CMacTeX 3.0 installation.

The programs and installation instructions can be found at `http://www.kiffe.com/cmacomega.html`, and they are also posted on CTAN in `systems/mac/cmactex/cmacomega/`.

EuroTeX '98 — The Tenth European TeX Conference

The 10th European TeX Conference was held in St.-Malo, France, from March 29–April 1, as one facet of the "Second Week on Electronic Publishing and Typography" (WEPT'98). I was privileged to attend, and was delighted to renew many old acquaintances among TeX users and participants in the other conferences as well.

Some of the "hottest" topics were fonts (particularly cyrillic and math, as well as one fascinating study in developing, with METAFONT, fonts particularly suitable for use in telephone directories) and tools for use on the Web and in electronic publishing (pdfTeX, HTML, XML, and techexplorer). (Summaries of the articles from the Proceedings appear in this issue starting on page 222.)

My reason for attending was to carry some news concerning an initiative to obtain Unicode assignments for math symbols that are not presently included in that collection, and are, for this and other reasons, difficult to use in Web-based documents. The Unicode Technical Committee has received our proposal, and a member of the committee with quite a bit of knowledge of technical publishing has been assigned to work with us. I expect to report on the outcome of this project later in the year.

◇ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940 USA
bnb@ams.org

April Fool's Hoax

Webster's 3rd International Dictionary contains several pertinent definitions:

April Fools' Day April 1st, when practical jokes are played on the unwary.

hoax to trick into believing or accepting or doing something : play upon the credulity of [someone] so as to bring about belief in or acceptance of what is actually false and often preposterous.

practical joke a joke whose humor stems from the tricking or abuse of an individual placed somehow at a disadvantage.

Well, the attached "news release" certainly fits this description. For the first few paragraphs, I was in turns curious, shocked, horrified, ..., and laughing out loud. The quote attributed to me a couple of paragraphs from the end was suspicious — I don't think I would have phrased it quite that way. ("We must have been out of our minds!" is perhaps closer to the flavor.) And the date — April 1 — clinched the matter.

You may have seen this document somewhere else, either in print (it appears in the NTG MAPS No. 20, complete with color photos of Don and Bill), or in electrons (`comp.text.tex` is one place where it was circulated). Be reassured that it is truly a *hoax*, a genuine *practical joke*, and indeed *preposterous*.

As Don — Professor Knuth — has stated on a number of occasions, the TEX program is meant to be freely available, but the only person who is allowed to change the source code directly is Donald Knuth. He says on one of his Web pages, relevant to the CM fonts, "...I decided to put these fonts into the public domain rather than to make them proprietary, all I have asked is that nobody change them, *unless the name is changed*, so that every user can obtain equivalent results on all computer systems, now and 50 years from now." The same sentiment applies to TEX itself. And to protect his investment of time and effort, Don assigned the ownership of the TEX logo to the American Mathematical Society.

We don't hold it against the perpetrator of the present hoax — Richard Kinch — that his words have been so successful. But we do all hope that the rest of you, who may see this out of context, are not fooled. Read it and laugh, but *don't believe a word of it!!!*

◇ Barbara Beeton

Microsoft Buys TeX, Plans New Products
Stanford Professor Reaps Windfall

Palo Alto, California, USA (CNEWS/MSNBC) — In a major move into the scientific publishing market, Microsoft Corporation announced today that it has purchased all rights to the computer language and document compiler known as TeX (pronounced, "tech"), and plans a major new product line based on the 20-year-old software.

Stanford Professor Donald Knuth (pronounced, "kah-nooth"), the author of the widely-used TeX software, in a joint press conference at the university campus with Microsoft Chairman Bill Gates, acknowledged that the two had been negotiating for some months. "I felt that two decades of TeX in the public domain was enough. I am reasserting the copyright to my original work in TeX. Microsoft will carry the ball now, and I can get back to my computer science research." Knuth acknowledged he was paid a "seven-figure sum" from Microsoft, which he will use to finance his work on a project he has code-named "Volume 4".

At the press conference, Microsoft chairman Bill Gates said the acquisition was "the kind of cooperation between academia and industry that builds prosperity for both." He added that TeX would "finally give Microsoft a foothold in mathematical desktop publishing" that has eluded the software giant since its founding. Drawing gasps of surprise from the college audience, Gates asserted that "TeX will soon be biggest jewel in the Microsoft crown."

Apparently the jewel metaphor will include a hefty, unavoidable price tag for future TeX users. Gates outlined plans whereby all existing TeX compilers would be phased out, to be replaced by a new Microsoft master implementation written in C++. Beta versions for public testing on Windows 95 and NT platforms are expected in late 1998, issuing from a new 205-programmer project laboratory at Microsoft's Redmond campus. Microsoft TeX for other platforms, such as Unix workstations, will follow at an as-yet unspecified date. According to Gates, "the master TeX from Microsoft will ensure that the incompatibilities across platforms are once and for all eliminated." TeX software is widely used due its portability, although variations among operating systems have been troublesome due to uncoordinated development.

Unlike the technical aspects of the project, Gates explained that pricing for Microsoft TeX has already been firmly set. The single-user retail product is expected to have a street price of about $600 and consist of three CDs. When heckled by an graduate student complaining about a high price for a formerly free product, Gates seemed startled, explaining that a "student edition at $299 is likely" and that "Microsoft will use the revenue to make TeX better."

Most current users of TeX have paid nothing for their implementations, derived from Professor Knuth's formerly-free work. Before leaving the podium, Gates made a final comment that "TeX hasn't changed in years. What kind of a product can that be?", and then handed the microphone to an assistant, introduced only as the project leader for Microsoft TeX.

The assistant displayed an overhead presentation using the current test version of Microsoft TeX. Equations and tables could be seen dissolving into each other in a morphing action between frames. "No one has ever done that with TeX," Gates announced from an audience seat at one point. "It's the kind of sizzle that can really enliven a dull paper at an academic conference." Some onlookers were not convinced, especially when the program crashed midway through the demonstration, resulting in a five-minute delay while Windows 95 was restarted. Microsoft technicians later blamed a third-party display driver.

The impact on the large base of existing TeX users was unclear. During a question-and-answer period, Gates said that the "TeX" trademark would be registered as the exclusive property of Microsoft, and could not appear in any competitive or free software. "We are granting of our own good will until the 3rd quarter of 1998, free use to any existing TeX vendors or public-domain authors. That's plenty of time for an orderly phase-out and change-over to Microsoft TeX, or no TeX at all. After that, our legal department will be contacting them."

A Microsoft attorney added that some of the project personnel would be dedicated to searching the Internet to find non-Microsoft TeX software. "Archives and collections of TeX-related programs will not be permitted. The standards must be enforced, or they become meaningless. We are rescuing a fine piece of work from being diluted into worthlessness. You would not believe the number of programs that have been based on TeX without any central, controlling authority. We will stop this infringement."

Some large organizations dependent on TeX were stunned by the announcment and had not yet formed plans for dealing with the change. At the American Mathematical Society, whose publications largely depend on TeX for typesetting, editor Barbara Beeton was incensed. "I can't believe Don [Professor Donald Knuth] sold us out like this. We should have never based a publishing enterprise of this scope on so-called public-domain software. What were we thinking?" Publication schedules for the rest of 1998 were on hold, and journal editors scrambled to reassure their authors that deadlines would not slip more than a few months.

Certain small businesses are also expected to feel the impact of the Microsoft ownership of TeX. Palo Alto restaurant owner Wu Chen appeared unhappy at the news, stating that "for ten year I print new menu every day with TeX, now I will pay big time." He displayed a crumpled, grease-spotted take-out flyer, and with tears in his eyes explained how multiple columns, exotic typefaces, and daily price changes could all be printed by TeX in a multilingual format. "In Wordperfect this would be a long journey."

Commercial vendors of TeX software stand to lose everything in the face of the new Microsoft monopoly. While most derivatives of TeX were freely published, several companies had made a business of publishing proprietary versions. One anonymous source from a leading TeX firm said that "publishing TeX was a gold mine while it lasted, and the Internet let us mine it deeper and deeper. Now this is a cave-in right on our heads. TeX was a monumental work of beauty and utility, freely given to the world by one of the finest and most generous minds of the 20th century. Now it belongs to a lucky dropout. We're finished."

Date of Publication 04/01/98
For further information see
`http://idt.net/~truetex/30/`

Newsgroups: `comp.text.tex`
Date: 01 Apr 1998 09:45:12 -0800
From: Matt Austern `<austern@sgi.com>`
Subject: Re: Microsoft buys TeX! Knuth sells out!

Richard Kinch `<truetex@IDT.NET>` writes:

> Did anybody else see this news item today?

I saw that, but the story is wrong — in fact, it's quite backward. The actual situation is that Donald Knuth bought Microsoft. He is currently working on rewriting Microsoft Word in Pascal, in accordance with the principles of literate programming. The new release will be closely integrated with Metafont.

You can expect to see *The Wordbook* and *Word: The Program* in late 1998, shortly after the publication of volume 5 of *The Art of Computer Programming*.

CTAN CD-ROM series, compliments of DANTE

Editor's note: With this issue of *TUGboat* is included a 3 CD-ROM collection compiled and contributed by DANTE e.V., the German TEX users group.

The Editor and production team and the TUG Board are deeply grateful to the board and members of DANTE for their generosity, in particular to Marion Neubauer, the current President of DANTE, and to Joachim Lammarsch, the past President.

The following notices, included on the CDs, are repeated here for information.

COPYRIGHT

All copyright regulations were respected during the creation of this CD-ROM. If there are faults concerning copyright, please report them to us.

In each case, the licensing restrictions of individual software products shall apply.

The main parts of the software on this CD-ROM are copyrighted under the GNU Public Licence (copyleft). The text of this licence is contained on this CD-ROM.

Neither DANTE e.V. nor the author(s) of the software packages will be in any way liable for maintenance or support of the software on this CD-ROM, nor will they be in any way liable for any damage or loss resulting from the use of this software, no matter how caused. No guarantee or warranty, express or implied, is offered in respect of this CD-ROM or the software thereon.

Reproduction and/or redistribution of this CD-ROM is prohibited as well as its use in software servers or mailboxes without the written permission of DANTE e.V.

Heidelberg, Germany, 25th of January 1998

DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 101840
D-69008 Heidelberg
Germany
Phone: +49/6221/29766
Fax: +49/6221/167906
email: dante@dante.de
www: http://www.dante.de

Important information concerning the CTAN CD-ROM series

The CTAN CD-ROM series, consisting of 3 CD-ROMs, is a "nearly complete" copy of the CTAN (Comprehensive TEX Archive Network) server of DANTE e.V. The copy was made on January 25th 1998. "Nearly" means that a few parts of the archive (containing nearly 3 GB of software) have not been copied because of lack of space.

On each CD-ROM of this series one can find a file named `FILES.cd` in the root directory. It contains the table of contents of all three CD-ROMs. This `README` and the `COPYRIGHT` are located there too. On the first CD-ROM [CD-1] additionally the file `CTAN.ori` can be found, which contains the complete list of all files on the CTAN server (the result of the command `ls -r`).

Almost all software can be installed directly as described in the various packages. Only a few subdirectories have been compressed by the program infozip, which is compatible to the well-known pkzip program. The infozip program can be found in the subdirectory `/tools/zip/info-zip` on the second CD-ROM [CD-2]. The source code as well as binaries for several operating systems are stored there.

The three CD-ROMs do not contain the whole CTAN. Some software packages which have no relation to TEX, which are old versions or where copyright prohibits a distribution via CD-ROM, have not been copied.

The packages are distributed on the CD-ROMs as follows:

CD-1
`/biblio`
`/digests`
`/dviware`
`/fonts` [zipped]
`/graphics`
`/indexing`
`/info`
`/language`
`/macros`

CD-2
`/help`
`/support`
`/systems` [only acorn, e-tex and unix subdirectories]
`/tds`
`/tools`
`/usergrps`
`/web`

CD-3
`/systems` [without acorn, e-tex and unix subdirectories]

All directories which have not been included will be listed at the end of the [copyright] file.

The CD-ROMs have been mastered under Linux (thanks to Linus Torvald!) to the ISO9660 standard with Rock Ridge extensions. They should therefore be readable under all systems which supports ISO9660. The (sometimes very long) directory or file names have been automatically converted into

the ISO9660 format. For reference to the original names a file named `TRANS.TBL`" is stored in every directory. The file contains the original names and the name in ISO format. Operating systems with Rock Ridge support can still see and use the original names.

There exist symbolic links in the CTAN directory structure. Such symbolic links are supported only from relatively few operating systems and with the distribution on three CD-ROMs many of the links would be out of order. For that reason *all* symbolic links has been deleted. In order to help searching for distinct files, all symbolic links have been stored in the file `SYMLINKS.ori` in the root directory of the first CD-ROM [CD-1].

Have fun and success while using TEX and Co.

◇ DANTE e.V.
Postfach 101840
D-69008 Heidelberg, Germany
`http://www.dante.de`

Typography

Typographers' Inn

Peter Flynn

'C' stands for Euro

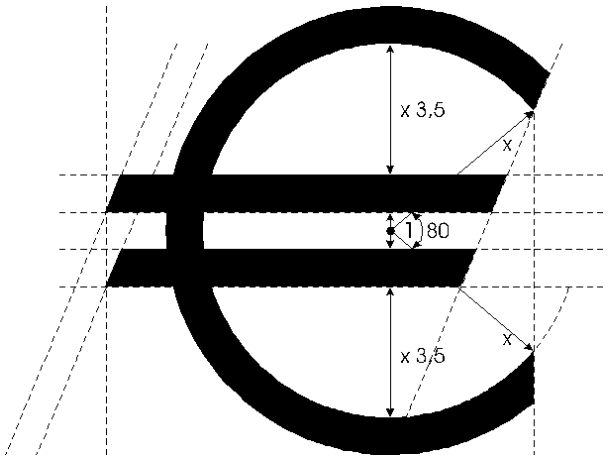
Just to take our minds off the Year 2000 problems, here in Europe we have a new currency on the horizon. With effect from the beginning of 1999, banking and commerce can be conducted in a single currency valid throughout the European Union. The old national currencies will continue in use until 2002, when a uniform coinage and set of notes will replace them in most states (a few have opted out for the moment). The whole business will entail lots of dual- or multi-currency computing for the transitional years, and doubtless manufacturers of POS equipment will have a field day, but in the long run it can only benefit the moves towards further integration. It is, after all, only just over 200 years since the United States of another continent replaced the pounds, doubloons, reales, and moidores of their mixed English, Spanish, and French heritage with the pieces-of-eight of Seville and Mexico, known from their resemblance to an older German coinage as 'thalers' or 'dollars'. However, the task of creating a name for the new European currency was not the only problem: a typographic one has arisen also.

Our unelected lords and masters in the European Commission, ineptly supported by our elected public representatives, demonstrated their feeble grasp on reality by making what is perhaps the most crass naming mistake this millenium: they decided to call the new currency the 'Euro' instead of using the well-established and perfectly adequate ECU. The PR suits claim this was to avoid offending the Germans, who would have been upset at the use of an ostensibly French name (the Ecu was an old French coin, although the modern ECU actually stands for European Currency Unit). I am perfectly sure the modern Germans are far too sensible to be offended by so trivial an excuse, and I'm equally sure many millions of us would have been very pleased to see an historically important name revived. But it was not to be, and we're now lumbered with one of the silliest and most inelegant names ever devised for a monetary unit. End of rant.

However, the Commission have redeemed themselves to a small extent at least by producing an inoffensive design for the Euro which represents a rounded 'E' with a double bar through it, taking the symbology from the double bar through the traditional versions of the £ and the $ (see Figure 1: you can read more at `http://europa.eu.int/euro/`).

Microsoft, in a laudable attempt to keep their fonts up to date and usable by Europeans, rather missed the point, and added a symbol to their serif font files based on a capital C with a single serif at

Figure 1: The European Commission's design for the Euro symbol



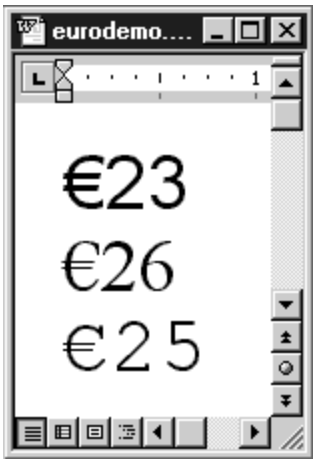
the top (see Figure 2 and `http://www.microsoft.com/typography/faq/faq12.htm`). Monotype were apparently retained by Microsoft to make the designs, which makes it all the more surprising that they seem to have failed to grasp that the Commission's design showed a symbol with the central bars and no serifs, and this seems to have been misinterpreted as being a 'C' rather than an 'E'. In a seriffed font, instead of adding a serif at the bottom to retain the same degree of symmetry, Microsoft left it as a 'C' with a top serif and two lines through it, which unfortunately fails to convey the notion of 'E' — which is (presumably) central to the whole concept.

I griped about this on `TYPO-L` in January, and Simon Daniels from Microsoft kindly brought it to the attention of the people at Monotype responsible for the outlines and hinting. The screen shot in Figure 2 had apparently been on their site for about five months, and no one else had noticed. I haven't seen any designs for Metafont fonts yet: maybe the TEX community can be the first to get it right.

Oops

Christina Thiele and a number of others picked me up on my remark in the *Quote unquote* section of the last *Typographers' Inn* about there being a reverse-quote in the `wsuipa` fonts at `\char'163`. I jumped the gun on that: it's not a reverse-quote, it's there because it's a standard way of representing the Arabic letter 'ain'... so it's got zero to do with quote marks and everything to do with transcription. The IPA usage is that it is recommended for 'weak aspiration after voiceless stops'[1]. Sorry about that — but I'm still no closer to finding out where this →'quote (so-called) comes from.

Figure 2: Microsoft's designs for the Euro symbol



TEX and TEXability

I said I was going to use LATEX 2ε for my forthcoming book on SGML[2] to see how it coped. The answer was: pretty well, far better than I had expected. My big concern, coming from nearly two decades of using plain TEX, was that I would find myself being almost forced to use predetermined styles because of the notorious difficulty of making even small changes to the LATEX defaults (if any skeptics disbelieve that, they have only to read `comp.text.tex` for a few hours and count the FAQs about how to make modifications).

As I explained last time, there are still some rough edges to LATEX 2ε, but I didn't hit any major snags. My publisher provided a class file, which was still under development at the time, so I had to make a few changes to it. But I needed 13 packages to enable the things I needed to do, which nicely illustrates what Paul Anagnostopoulos pointed out to me after my last gripe, that 'the reason that there is a tendency... to concentrate on the "borderline cases and special parameters" rather than the daily necessities is because most of the people working on LATEX don't know much about books. This is no better witnessed than by the fact that, after 10 years of LATEX development, blank pages still have running heads!' While I would dispute the 'most' — there are several people working on LATEX 2ε who know lots about books — it is still true that book production in LATEX needs better parameterization. There are several style files already in existence to do some of this, but once the current backlog is out of the way it's a project I'd like to look at more closely.

While I'm riding this hobby, is there no way we can get rid of the weird concept that reports have chapters? Very few of them that I have ever seen in business or research have chapters: only a small number of very large ones do; the rest have sections as their major division. It's one of those embarrassing 'features' that lead new users, especially business users, to look at LATEX numbering their first section as 0.1, roll their eyes to heaven, and mutter 'academics!' — a gross slur, but understandable in the circumstances. By all means make it an option, but not the default.

Usage and abusage

The result of my forays into LATEX has been that I've started using it for many more tasks for which I would have used plain TEX before, and I've even

started writing a class file for my in-house memo document type as a way of getting into it. The regularity and consistency of macro-driven typesetting makes LaTeX's use of environments an especially attractive proposition if you deal with SGML because of the availability of public-domain packages like jade and commercial programs like Omnimark (which also has a free version), as these make conversion from SGML to TeX (amongst other formats) relatively straightforward. It's clear that for future development we need many more document classes than articles, books, letters, and reports, and I'm getting tired of seeing people doing what I did today, writing an advertising leaflet using the article class.

This is known in markup circles as 'tag abuse', and it's surprisingly prevalent. I'm as guilty as anyone, and I probably rant about it just as much: it *is* frustrating when you want to signal something you consider vital in a document only to find no-one else has apparently ever considered it important before, and has provided no control sequence to do it. This is especially true if the something doesn't actually have a typographic instantiation, such as a personal name. In the days when I wrote directly in TeX, I often used a dummy control sequence such as `\person{...}` because I use what I write as a database, and it can be very convenient to be able to dig back through files with a tool like *grep* or *Perl* and use the existence or proximity of names to help find what I'm looking for.

I'm happy to make two announcements, therefore: one is for a new (well, 1-year-old) organization, SDATA, the Society for the Definitive Abolition of Tag Abuse. There is a Web site at `http://www.ucc.ie/sdata` and members can contribute lore, suggestions, anecdotes, code, patches, and advice on how to avoid or cope with it. I don't know if it will achieve any major change in the hearts of document type designers, but it may help relieve the annoyance of having to abuse an otherwise inoffensive control sequence — like all those who sedulously use `\emph` when they actually want italics, because someone told them it was evil to hard-code appearance when you ought really to be using generic encoding, and emphasis is all you've got apart from `\textit`. In the absence of `\linnaean`, `\product`, `\citetitle`, and `\foreign`, can we blame them?

The other announcement is for a new journal, *Markup Languages: Theory & Practice*, from MIT Press (ISSN: 1099-6621), starting in early 1999. This quarterly, peer-reviewed technical journal will be the first one devoted to research, development, and practical applications of text markup for computer processing, management, manipulation, and display. There is a Call for Papers being circulated in the appropriate places on the network: contact Tommie Usdin (`btusdin@mulberrytech.com`) or Michael Sperberg-McQueen (`tei@uic.edu`) for more details (doubtless there will be a Web site soon), and get your fingers working: I'm on the Editorial Board and I'd like to see TeX and LaTeX users writing submissions.

H&J revisited

Another point Paul A. (see above) made to me was that some publications (*PC Magazine* was one example he gave; but I've seen it in *Byte* and *Dr. Dobb's* also) have a policy that URL punctuation should *not* fall at the end of a line, but at the beginning of the next (I was recommending the opposite). This is apparently because a period at the end of the line looks like it ends the sentence, and thus the URL. It looks ugly, but may serve a real purpose.

Finally, has really no-one else ever hit the snag with `\path` I mentioned in the last issue? It's a great concept (the path package), like an extended `\verb` which lets you define your own set of allowable breakpoints that can break the line without hyphenation. But the list of breakpoint characters is also the list of allowed characters for treating *verbatim*, which means if you want it to handle backslashes as they stand, but not to break a line after one, you're snookered. Suggestions on a Möbius Strip, please: I'm on vacation.

References

[1] Geoffrey K. Pullum and William A. Ladusaw, *Phonetic Symbol Guide*, University of Chicago Press, Chicago, 1986, p. 216.

[2] Peter Flynn, *Understanding SGML and XML Tools*, Kluwer Academic Publishers, Boston, 1998.

◇ Peter Flynn
Computer Centre,
University College
Cork,
Ireland
pflynn@imbolc.ucc.ie
http://imbolc.ucc.ie/~pflynn/

Graphics Applications

pst-fill — a PSTricks package for filling and tiling areas

Denis Girou

Abstract

pst-fill is a PSTricks (van Zandt, 1993), (Girou, 1994), (van Zandt and Girou, 1994), (Hoenig, 1998), (Goossens, Rahtz, and Mittelbach, 1997) package for simple drawing of various kinds of filling and area tiling. It is also a good example of the great power and flexibility of PSTricks, as it is very short (around 200 lines long) but nevertheless extremely powerful.

The package was written in 1994 by Timothy van Zandt but publicly available only in PSTricks 97 and without any documentation. We describe here version *97 patch 2* of December 12, 1997, which is the original one modified by Denis Girou to manage *tilings* in so-called *automatic* mode. This article serves as both reference manual and user's guide.[1]

This package is available on CTAN in the `graphics/pstricks` directory (files `latex/pst-fill.sty` and `generic/pst-fill.tex`).

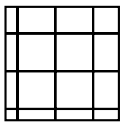
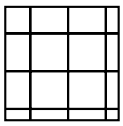
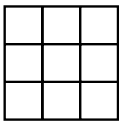
1 Introduction

We use *filling* to describe the operation which consists of filling a defined area by a pattern (or a composition of patterns), and *tiling* as the operation which is like filling, but with control of the starting point (we use the upper left corner), where the pattern is positioned relative to this point. There is an essential difference between the two modes, as without control of the starting point we cannot create the *tilings* (sometimes called *tesselations*) used in many fields of Art and Science[2].

Tilings are a wide and difficult field of mathematics, and this package is limited to simple ones, mainly *monohedral* tilings with one prototile (which can be composite, see section 3.1). With some experience and wiliness we can do more, and easily obtain quite sophisticated results, but obviously hyperbolic tilings like the famous Escher ones or aperiodic tilings like the Penrose ones are not within the capabilities of this package. For more complex needs, we must use low level and more painfull techniques, with the basic `\multido` and `\multirput` macros.

2 History of the package, and its two different modes

This package was written in 1994 by Timothy van Zandt. Two modes are defined, called respectively *manual* and *automatic*. For both, the pattern is generated on contiguous positions in a large area which includes the region to fill, which is later cut to the required dimensions by a clipping mechanism. In the first mode, the pattern is explicitely inserted in the PostScript output file each time. In the second, the result is the same but with a single insertion of the pattern and a repetition done by PostScript. Control over the starting point was lost, so it allowed only *filling* a region and not to *tiling* it.

The difference between the two modes is shown here; *filling*:  or also  where, as you can see, the initial position is arbitrary and depends on the current point, and *tiling*: 

It is clear that filling is very restrictive compared to tiling, as the desired effect very often requires the possibility of controlling the starting point. The automatic mode was therefore of limited interest, but unfortunately the *manual* one has the very big disadvantage of requiring very large resources, in disk space and subsequently in printing time. A small tiling can sometimes require several megabytes in *manual* mode! The original package was thus not really usable in practice for tilings.

I modified the code to allow tiling in *automatic* mode, also giving control over the starting point. Most of the time, if some special options are not used, the tiling is done exactly in the region described, which make it faster. There is little reason to use the *manual* mode, apart very special cases where the *automatic* one cannot work, as explained later – currently, we know of only one case.

To load this modified *automatic* mode, with LaTeX use simply:

---

[1] Great thanks are due to Sebastian Rahtz for his help in correcting my English and of course to Timothy van Zandt for his impressive development of the PSTricks package.

[2] For an extensive description of tilings, and their history and usage in many fields, see the reference book (Grünbaum and Shephard, 1987). French readers can also find much explanation and reference material in (André and Girou, To appear), and especially in (Girou, To appear).

In the TeX world, very little work has been done on tilings. There is mainly the *tile* extension of the XY-pic package (Rose and Moore, 1991-1998), the article of Kees van der Laan (van der Laan, 1996, paragraph 7) (the tiling was in fact done directly in PostScript) and the MetaPost program (available in `graphics/metapost/contrib/macros/truchet`) by Denis Roegel for the Truchet contest in 1995 (Esperet and Girou, To appear).

`\usepackage[tiling]{pst-fill}`
and in plain TEX after:
`\input{pst-fill}`
add the following definition:
`\def\PstTiling{true}`

To obtain the original behaviour, simply do not use the *tiling* optional.

Users should be aware that in *tiling* mode, some other changes were introduced. Aliases for some parameter names were defined for consistancy (all parameters begin with the `fill` prefix) and some default values which were not well adapted for tilings were changed (`fillsep` is set to 0 and `fillsize` set to `auto`). `fillcycle` was renamed to `fillcyclex`, and the normal behaviour was restored whereby the frame of the area is drawn and all line (`linestyle`, `linecolor`, `doubleline`, etc.) parameters are now active (but not in non *tiling* mode). Some new parameters were introduced to control tiling, described below.

In all the following examples, we always use *tiling* mode.

To do a tiling, we just have to define the pattern with the `\psboxfill` macro and to use the new `fillstyle` `boxfill`. Note that tilings are drawn from left to right and top to bottom, which can can be important in some circumstances.

PostScript programmers may be interested to know that, even in *automatic* mode, the iterations of the pattern are managed directly by the PostScript code of the package, which uses only PostScript Level 1 operators. The special ones introduced in Level 2 for drawing patterns (Adobe, 1995, section 4.9) are not used.

First, for convenience, we define a simple `\Tiling` macro, which will simplify our examples:

```
\newcommand{\Tiling}[2][]{%
  \edef\Temp{#1}%
  \begin{pspicture}#2
    \ifx\Temp\empty
      \psframe[fillstyle=boxfill]#2
    \else
      \psframe[fillstyle=boxfill,#1]#2
    \fi
  \end{pspicture}}
```

2.1 Parameters

There are **14** parameters available to change the way the filling/tiling is defined, and one debugging option.

`fillangle` (real): the value of the rotation applied to the patterns (*Default: 0*).

In this case, we must force the tiling area to be noticeably larger than the area to cover, to be sure that the defined area will be covered after rotation.

```
\newcommand{\Square}{%
  \begin{pspicture}(1,1)
    \psframe[dimen=middle](1,1)
  \end{pspicture}}

\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillangle=45]{(3,3)}\hspace{3cm}
\Tiling[fillangle=-60]{(3,3)}
```

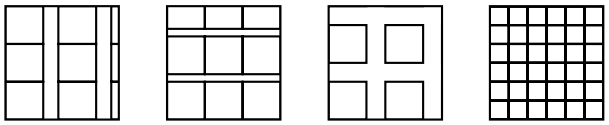
`fillsepx` (real|dim): value of the horizontal separation between consecutive patterns (*Default: 0 for tilings*[3]*, 2pt otherwise*).

`fillsepy` (real|dim): value of the vertical separation between consecutive patterns (*Default: 0 for tilings*[3]*, 2pt otherwise*).

`fillsep` (real|dim): value of horizontal and vertical separations between consecutive patterns (*Default: 0 for tilings*[3]*, 2pt otherwise*).

These values can be negative, which allow the tiles to overlap.

```
\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillsepx=2mm]{(3,3)}\hfill
\Tiling[fillsepy=1mm]{(3,3)}\hfill
\Tiling[fillsep=0.5]{(3,3)}\hfill
\Tiling[fillsep=-0.5]{(3,3)}
```



`fillcyclex`[4] (integer): Shift coefficient applied to each row (*Default: 0*).

`fillcycley`[3] (integer): Same thing for columns (*Default: 0*).

`fillcycle`[3] (integer): Allow for setting both `fillcyclex` and `fillcycley` to the same value (*Default: 0*).

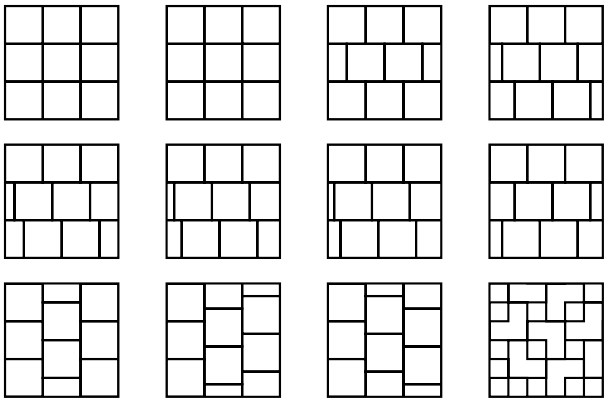
For instance, if `fillcyclex` is 2, the second row of patterns will be horizontally shifted by a factor

[3] This option was added by me. It is not part of the original package and is available only if the `tiling` keyword is used when loading the package.

[4] It was `fillcycle` in the original version.

of $\frac{1}{2} = 0.5$, and by a factor of 0.333 if `fillcyclex` is 3, etc. These values can be negative.

```
\psset{unit=0.5}
\psboxfill{\Square}
\newcommand{\TilingA}[1]
  {\Tiling[fillcyclex=#1]{(3,3)}}

\TilingA{0}\hfill
\TilingA{1}\hfill
\TilingA{2}\hfill
\TilingA{3}

\vspace{3mm}
\TilingA{4}\hfill
\TilingA{5}\hfill
\TilingA{6}\hfill
\TilingA{-3}

\vspace{3mm}
\Tiling[fillcycley=2]{(3,3)}\hfill
\Tiling[fillcycley=3]{(3,3)}\hfill
\Tiling[fillcycley=-3]{(3,3)}\hfill
\Tiling[fillcycle=2]{(3,3)}\hfill
```



`fillmovex`[3] (real|dim): value of the horizontal move between consecutive patterns (*Default: 0*).

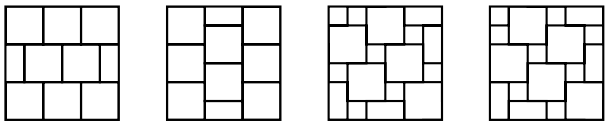
`fillmovey`[3] (real|dim): value of the vertical move between consecutive patterns (*Default: 0*).

`fillmove`[3] (real|dim): value of horizontal and vertical move between consecutive patterns (*Default: 0*).

These parameters allow the patterns to overlap and to draw some special kinds of tilings. They are implemented only for the *automatic* and *tiling* modes and their values can be negative.

In some cases, the effect of these parameters will be the same as that with the `fillcycle?` ones, but this is not true for all values.

```
\psset{unit=0.5}
\psboxfill{\Square}
\Tiling[fillmovex=0.5]{(3,3)}\hfill
\Tiling[fillmovey=0.5]{(3,3)}\hfill
\Tiling[fillmove=0.5]{(3,3)}\hfill
\Tiling[fillmove=-0.5]{(3,3)}
```



`fillsize`
(auto|{(real|dim,real|dim)(real|dim,real|dim)}): The choice of *automatic* mode or the size of the area in *manual* mode. If first pair values are not given, (0,0) is used. (*Default: auto when* tiling *mode is used, (-15cm,-15cm)(15cm,15cm) otherwise*).

As explained in the introduction, the *manual* mode can use up a large amount of computer resources. It's usage is therefore discouraged in favour of *automatic* mode. It only seems useful in special circumstances, when the *automatic* mode fails; only one case is known, when some kinds of EPS files are used, such as the ones produced by partial screen dumps (see 3.2).

`fillloopaddx`[3] (integer): number of times the pattern is added on left and right positions (*Default: 0*).

`fillloopaddy`[3] (integer): number of times the pattern is added on top and bottom positions (*Default: 0*).

`fillloopadd`[3] (integer): number of times the pattern is added on left, right, top and bottom positions (*Default: 0*).

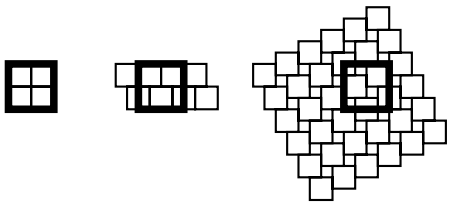
These parameters (exclusively for the *tiling* mode) are only useful in special circumstances, such as in complex patterns when the size of the rectangular box used to tile the area does not correspond to the pattern itself (there is an example in Figure 1) and also sometimes when the size of the pattern is not a divisor of the size of the area to fill and when the number of loop repeats is not properly computed, which can occur.

`PstDebug`[3] (integer, 0 or 1): to see the exact tiling done, without clipping (*Default: 0*).

This is mainly useful for debugging or to understand better how the tilings are done. It is implemented only for the *tiling* mode.

```
\psset{unit=0.3,PstDebug=1}
\psboxfill{\Square}
\psset{linewidth=1mm}
\vspace*{7mm}
\Tiling{(2,2)}\hspace{1cm}
```

```
\Tiling[fillcyclex=2]{(2,2)}\hspace{2cm}
\Tiling[fillmove=0.5]{(2,2)}
```



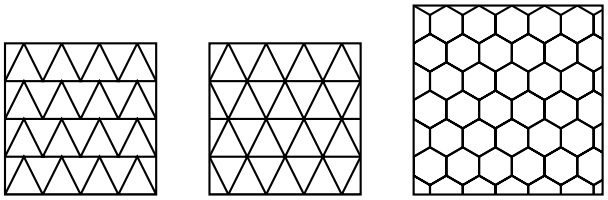
3 Examples

The single **\psboxfill** macro has many variations and different uses. We will try here to demonstrate many of them:

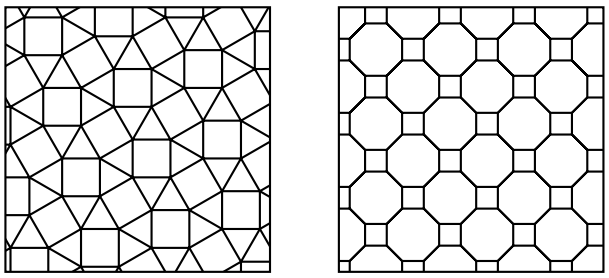
3.1 Kind of tiles

Since we can access all the power of PSTricks macros to define the *tiles* (*patterns*) used, very complicated ones can be created. Here we give four Archimedian tilings (those built with only some regular polygons) from the eleven known, first discovered completely by Johanes Kepler at the beginning of 17th century (Grünbaum and Shephard, 1987), the two *regular* ones with the tiling by squares, formed by a single regular polygon, and two formed by two different regular polygons.

```
\newcommand{\Triangle}{%
  \begin{pspicture}(1,1)
    \pstriangle[dimen=middle](0.5,0)(1,1)
  \end{pspicture}}
\newcommand{\Hexagon}{%
  % sin(60)=0.866
  \begin{pspicture}(0.866,0.75)
    \SpecialCoor
    % Hexagon
    \pspolygon[dimen=middle]
              (0.5;30)(0.5;90)(0.5;150)
              (0.5;210)(0.5;270)(0.5;330)
  \end{pspicture}}

\psset{unit=0.5}
\psboxfill{\Triangle}
\Tiling{(4,4)}\hfill
% The two other regular tilings
\Tiling[fillcyclex=2]{(4,4)}\hfill
\psboxfill{\Hexagon}
\Tiling[fillcyclex=2,fillloopaddy=1]{(5,5)}
```



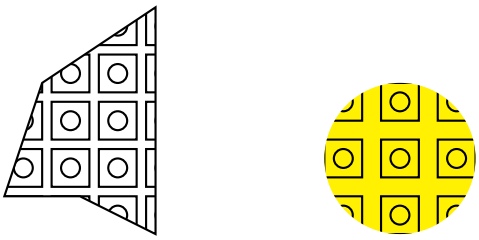
```
\newcommand{\ArchimedianA}{%
  % Archimedian tiling 3.4.6.4
  \psset{dimen=middle}
  % sin(60)=0.866
  \begin{pspicture}(1.866,1.866)
    \psframe(1,1)
    \psline(1,0)(1.866,0.5)(1,1)
           (0.5,1.866)(0,1)(-0.866,0.5)
    \psline(0,0)(0.5,-0.866)
  \end{pspicture}}
\newcommand{\ArchimedianB}{%
  % Archimedian tiling 3.12^2
  \psset{dimen=middle,unit=1.5}
  % cos(22.5) + sin(22.5) = 1.3066
  % cos(22.5) - sin(22.5) = 0.6533
  \begin{pspicture}(1.3066,0.6533)
    \SpecialCoor
    % Octogon
    \pspolygon(0.5;22.5)(0.5;67.5)
      (0.5;112.5)(0.5;157.5)(0.5;202.5)
      (0.5;247.5)(0.5;292.5)(0.5;337.5)
  \end{pspicture}}

\psset{unit=0.5}
\psboxfill{\ArchimedianA}
\Tiling[fillmove=0.5]{(7,7)}\hfill
\psboxfill{\ArchimedianB}
\Tiling[fillcyclex=2,fillloopaddy=1]{(7,7)}
```



We can of course tile an arbitrarily defined area; with the **addfillstyle** parameter[5], we can easily mix the **boxfill** style with another one.

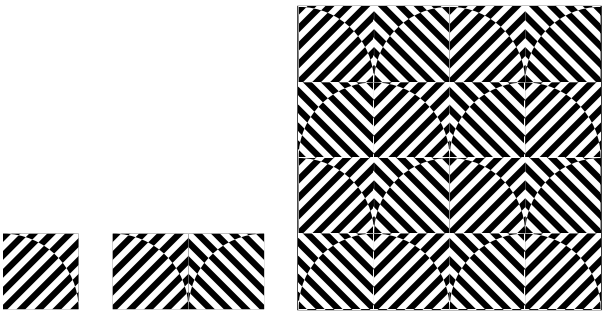
```
\psset{unit=0.5,dimen=middle}
\psboxfill{%
  \begin{pspicture}(1,1)
    \psframe(1,1)
    \pscircle(0.5,0.5){0.25}
  \end{pspicture}}
\begin{pspicture}(4,6)
  \pspolygon[fillstyle=boxfill,
             fillsep=0.25]
            (0,1)(1,4)(4,6)(4,0)(2,1)
\end{pspicture}
\hspace{2cm}
\begin{pspicture}(4,4)
 \pscircle[linestyle=none,fillstyle=solid,
           fillcolor=yellow,fillsep=0.5,
           addfillstyle=boxfill](2,2){2}
\end{pspicture}
```

[5] Introduced in PSTricks 97.



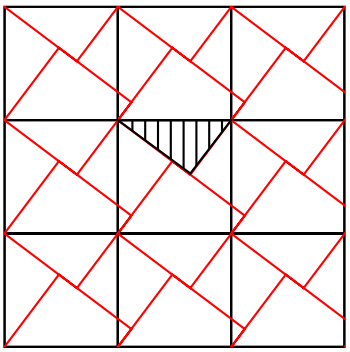
Various effects can be obtained; sometimes complicated ones are surprisingly easy, as in this example reproduced from one by Slavik Jablan in the field of *OpTiles*, inspired by *Op-art*:

```
\newcommand{\ProtoTile}{%
  \begin{pspicture}(1,1)
   % 1/12=0.08333
   \psset{linestyle=none,linewidth=0,
          hatchwidth=0.08333\psunit,
          hatchsep=0.08333\psunit}
   \psframe[fillstyle=solid,fillcolor=black,
            addfillstyle=hlines,
            hatchcolor=white](1,1)
   \pswedge[fillstyle=solid,fillcolor=white,
            addfillstyle=hlines]{1}{0}{90}
  \end{pspicture}}

\newcommand{\BasicTile}{%
  \begin{pspicture}(2,1)
    \rput[lb](0,0){\ProtoTile}
    \rput[lb](1,0){\rotateleft{\ProtoTile}}
  \end{pspicture}}

\ProtoTile\hfill\BasicTile\hfill
\psboxfill{\BasicTile}
\Tiling[fillcyclex=2]{(4,4)}
```



It is also possible to superimpose several different tilings. Here is the splendid visual proof of the Pythagore theorem done by the Arab mathematician Annairizi around the year 900, given by superposition of two tilings by squares of different sizes.

```
\psset{unit=1.5,dimen=middle}
\begin{pspicture*}(3,3)
  \psboxfill{\begin{pspicture}(1,1)
               \psframe(1,1)
             \end{pspicture}}
  \psframe[fillstyle=boxfill](3,3)
  \psboxfill{\begin{pspicture}(1,1)
    \rput{-37}{\psframe[linecolor=red]
                        (0.8,0.8)}
             \end{pspicture}}
  \psframe[fillstyle=boxfill](3,4)
  \pspolygon[fillstyle=hlines,hatchangle=90]
            (1,2)(1.64,1.53)(2,2)
\end{pspicture*}
```



In a same way, it is possible to build tilings based on figurative patterns, in the style of the famous Escher ones. Following an example of André Deledicq (Deledicq, 1997), Figure 1 shows a simple tiling of the *p1* category (according to the international classification of the 17 symmetry groups of the plane first discovered by the Russian crystallographer Jevgraf Fedorov at the end of the 19th century).

Figure 2 shows a tiling of the *pg* category (the code for the kangaroo itself is too long to be shown here, but has no difficulties; the kangaroo is reproduced from an original picture by Raoul Raba and here is a translation into PSTricks from the one drawn by Emmanuel Chailloux and Guy Cousineau for their MLgraph system (Chailloux, Cousineau, and Suárez, 1996)).

And now a Wang tiling (Wang, 1965), (Grünbaum and Shephard, 1987, chapter 11), based on very simple tiles in the form of a square and composed of four colored triangles. Such tilings are simply built with a matching color constraint. Despite its simplicity, it is an important kind of tiling, as Wang and others used them to study the special class of *aperiodic* tilings, and also because it was shown that (surprisingly) this tiling is similar to a Turing machine.

```
\newcommand{\WangTile}[4]{%
\begin{pspicture}(1,1)
\pspolygon*[linecolor=#1](0,0)(0,1)(0.5,0.5)
\pspolygon*[linecolor=#2](0,1)(1,1)(0.5,0.5)
\pspolygon*[linecolor=#3](1,1)(1,0)(0.5,0.5)
\pspolygon*[linecolor=#4](1,0)(0,0)(0.5,0.5)
\end{pspicture}}

```

```
\newcommand{\SheepHead}[1]{%
\begin{pspicture}(3,1.5)
  \pscustom[liftpen=2,fillstyle=solid,fillcolor=#1]{%
    \pscurve(0.5,-0.2)(0.6,0.5)(0.2,1.3)(0,1.5)(0,1.5)(0.4,1.3)(0.8,1.5)
            (2.2,1.9)(3,1.5)(3,1.5)(3.2,1.3)(3.6,0.5)(3.4,-0.3)(3,0)(2.2,0.4)(0.5,-0.2)}
  \pscircle*(2.65,1.25){0.12\psunit}              % Eye
  \psccurve*(3.5,0.3)(3.35,0.45)(3.5,0.6)(3.6,0.4) % Muzzle
  \pscurve(3,0.35)(3.3,0.1)(3.6,0.05)             % Mouth
  \pscurve(2.3,1.3)(2.1,1.5)(2.15,1.7)\pscurve(2.1,1.7)(2.35,1.6)(2.45,1.4) % Ear
\end{pspicture}}

\psboxfill{\psset{unit=0.4}\SheepHead{yellow}\SheepHead{cyan}}
\Tiling[fillcyclex=2,fillloopadd=1]{(10,5)}
```

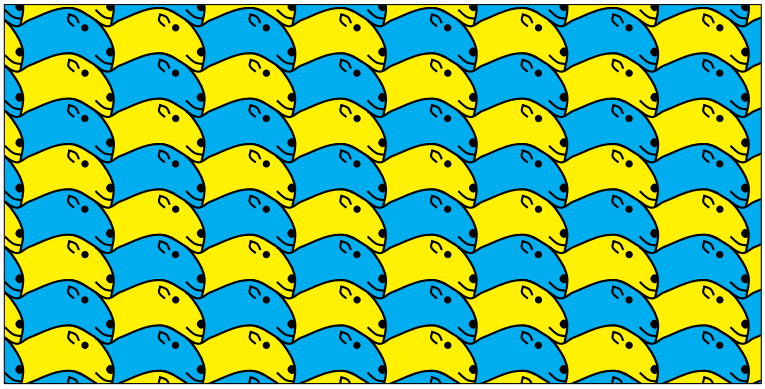


Figure 1: Tiling of *p1* category

```
\psboxfill{\psset{unit=0.4}
  \Kangaroo{yellow}\Kangaroo{red}\Kangaroo{cyan}\Kangaroo{green}%
  \scalebox{-1 1}{\rput(1.235,4.8){%
    \Kangaroo{green}\Kangaroo{cyan}\Kangaroo{red}\Kangaroo{yellow}}}}
\Tiling[fillloopadd=1]{(10,6)}
```

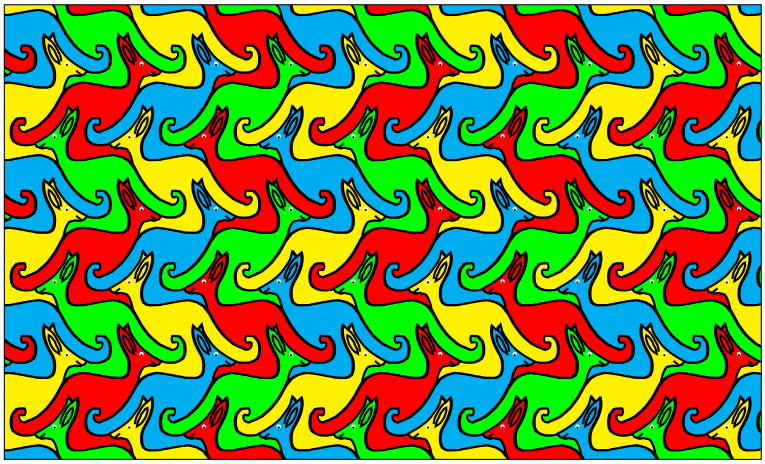
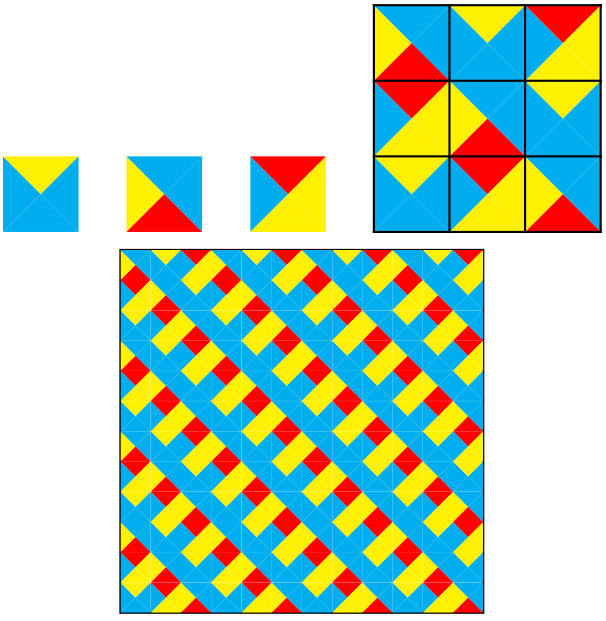


Figure 2: Tiling of *pg* category

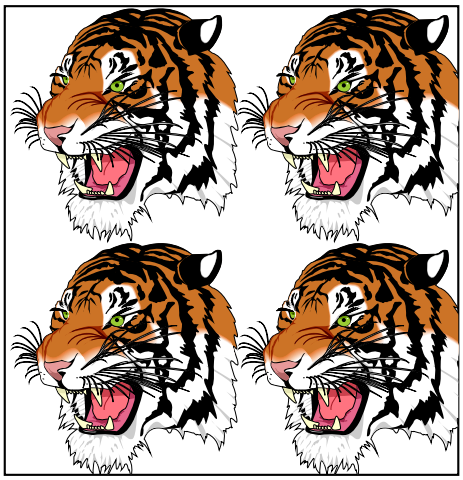
```
\newcommand{\WangTileA}{%
  \WangTile{cyan}{yellow}{cyan}{cyan}}
\newcommand{\WangTileB}{%
  \WangTile{yellow}{cyan}{cyan}{red}}
\newcommand{\WangTileC}{%
  \WangTile{cyan}{red}{yellow}{yellow}}

\newcommand{\WangTiles}[1][]{%
  \begin{pspicture}(3,3)
    \psset{ref=lb}
    \rput(0,2){\WangTileB}%
      \rput(1,2){\WangTileA}%
      \rput(2,2){\WangTileC}
    \rput(0,1){\WangTileC}%
      \rput(1,1){\WangTileB}%
      \rput(2,1){\WangTileA}
    \rput(0,0){\WangTileA}%
      \rput(1,0){\WangTileC}%
      \rput(2,0){\WangTileB}
    #1
  \end{pspicture}}

\WangTileA\hfill\WangTileB
 \hfill\WangTileC\hfill
\WangTiles[{\psgrid[subgriddiv=0,
                    gridlabels=0](3,3)}]

\vspace{2mm}
\psset{unit=0.4}
\psboxfill{\WangTiles}
\Tiling{(12,12)}
```



3.2 External graphic files

We can fill an arbitrary area with an external PostScript image. We have only, as usual, to worry about the *BoundingBox* definition if there is not one provided or if it is inaccurate, as in the case of the well known **tiger** picture (part of the Ghostscript distribution).

```
\psboxfill{%
  \raisebox{-1cm}{%
    \includegraphics[bb=17 176 562 740,
                     width=3cm]{tiger}}}
\Tiling{(6,6.2)}
```



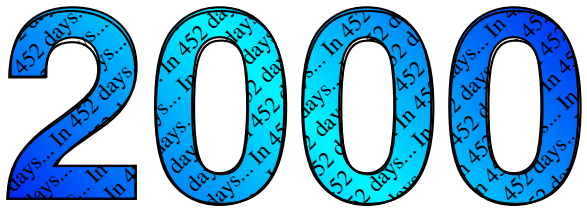
Be warned there are some types of PostScript file for which the *automatic* mode does not work, specifically those produced by a screen dump. This is demonstrated in the next example, where a picture was reduced before conversion to the *Encapsulated PostScript* format by a screen dump utility. In this case, use of the *manual* mode is the only alternative, at the price of real multiple inclusion of the EPS file. We must take care to specify the correct **fillsize** parameter, because otherwise the default values are large and will load the file too many times, perhaps just actually using a few occurrences as the other ones are clipped away...

```
\psboxfill{\includegraphics{flowers}}
\begin{pspicture}(8,4)
  \psellipse[fillstyle=boxfill,
             fillsize={(8,4)}](4,2)(4,2)
\end{pspicture}
```

3.3 Tiling of characters

We can also use the **psboxfill** macro to fill the interior of characters for special effects like the following:

```
\DeclareFixedFont{\Sf}{T1}{phv}{b}{n}{3.5cm}
\DeclareFixedFont{\Rm}{T1}{ptm}{m}{n}{3mm}
\psboxfill{\Rm In 452 days...}
\begin{pspicture*}(8,3)
  \rput(4,0.2){%
    \pscharpath[fillstyle=gradient,
      gradangle=-45,gradmidpoint=0.5,
      addfillstyle=boxfill,
      fillangle=45,fillsep=0.7mm]
      {\rput[b](0,0){\Sf 2000}}}
\end{pspicture*}
```



```
\DeclareFixedFont{\Rmm}{T1}{ptm}{m}{n}{2cm}
\psboxfill{%
 \psset{unit=0.1,linewidth=0.2pt}
 \Kangaroo{PeachPuff}\Kangaroo{PaleGreen}%
 \Kangaroo{LightBlue}\Kangaroo{LemonChiffon}%
 \scalebox{-1 1}{%
   \rput(1.235,4.8){%
     \Kangaroo{LemonChiffon}%
     \Kangaroo{LightBlue}%
     \Kangaroo{PaleGreen}%
     \Kangaroo{PeachPuff}}}}
% A kangaroo of kangaroos...
\begin{pspicture}(7.8,2)
  \pscharpath[linestyle=none,fillloopadd=1,
              fillstyle=boxfill]
             {\rput[b](4,0){\Rmm Kangaroo}}
\end{pspicture}
```

3.4 Other uses

Other uses can be imagined. For instance, we can use tilings in a sort of degenerate way to draw special lines made by a single or multiple repeating patterns. It might be just a special dashed line, as here with three different dashes:

```
\newcommand{\Dashes}{%
  \psset{dimen=middle}
  \begin{pspicture}(0,-0.5\pslinewidth)
                  (1,0.5\pslinewidth)
    \rput(0,0){\psline(0.4,0)}%
      \rput(0.5,0){\psline(0.2,0)}%
      \rput(0.8,0){\psline(0.1,0)}
  \end{pspicture}}

\newcommand{\SpecialDashedLine}[3]{%
  \psboxfill{#3}
  \Tiling[linestyle=none]
         {(#1,-0.5\pslinewidth)
          (#2,0.5\pslinewidth)}}
\SpecialDashedLine{0}{7}{\Dashes}

\psset{unit=0.5,linewidth=1mm,linecolor=red}
\SpecialDashedLine{0}{10}{\Dashes}
```

We can also use special patterns in business graphics, as in the following example generated by **PstChart** (Girou, 1993-1998) (see Figure 3).

4 "Dynamic" tiling

In some cases, tilings use *non-static* tiles, that is to say the *prototile(s)*, even if unique, can have several forms, for instance specified by different colors or rotations, not fixed before generation, or varying each time.

4.1 Lewthwaite-Pickover-Truchet tiling

We present here as an example the so-called *Truchet* tiling, which is in fact better called *Lewthwaite-Pickover-Truchet (LPT)* tiling, as explained in (Girou, To appear)[6].

The single prototile is just a square with two opposing circle arcs. This tile obviously has two positions, if we rotate it through 90 degrees (see the two tiles on the next figure). A *LPT tiling* is a tiling with randomly oriented LPT tiles. We can see that even if it is very simple in it principle, it draws sophisticated curves with strange properties.

Unfortunately, pst-fill does not work in a straightforward manner, because the **\psboxfill** macro stores the content of the tile in a TEX box, which is static. So the call of the random function is done only once, which explains why only one rotation of the tile is used for all the tiling. Only the one of the two rotations can differ from one drawing to the next ...

```
% LPT prototile
\newcommand{\ProtoTileLPT}{%
```

[6] For description of the context, history and references about Sébastien Truchet and this tiling, see (André and Girou, To appear) and specially (André, To appear), (Esperet and Girou, To appear) and (Girou, To appear).

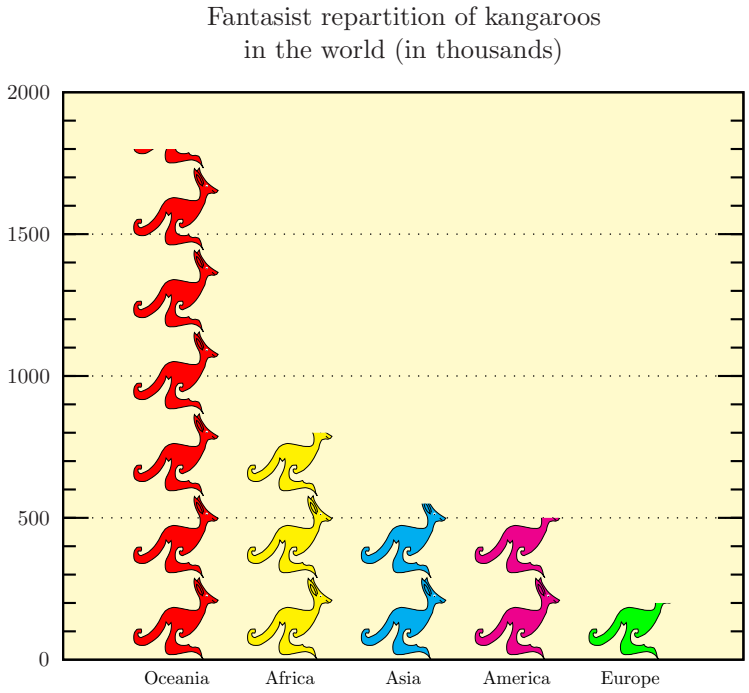
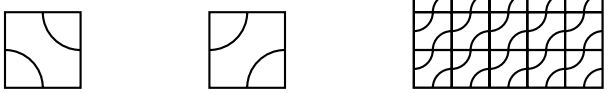


Figure 3: Bar chart generated by PstChart, with bars filled by patterns

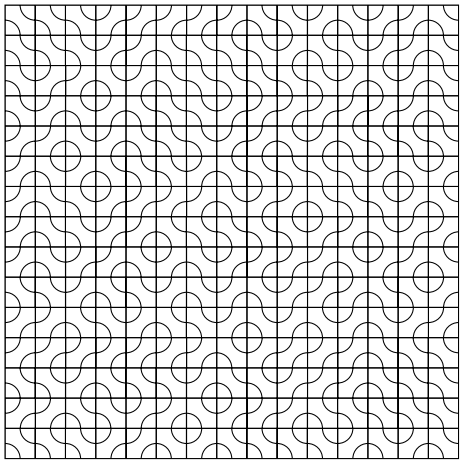
```
  \psset{dimen=middle}
  \begin{pspicture}(1,1)
    \psframe(1,1)
    \psarc(0,0){0.5}{0}{90}
    \psarc(1,1){0.5}{-180}{-90}
  \end{pspicture}}

% LPT tile
\newcount\Boolean
\newcommand{\BasicTileLPT}{%
  % From random.tex by Donald Arseneau
  \setrannum{\Boolean}{0}{1}%
  \ifnum\Boolean=0
    \ProtoTileLPT%
  \else
    \rotateleft{\ProtoTileLPT}%
  \fi}

\ProtoTileLPT\hfill
\rotateleft{\ProtoTileLPT}\hfill
\psset{unit=0.5}
\psboxfill{\BasicTileLPT}
\Tiling{(5,5)}
```



For simple cases, there is a solution to this problem using a mixture of PSTricks and PostScript programming. Here the PSTricks construction **\pscustom{\code{...}}** allows us to insert PostScript code inside the LaTeX+PSTricks one. The programming is less straightforward than solving this problem using the basic PSTricks **\multido** macro, but it has the advantage of being noticeably faster, since all tilings operations are done in PostScript, and we are not limited by TeX memory (the solution without the pst-fill package I wrote in 1995 for the colored problem was limited to small sizes for this reason). Note also that **\pslbrace** and **\psrbrace** are PSTricks macros which insert the **{** and **}** characters.

```
% LPT prototile
\newcommand{\ProtoTileLPT}{%
  \psset{dimen=middle}
  \psframe(1,1)
  \psarc(0,0){0.5}{0}{90}
  \psarc(1,1){0.5}{-180}{-90}}

% Counter to change the random seed
\newcount\InitCounter

% LPT tile
\newcommand{\BasicTileLPT}{%
  \InitCounter=\the\time
```

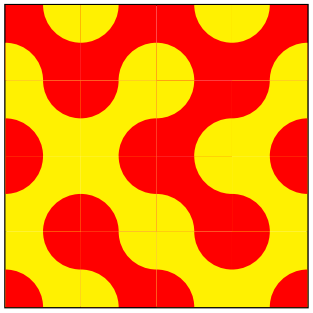
```
  \pscustom{\code{%
    rand \the\InitCounter\space
    sub 2 mod 0 eq \pslbrace}}
  \begin{pspicture}(1,1)
    \ProtoTileLPT
  \end{pspicture}%
  \pscustom{\code{\psrbrace \pslbrace}}
  \rotateleft{\ProtoTileLPT}%
  \pscustom{\code{\psrbrace ifelse}}}

\psset{unit=0.4,linewidth=0.4pt}
\psboxfill{\BasicTileLPT}
\Tiling{(15,15)}
```

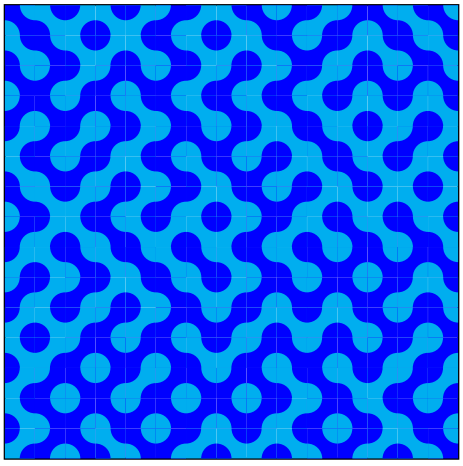


Using the very surprising fact (see (Esperet and Girou, To appear)) that the coloring of these tiles does not depend on their neighbors (even if it is difficult to believe as the opposite seems obvious!) but only on the parity of the value of row and column positions, we can directly program in the same way a colored version of the LPT tiling.

We have also introduced in the pst-fill code for *tiling* mode two new accessible PostScript variables, `row` and `column`[3], which can be useful in some circumstances, like this one.

```
% LPT prototile
\newcommand{\ProtoTileLPT}[2]{%
  \psset{dimen=middle,linestyle=none,
         fillstyle=solid}
  \psframe[fillcolor=#1](1,1)
  \psset{fillcolor=#2}
  \pswedge(0,0){0.5}{0}{90}
  \pswedge(1,1){0.5}{-180}{-90}}

% Counter to change the random seed
\newcount\InitCounter

% LPT tile
\newcommand{\BasicTileLPT}[2]{%
  \InitCounter=\the\time
  \pscustom{\code{%
    rand \the\InitCounter\space sub 2
    mod 0 eq \pslbrace
    row column add 2 mod 0 eq \pslbrace}}
  \begin{pspicture}(1,1)
    \ProtoTileLPT{#1}{#2}
  \end{pspicture}%
  \pscustom{\code{\psrbrace \pslbrace}}
  \ProtoTileLPT{#2}{#1}%
  \pscustom{\code{%
    \psrbrace ifelse \psrbrace \pslbrace
    row column add 2 mod 0 eq \pslbrace}}
  \rotateleft{\ProtoTileLPT{#2}{#1}}%
  \pscustom{\code{\psrbrace \pslbrace}}
  \rotateleft{\ProtoTileLPT{#1}{#2}}%
  \pscustom{\code{\psrbrace ifelse
                  \psrbrace ifelse}}}

\psboxfill{\BasicTileLPT{red}{yellow}}
\Tiling{(4,4)}

\vspace{2mm}
\psset{unit=0.4}
\psboxfill{\BasicTileLPT{blue}{cyan}}
\Tiling{(15,15)}
```





Another classic example is generation of coordinates and labelling for a grid. Of course, it is possible to do it directly in PSTricks using nested `\multido` commands, and it would clearly be easy to program. Nevertheless, for users who have a little knowledge of PostScript programming, this method offers an alternative which is useful for large cases,

because it will be noticeably faster and use less computer resources.

Remember here that the tiling is drawn from left to right, and top to bottom, and note that the PostScript variable `x2` contains the total number of columns.

```
% \Escape will be the \ character
{\catcode`\!=0\catcode`\\=11!gdef!Escape{\}}

\newcommand{\ProtoTile}{%
 \Square%
 \pscustom{%
  \moveto(-0.9,0.75) % In PSTricks units
  \code{%
   /Times-Italic findfont 8 scalefont setfont
   (\Escape() show row 3
   string cvs show (,) show column 3 string
   cvs show (\Escape)) show}
  \moveto(-0.5,0.25) % In PSTricks units
  \code{%
   /Times-Bold findfont 18 scalefont setfont
   1 0 0 setrgbcolor % Red color
   /center {dup stringwidth pop 2
            div neg 0 rmoveto} def
   row 1 sub x2 mul
   column add 3 string cvs center show}}}
\psboxfill{\ProtoTile}
\Tiling{(6,4)}
```

(1,1) **1**	(1,2) **2**	(1,3) **3**	(1,4) **4**	(1,5) **5**	(1,6) **6**
(2,1) **7**	(2,2) **8**	(2,3) **9**	(2,4) **10**	(2,5) **11**	(2,6) **12**
(3,1) **13**	(3,2) **14**	(3,3) **15**	(3,4) **16**	(3,5) **17**	(3,6) **18**
(4,1) **19**	(4,2) **20**	(4,3) **21**	(4,4) **22**	(4,5) **23**	(4,6) **24**

```
\newcommand{\Pattern}[1]{%
  \begin{pspicture}(-0.25,-0.25)(0.25,0.25)
    \rput{*0}{\psdot[dotstyle=#1]}
  \end{pspicture}}
\newcommand{\West}{\Pattern{o}}
\newcommand{\South}{\Pattern{x}}
\newcommand{\Central}{\Pattern{+}}
\newcommand{\North}{\Pattern{square}}
\newcommand{\East}{\Pattern{triangle}}

\newcommand{\Cross}{%
 \pspolygon[unit=0.5,linewidth=0.2,
            linecolor=red]
   (0,0)(0,1)(1,1)(1,2)(2,2)(2,1)(3,1)(3,0)
   (2,0)(2,-1)(1,-1)(1,0)}

\newcommand{\StylePosition}[1]{%
  \LARGE\textcolor{red}{\textbf{#1}}}

\newcommand{\SubDomain}[4]{%
  \psboxfill{#4}
  \begin{psclip}{\psframe[linestyle=none]#1}
    \psframe[linestyle=#3](5,5)
    \psframe[fillstyle=boxfill]#2
  \end{psclip}}

\newcommand{\SendArea}[1]{%
  \psframe[fillstyle=solid,fillcolor=cyan]#1}

\newcommand{\ReceiveData}[2]{%
  \psboxfill{#2}
  \psframe[fillstyle=solid,fillcolor=yellow,
           addfillstyle=boxfill]#1}

\newcommand{\Neighbor}[2]{%
 \begin{pspicture}(5,5)
   \rput{*0}(2.5,2.5){\StylePosition{#1}}
   \ReceiveData{(0.5,0)(4.5,0.5)}{\Central}
   \SendArea{(0.5,0.5)(4.5,1)}
   \SubDomain{(5,2)}{(0.5,0.5)(4.5,3)}
             {dashed}{#2}%
   % Receive and send arrows
   \pcarc[arcangle=45,arrows=->]
         (0.5,-1.25)(0.5,0.25)
   \pcarc[arcangle=45,arrows=->,
          linestyle=dotted,dotsep=2pt]
         (4.5,0.75)(4.5,-0.75)
 \end{pspicture}}

\psset{dimen=middle,dotscale=2,fillloopadd=2}
\begin{pspicture}(-5.7,-5.7)(5.7,5.7)
  % Central domain
  \rput(0,0){%
    \begin{pspicture}(5,5)
      % Receive from West, East, North and S.
      \ReceiveData{(0,0.5)(0.5,4.5)}{\West}
      \ReceiveData{(4.5,0.5)(5,4.5)}{\East}
      \ReceiveData{(0.5,4.5)(4.5,5)}{\North}
      \ReceiveData{(0.5,0)(4.5,0.5)}{\South}
      % Send area for West, East, North and S.
      \SendArea{(0.5,0.5)(1,4.5)}
      \SendArea{(4,0.5)(4.5,4.5)}
      \SendArea{(0.5,0.5)(4.5,1)}
      \SendArea{(0.5,4)(4.5,4.5)}
      % Central domain
      \SubDomain{(5,5)}{(0.5,0.5)(4.5,4.5)}
                {solid}{\Central}
      % Redraw overlapped lines
      \psline(1,0.5)(1,4.5)
      \psline(4,0.5)(4,4.5)
      % Two crosses
      \rput(1.5,4){\Cross}
      \rput(2,2){\Cross}
    \end{pspicture}}
  % The four neighbors
  \rput(0,5.5){\Neighbor{N}{\North}}
  \rput{-90}(5.5,0){\Neighbor{E}{\East}}
  \rput{90}(-5.5,0){\Neighbor{W}{\West}}
  \rput{180}(0,-5.5){\Neighbor{S}{\South}}
\end{pspicture}
```

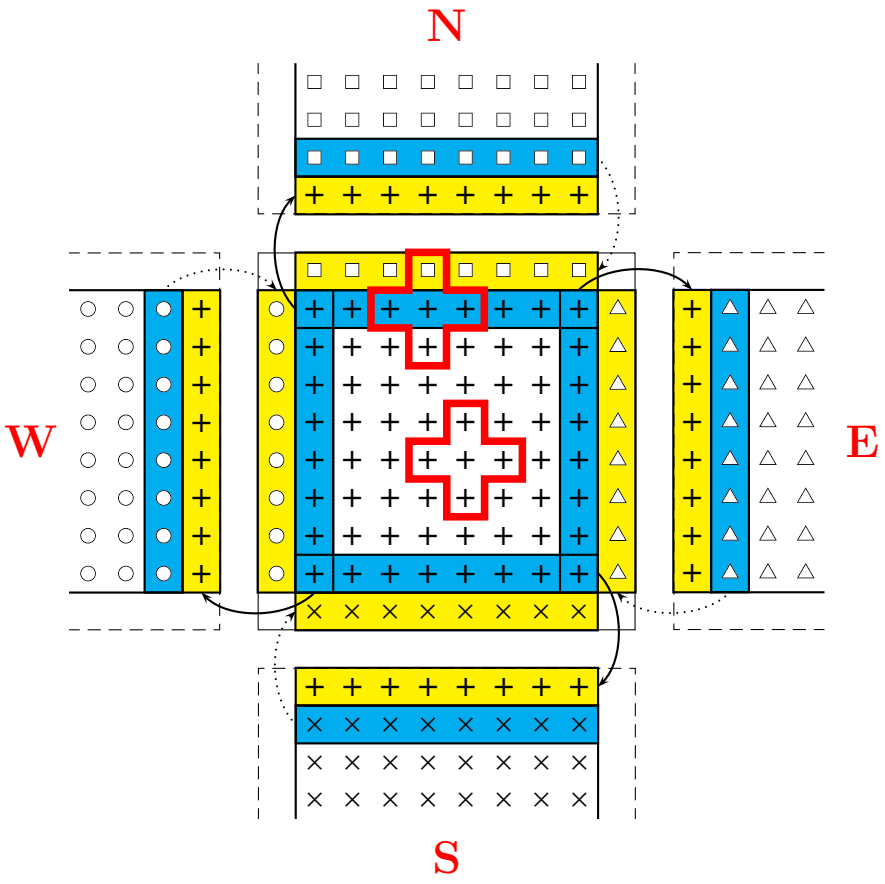


Figure 4: Communication scheme to solve the Poisson equation on a distributed memory computer

4.2 A complete example: the Poisson equation

To finish, we show in Figure 4 a complete real example, a drawing to explain the method used to solve the Poisson equation by a domain decomposition method, adapted to distributed memory computers. The objective is to show the communications required between processes and the position of the data to exchange. The code (listed below) also shows some useful and powerful techniques for PSTricks programming (look especially at the way some higher level macros are defined, and how the same object is used to draw the four neighbors).

References

Adobe, Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, 2 edition, 1995.

Chailloux, Emmanuel, G. Cousineau, and A. Suárez. "Programmation fonctionnelle de graphismes pour la production d'illustrations techniques". *Technique et science informatique* **15**(7), 977–1007, 1996.

Deledicq, André. *Le monde des pavages*. ACL Éditions, 1997.

Girou, Denis. "PstChart. Business charts in (LA)TEX + PostScript with PSTricks". `http://www.tug.org/applications/PSTricks/PstChart`, 1993-1998.

Girou, Denis. "Présentation de PSTricks". *Cahiers GUTenberg* **16**, 21–70, 1994.

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LATEX Graphics Companion*. Addison-Wesley, 1997.

Grünbaum, Branko and G. Shephard. *Tilings and Patterns*. Freeman and Company, 1987.

Hoenig, Alan. *TEX Unbound: LATEX & TEX Strategies, Fonts, Graphics, and More*. Oxford University Press, 1998.

Rose, Kristoffer H. and R. Moore. "XY-pic. Pattern and Tile extension". Available from CTAN, `macros/generic/diagrams/xypic`, 1991-1998.

van der Laan, Kees. “Paradigms: Just a little bit of PostScript”. *MAPS* **17**, 137–150, 1996.

van Zandt, Timothy. “PSTricks. PostScript macros for Generic TEX”. Available from CTAN, `graphics/pstricks`, 1993.

van Zandt, Timothy and D. Girou. “Inside PSTricks”. *TUGboat* **15**(3), 239–246, 1994.

Wang, Hao. “Games, Logic and Computers”. *Scientific American* pages 98–106, 1965.

◇ Denis Girou
CNRS/IDRIS — Centre National de la Recherche Scientifique / Institut du Développement et des Ressources en Informatique Scientifique
B.P. 167
91403 Orsay cedex
France
`Denis.Girou@idris.fr`

Book Reviews

Book review: *TEX Unbound*, by Alan Hoenig

Michael Doob

Alan Hoenig, *TEX Unbound: LATEX & TEX Strategies for Fonts, Graphics, & More*. Oxford University Press, New York, 1998, ISBN 0-19-509686-X.

It is a daunting task to try to describe in some detail the principal extensions available to enhance the typesetting abilities of TEX. Some, such as MetaFont, go back to the very origins of TEX; some, such as MetaPost, are adaptations of older programs to newer technologies; some, such as those involving hypertext references or using TEX to create documents for use on the internet, are responses to very current changes in methods of transmitting information. *TEX Unbound* is the first book (all 600 pages of it) that makes a serious attempt at fulfilling this task.

Reviewing a book about TEX is necessarily a multifaceted task. In addition to considering the usual content, one is inevitably drawn to the typesetting and the quality (or possible lack thereof) of design. For *TEX Unbound* it is obvious that great effort has been put into producing an attractive book. The text has been set in Adobe Garamond, a refreshing change from the plethora of books typeset using Computer Modern; the style is generally attractive and consistent. Nonetheless, your reviewer did find a small pride of typographical errors, but surprisingly few for a book of this size (in fact this review came from reading the final proofs of the book; perhaps there are even fewer typos now). This is not to say that all is perfect: the author spends some time describing f-words; these are defined by him to be words that end in the letter f (just to be sure that the reader's imagination doesn't run wild with any possibly prurient thoughts). Given this emphasis, it is surprising that whenever the phrase "of TEX" appears, the close kerning stops the reading process and is just plain annoying. That having been said, the book is attractive, and this in itself is helpful as the author gently introduces some principles of good style. A particularly interesting example of this is the "Rogues' Gallery" of 28 different combinations of mathematical fonts (Computer Modern, Math Times, Euler, Lucida New Math, Mathematica) and text fonts (Computer Modern, Times New Roman, Palatino, Baskerville, Galliard, Lucida Bright, Lucida Sans). The results (intentionally) range from attractive to disastrous. Looking over these samples carefully really clarifies many typesetting issues, especially in the cases where the math and text italic fonts are the same. Other interesting applications of TEX showed up from time to time in the text, e.g. ct ligatures, and provided pleasant surprises to the reader.

The intended audience of this book is clear from the topics covered: installing and running TEX, using MetaFont and MetaPost, installing new fonts, the use (via nontrivial examples) of virtual fonts, and various method of graphic inclusions including the use of the LATEX picture environment, epic, eepic, PICTEX, MetaFont, MetaPost, mfpic, and PSTricks. A comfortable working knowledge of TEX is generally assumed, but no higher expertise is demanded. It must be said that there is a bias towards the UNIX operating system, and towards the C shell within it. This is clearly not a first-level book, but neither does it require any system-level knowledge of either TEX or the underlying operating system (a somewhat more advanced knowledge is required to understand the Perl scripts, but they aren't very numerous so this might be considered "knit"-picking).

The first 10% of the book describes the main principles of running TEX and the sources for TEX: the internet and the CD-ROM. It also describes newsgroups, some tools (mainly ftp), CTAN and the

different TEX users groups. It is really more of a refresher, but does describe a number of relatively new web sites that might be of interest to the more seasoned user of TEX.

The real nuts and bolts of *TEX Unbound* starts with the discussion of MetaFont and MetaPost. Since there are relatively few introductions to MetaFont, especially *vis-à-vis* its companion program TEX, it is fortunate that this one starts from the beginning and explains how to construct basic shapes using lines and (Bézier) curves, how to change pen shapes, and how to adjust the parameters of MetaFont; this is really welcome. The discussion of bitmapped versus outline fonts is also useful (although the lack of any discussion of hinting of outline fonts is particularly unfortunate). Many samples of MetaFont fonts are given; it would have been useful to have samples of a few letters with both their control points and MetaFont code displayed. In the latter part of *TEX Unbound* there are a number of excellent examples of MetaFont and MetaPost code; the results are elegant, beautiful and reflect both mathematical and artistic beauty (it should be noted that the author is clearly the Captain Ahab of the TEX world pursuing the perfect graphic output).

TEX Unbound then proceeds with a short introduction to LATEX followed by some elementary interactions with other types of software, e.g. Mathematica. Along with the introduction to plain TEX in the appendix, these seem rather out of place, having neither the sophistication nor excitement of the rest of the book.

On the other hand, the description of font selection is excellent and fills a much-unneeded gap in the literature. The use of fonts other than Computer Modern with LATEX, especially with the new font selection scheme (NFSS) and its extensions to LATEX 2_ε, is somewhat byzantine, and having the concepts of font selection, encodings, and naming schemes explained in one consistent chapter is welcome.

The topic of virtual fonts follows naturally from font selection. Again, the reason behind and needs for virtual fonts and various encodings are explained, and methods of installing them are discussed (with examples). Complete descriptions of several projects are given: strikeout fonts, underline fonts, and composite fonts. The examples displayed in the accompanying figures are wonderful. (A personal *steckenpferd* of the reviewer: using `\bar` as a math accent gives the same size accent for all letters of all widths so that i and M get the same accent. Using `\overline` produces an accent that is too wide. Why not a virtual overline font?) While several useful tools for manipulating virtual fonts are described, the basic structure of the `vfl` files remains unmentioned. Even a brief description would convey better the power of virtual fonts.

Want to install new math fonts? Software for doing this (available on CTAN) is described and used to produce the aforementioned Rogues' Gallery.

Finally, there are descriptions and extensive examples of various methods of inserting graphics inclusions into TEX files. These include some that are internal to TEX (in the sense that auxiliary macros are input and make graphic commands available) such as the LATEX picture envrionment, PSTricks, PSfrag, and PICTEX; others create external files that can then be processed by TEX such as `bm2font` or mfpic. *TEX Unbound* is certainly the best collective description of the various methods of adding graphics. Perhaps a little more might have been said to compare the relative strengths and weaknesses of the various programs.

Finally, a few words must be said about the writing style of the author. This is a highly technical book, and such books are often somewhat unpleasant to read. Sometimes it seems that an author is trying to write a entire book without using an adverb. In contrast, this book is well written using a good deal of style and humour. Much of the material presented in this book could have been dry and repetitive; instead the descriptions and the examples used are attractive and interesting. Typical is the short example about buckling of beams under compression entitled "Necking in bars". However, it probably takes the condescension of a true New Yorker to appreciate the comments about Hoboken.

All in all this is an excellent addition to the TEX references available. Anyone who uses TEX on a day to day basis will definitely want it. Anyone who is even mildly interested in the limits of TEX will also want this book. For a TEXie it's a good read.

◇ Michael Doob
Department of Maths and Astronomy
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
doob@cc.umanitoba.ca

Fonts

An Overview of Indic Fonts for TEX

Anshuman Pandey

1 Introduction

Many scholars and students in the humanities have preferred TEX over other "word processors" or document preparation systems because of the ease TEX provides them in typesetting non-Roman scripts, the availability of TEX fonts of interest to them, and the ability TEX has in producing well-structured documents.

However, this is not the case amongst Indologists. The lack of Indic fonts for TEX and the perceived difficulty of typesetting them have often turned Indologists away from using TEX. Little do they realize that TEX is *the* foremost tool for developing Indic language/script documents. With an increase over the past few years in the development and availability of Indic language and font packages, the introduction of other fonts and style packages, the flexibility of the LATEX 2ε system, and the creation of TUGIndia (which may revolutionize the typesetting of Indic scripts) there is now even more reason for Indologists to implement TEX in their work.

There are roughly thirteen major Indic scripts (Tibetan is included in this list) which are used throughout South Asia to write the major languages and dialects of the region. As of this article all of these major scripts can be typeset with TEX, the exception being Assamese (see Section 6).

Not only is it fascinating that the major scripts of South Asia can be typeset with TEX, but the ease with which such a task can be accomplished is itself an amazing feat. Anyone who has ever tried writing a document with multiple non-Roman scripts and diacritic text in an environment other than TEX understands the complexity of such a task. TEX takes the user beyond such difficulties by facilitating the implementation of multiple scripts without the hassle of worrying about various fonts and their encodings, manual font switching, and other such hindrances to productivity caused by common "word processors".

TEX enables the incorporation of several non-Roman scripts within a single document through transliterated input of the scripts. Indic scripts are based on the phonetic template of the languages they represent, a template which is uniform in both the Indo-Aryan and Dravidian language families of India. Such uniformity in phonetics is reflected in orthography, which in turn enables all scripts to be transliterated through a single scheme. This uniformity has subsequently been reflected in the transliteration schemes of the Indic language/script packages.

Most packages have their own transliteration scheme, but these schemes are essentially variations on a single scheme, differing merely in the coding of a few vowel, nasal, and retroflex letters. Most of these packages accept input in one of the two primary 7-bit transliteration schemes — ITRANS or Velthuis — or a derivative of one of them. There is also an 8-bit format called CS/CSX which a few of these packages support. CS/CSX is described in further detail in Section 3.

2 The Fonts and Packages

Figure 1 shows examples of the various fonts described in this article. Table 1 lists the sites from which all of the fonts and packages described in this article are available.

3 CS/CSX

CS/CSX (Classical Sanskrit/Classical Sanskrit eXtended) is the closest thing to an accepted standardization of 8-bit transliteration of Indic scripts. Adopted in 1990 at the 8th World Sanskrit Conference in Vienna, CS/CSX enables Indologists to exchange electronic data in a variety of platform-independent media.

CS/CSX is an encoding convention based on IBM Code Page 437. CS is a basic inventory of diacritic letters which are traditionally used to transliterate Sanskrit written in the Devanagari script. CSX is an extension of this basic inventory to include accented and other characters. Contrary to what the name indicates, the inventory of CS/CSX characters is not limited to Sanskrit, and may be used to transliterate other Indic languages.

Introductory information on CS/CSX is found in an article by Dominik Wujastyk titled *Standardization of Sanskrit for Electronic Data Transfer and Screen Representation* [1]. This document, as well as supporting screen fonts and drivers for DOS-based machines, is available from the INDOLOGY site as well as from CTAN.

Various fonts and packages have been developed which enable TEX to typeset documents encoded in the CS/CSX convention. These are enumerated below:

`cp437csx` The file `cp437csx.def` is an input encoding definition file for LATEX 2ε which enables

Figure 1: Example of Indic Fonts

CS/CSX encoded documents to be typeset in LaTeX without need for conversion. The file is available from CTAN as part of the csx package.

csxtimes John Smith has made available the font metrics and virtual fonts of the commonly-used PostScript fonts re-encoded with the CSX encoding. The use of these fonts enable CS/CSX documents to be typeset directly by TeX without the need for any conversions. This is facilitated through the `csx.def` file which provides the CS/CSX input encoding definitions for standard LaTeX and for the standard TeX fonts.

cscharter Dominik Wujastyk produced an extension and re-encoding of the Bitstream Charter font according to the CS/CSX convention called 'CS Charter'. Users should note that 'CS Charter' supports *only* the characters of the Classical Sanskrit encoding and does not support the Extended encoding. This font is available from the INDOLOGY site and is also bundled with the ITRANS package.

csutopia Dominik Wujastyk produced an extension and re-encoding of the Adobe Utopia font according to the CS/CSX convention called 'CS Utopia'. Users should note that 'CS Utopia' supports *only* the characters of the Classical Sanskrit encoding and does not support the Extended encoding. This font is available from the INDOLOGY site and is also bundled with the ITRANS package.

wnri Thomas Ridgeway developed a package called 'Washington Roman Indic' which contained a family of fonts based on Computer Modern and which were encoded with the CS/CSX and other supplementary conventions. I recently revised the package for use with LaTeX 2$_\varepsilon$ and added a style file and input encoding definition file which does away with the need for the fonts.

Ridgeway also developed screen fonts and drivers for wnri for DOS-based machines. The updated package is available from CTAN.

4 babel

Recently Jun Takashima introduced two Indic language modules to the babel fold. These packages enable support for Romanized Sanskrit and for Kannada in both the original and Roman scripts. Please refer to Section 5.5 for a description of Takashima's Kannada package.

skthyph This module provides the hyphenation patterns for Romanized Sanskrit. As of this article these files are not distributed with the current version of babel but will be included in the next release. This module is presently available only from the developer's FTP site.

5 ITRANS

The ITRANS package developed by Avinash Chopde is the primary component of an on-going project to make the typesetting of all Indic scripts possible by means of a single tool. As of this article, ITRANS supports the Bengali, Devanagari, Gujarati, Gurmukhi, Kannada, Tamil, and Telugu scripts. It also supports the 'CS Utopia' diacritic Roman font.

In addition to the default TEX output, ITRANS can produce direct HTML and PostScript output from the input file. ITRANS versions for both DOS and UNIX systems are available from the developer's website.

5.1 Bengali

arosgaon The *AroSgaon* package was developed by Muhammad Masroor Ali as an extension to the 'SonarGaon' HP Laserjet softfont designed by Anisur Rahman. This package contains an ITRANS module which provides glyphs for certain characters not available in the original 'SonarGaon' font. Although this package was designed as an supplement to the Bengali support of ITRANS, it may be used as an independent package with LATEX 2ε. The 'SonarGaon' font is not bundled with arosgaon or ITRANS, and must be obtained separately.

itrans ITRANS provides support for the Bengali script through the 'ItxBengali' PostScript Type 1 font developed by Shrikrishna Patil.

5.2 Devanagari

devnag The *Devanagari for TEX* package developed by Frans Velthuis was the original package for Devanagari. This package uses the 'Devnag' font also developed by Velthuis which contains the characters required to typeset Sanskrit, Hindi, Marathi, and any other languages which use the Devanagari script. The font 'Devnag Pen' developed by Thomas Ridgeway is a variation on 'Devnag' which resembles Devanagari written with an ordinary pen and is bundled with devnag.

Dominik Wujastyk, John Smith, myself, and a few others have recently upgraded devnag for use with LATEX 2ε. The package is now NFSS-compliant.

sanskrit The *Sanskrit for LATEX 2ε* package developed by Charles Wikner is an extensive package which enables the typesetting of Devanagari text with Vedic accents and other special characters not supported by the devnag package. Numerous options may be set in regard to transliteration, alternate characters, inter-character spacing, and other preferences. Only support for the Sanskrit language is available. The font 'Sanskrit', also developed by Wikner, is bundled with the package. It is a rather complete font in that it contains many complex ligatures and variants which enable excellent typesetting of Devanagari. This package is available from CTAN.

itrans Four fonts provide Devanagari support in ITRANS: the 'Devnag' and 'Devanagari Pen' fonts described above and two more called 'Devnac' and 'Xdvng'. The 'Devnac' font is a PostScript Type 3 font developed by Avinash Chopde for the ITRANS package. 'Devnac' was developed to enable users unfamiliar with TEX to still produce texts in Devanagari through the "dumb textual interface" mode of ITRANS. 'Xdvng' is a PostScript Type 1 font, rendered by Sandeep Sibal from Velthuis's 'Devnag' METAFONT, which enables users to produce direct HTML output of Devanagari text in addition to the standard TEX and PostScript output.

jtex Developed by John Smith, *Jaisalmer TEX* is a Perl preprocessor which enables Ken Bryant's 'Jaisalmer' font to be used with TEX. The font is not freely available and must be purchased from Bryant for a nominal fee. More information about this package is available from the URL given in Table 1.

5.3 Gujarati

itrans Currently the only package available for Gujarati is ITRANS, which uses the 'ItxGujarati'

PostScript Type 1 font developed by Shrikrishna Patil. The 'ItxGujarati' font is bundled with ITRANS.

5.4 Gurmukhi

gurmukhi Developed by Amarjit Singh, the *Gurmukhi for TEX* package enables support for the Gurmukhi script in Plain TEX. The package includes a preprocessor and the 'Gurmukhi' METAFONT also developed by Singh. This package has not been updated since the initial release of gurmukhi in October 1995. This package is available from CTAN.

itrans ITRANS supports Gurmukhi through the PostScript font 'Punjabi' developed by Hardip Singh Pannu. The font metric file module used by ITRANS for Gurmukhi was developed by me.

5.5 Kannada

kannadatex The *KannadaTEX* package developed by Jun Takashima provides Kannada support for LATEX and babel. The package includes the METAFONT source for the 'AA Institute Kannada' font, a preprocessor, and a hyphenation pattern for the Kannada language. The Kannada hyphenation patterns will be included in the next release of babel. The package is available from the developer's FTP site.

kantex The *KanTEX* package developed by G. S. Jagadeesh and Venkatesh Gopinath enables the Kannada script to be typeset with TEX. The 'Kannada' METAFONT is bundled with kantex. This package is available from the developers' website.

itrans To typeset Kannada ITRANS uses a modified module of the kantex package. The 'Kannada' METAFONT is bundled with ITRANS.

5.6 Malayalam

malayalam The *Malayalam-TEX* package was developed by Jeroen Hellingman. The malayalam package enables text in to be typeset in both the traditional and reformed Malayalam scripts. The package includes a preprocessor and fonts in the regular, slanted, bold, and calligraphic typefaces. This package also supports the Devanagari and Tamil scripts through the 'Devnag' and 'Washington Tamil' fonts. A modified version of the Velthuis scheme is used for transliterated input. This package is available from CTAN.

5.7 Oriya

oriyatex The *Oriya-TEX* package is being developed by Jeroen Hellingman. The oriyatex package currently provides the two fonts 'Cuttack' and 'Konark' designed by Hellingman. The first is a regular face while the second is a calligraphic variation of the former. The preprocessor is still being developed. *Oriya-TEX* uses a modified version of the Velthuis scheme for transliterated input. The beta-version of this package is available from CTAN.

5.8 Perso-Arabic

In South Asia the Perso-Arabic script is used predominantly to write the Urdu, Sindhi, and Kashmiri languages. Each language has distinct forms for certain letters, but the character shapes are generally identical.

arabtex *ArabTEX* was developed by Klaus Lagally and functions through a system of style files, eliminating the use for a preprocessor. *ArabTEX* supports the typesetting of almost all languages whose orthography is based on the Perso-Arabic system. As concerns South Asian orthography *ArabTEX* currently supports Urdu and Sindhi; the extensions for Kashmiri are being developed.

Currently only the font 'Naskh' is supported. 'Naskh' is a METAFONT designed by Lagally and is based on the *naskhī* style of Arabic calligraphy. A font based on the *nast'alīq* style, used predominantly for Urdu, has not yet been developed.

A variety of both 7- and 8-bit input encoding schemes are supported, yet the *ArabTEX* encoding itself (based on the transliteration scheme of the *Zeitschrift der Deutschen Morgenländischen Gesellschaft*) is the only scheme which fully accommodates the extended Perso-Arabic script used by Indic languages.

This package is available from Lagally's FTP site as well as from CTAN.

5.9 Sinhalese

sinhala The *Sinhalese TEX* package has two different versions. The first and original package was developed by Yannis Haralambous. The second version is a modification of the first by Prasad Dharmasena to accommodate a second transliteration scheme called 'Samanala'. Both versions require the use of the *Indica* preprocessor, bundled with the package. The second version includes a DOS executable of the *Indica*

program. The original package is available from the INDOLOGY site and from CTAN; and the second version from Dharmasena's FTP site.

5.10 Tamil

tamilize *Tamilize* is a preprocessor developed by Thomas Ridgeway for the 'Washington Tamil' METAFONT. This font was designed at the former *Humanities and Arts Computing Center*[1] at the University of Washington for a Tamil-English dictionary project. It is no longer supported by the University of Washington. The tamilize package is available from CTAN.

itrans The interface provided by ITRANS for typesetting Tamil makes it easier to use than the *Tamilize* program. ITRANS makes use of the *Washington Tamil* font as well.

5.11 Telugu

telugutex The *TeluguTEX* package was developed Lakshmi V. S. Mukkavilli. It uses the 'Telugu' METAFONT also developed by Mukkavilli. The package is available from CTAN.

itrans ITRANS supports Telugu through a modified module of the telugutex package. ITRANS uses the 'Telugu' METAFONT.

5.12 Tibetan

The three Tibetan packages for TEX are essentially revisions and enhancements of the original package. These three packages are called sparkes, sirlin, and steiner after their developers and are all available from CTAN.

sparkes *Tibetan LATEX* was the original Tibetan package written by Jeff Sparkes. The package includes a preprocessor and the 'Tibetan' font.

sirlin Sam Sirlin fixed minor bugs in the sparkes package and provided an improved preprocessor. This package uses a METAFONT developed by Sirlin called 'GTibetan' and requires the sparkes package.

steiner *TEXTib* or *Tibetan Transcript Translator*, developed by Beat Steiner, introduces a major overhaul of the first two packages. Steiner created an improved preprocessor enabling support for different input schemes, better handling of ligatures, and more logical typesetting of Sanskrit in the Tibetan script. The steiner package uses the 'GTibetan' font and requires the sirlin package.

[1] The *Humanities and Arts Computing Center* was replaced by the *Center for Advanced Research in the Arts and Humanities*.

6 What's Next?

1. TEX support is currently being developed for the following Indic scripts.

   Assamese A LATEX package for Assamese is currently being developed by Jugal Kalita. Information about the font design and package is available from the URL given in Table 1.

   Brahmi The Brahmi script is the ancestor from which all scripts mentioned in this article are derived. This script was employed by, and perhaps even developed under, the Mauryan king Aśoka to have his edicts inscribed during the third century BCE. I am designing a METAFONT of the Brahmi script called 'Washington Brahmi'. The style is an approximation of the early Mauryan Brahmi style and based upon the characters found on the inscriptions at Gīrnār. The font is not yet complete, however a brief description and examples of the font are available from my website.

2. ISO/TC46/SC2/WG12 (the Working Group for the Transliteration of Indic Scripts) is progressing toward a standardized 7- and 8-bit scheme. Perhaps all Indic TEX packages will support this standard.
3. Unicode? Will there be a need for these packages once Unicode is firmly established?

References

[1] Wujastyk, Dominik. *Standardization of Romanized Sanskrit for Electronic Data Transfer and Screen Representation* [results of a session held at the 8th World Sanskrit Conference, Vienna, 1990], in *Sesame Bulletin* 4(1), 1991, pp. 27-29. Also available as a PostScript document from `CTAN/fonts/csx/csx-doc.ps`.

◇ Anshuman Pandey
University of Washington
Department of Asian Languages
and Literature
225 Gowen Hall, Box 353521
Seattle, WA 98195
apandey@u.washington.edu
http://weber.u.washington.edu/~apandey/

arabtex	ftp://ftp.informatik.uni-stuttgart.de/pub/arabtex/
	CTAN/language/arabtex/
arosgaon	CTAN/language/bengali/arosgn/
assamese	http://www.acsu.buffalo.edu/~talukdar/assam/language/assamlang.html
brahmi	http://weber.u.washington.edu/~apandey/texts/
csx	CTAN/fonts/csx/
csxtimes	ftp://bombay.oriental.cam.ac.uk/pub/john/software/fonts/
devnag	CTAN/language/devanagari/
gurmukhi	CTAN/language/gurmukhi/
itrans	http://www.aczone.com/itrans/
jtex	ftp://bombay.oriental.cam.ac.uk/pub/john/software/jtex/
kannadatex	ftp://ftp.aa.tufs.ac.jp/pub/tool/TeX/languages/kannada/
kantex	http://langmuir.eecs.berkeley.edu/~venkates/
malayalam	CTAN/language/malayalam/
oriyatex	CTAN/language/oriya/
sanskrit	CTAN/language/sanskrit/
sinhala	ftp://ftp.mq.edu.au/home/vsaparam/sinhala_tex/
	CTAN/language/sinhala/
sirlin	CTAN/language/tibetan/sirlin/
skthyph	ftp://ftp.aa.tufs.ac.jp/pub/tool/TeX/languages/sanskrit/
sonargaon	http://www.winsite.com/info/pc/win3/fonts/sgaon.zip
sparkes	CTAN/language/tibetan/original/
steiner	CTAN/language/tibetan/steiner/
tamilize	CTAN/language/tamil/tamilize/
telugutex	CTAN/language/telugu/
wnri	CTAN/fonts/wnri/
'bombay'	ftp://bombay.oriental.cam.ac.uk/pub/john/
INDOLOGY	http://www.ucl.ac.uk/~ucgadkw/indology.html
CTAN	ftp://ftp.tex.ac.uk/tex-archive/
	ftp://ftp.dante.de/tex-archive/
	ftp://ftp.tug.org/tex-archive/

Table 1: Package Sites

Diversity in math fonts

Thierry Bouche

Abstract

We will examine the issues raised when modifying (LA)TEX fonts within math environments, and attempt to suggest effective means of accessing a larger variety of font options, while avoiding typographic nonsense.

> "*Don't mix faces haphazardly when specialized sorts are required*"
>
> — Robert Bringhurst [9]

1 Stating the problem

The advent of LaTeX 2ε has resulted in a type of 'standardizing' of font selection schemes (NFSS, in other words). The advantages are many, but the main one for me is this: unlike other software that's more expensive and of poorer quality, changing fonts is as easy as changing your socks. In fact, the 'heroic' days of *plain* are just a memory, where changing from the default \textfont0 meant generating a new format, not to mention various encodings ... The temptation to play is therefore very great, especially if you want to break with the monotony of countless preprints and other (LA)TEX documents.[1] I won't say much about anything other than PostScript fonts, mainly because I can only test my hypotheses on them. Sebastian Rahtz' *psfonts* now allows anyone equipped with a PostScript printer to choose their text fonts for use with LaTeX: Times, Bookman, New Century Schoolbook, Palatino. You can ftp to CTAN sites to pick up everything you need to use a wide variety of commercial fonts. Alan Jeffrey's *fontinst* program makes it easier to create the interface needed to use PostScript fonts with LaTeX. The choices are almost limitless, with some 20,000 fonts to choose from for your document.[2]

Unfortunately, if your document has equations, this diversity is pretty much an illusion. There are actually very few math fonts, and of these, only a few are designed to work with TEX. To my knowledge, here are the font collections that provide a significant set of mathematical glyphs:

The native TEX fonts: these are, of course, cmmi/cmsy/cmex, with the addition of the AMS symbol fonts (msam/msbm);

Some non-native TEX fonts: initially developped in MetaFont format to complement the Concrete text fonts by Knuth, are the Euler fonts, which aren't coded in quite the same way as the standard TEX fonts, and do not really provide a replacement, as so many extra symbols are missing. There is an option available on CTAN, euler.sty by F. Jensen and F. Mittelbach, which makes installing the Euler fonts easier. However, the Eulers weren't designed to be combined with any particular text fonts — the best you can say is that they 'work' with Bitstream Charter or, of course, Concrete. Karl Berry has recently used Euler with Palatino, a valid combination since both font families were designed by Hermann Zapf. U. Vieth designed a math font based on Knuth's Concrete fonts. It is also missing many variants and glyphs, but enjoying an NFSS support package;

MathTime: this family is a full alternative to the CM collection, but is missing some glyphs from the AMS collection;

Lucida New Math: this family is as comprehensive as possible;

PostScript Symbol font: almost as widespread as Courier, it yields upright Greek letters, and includes a number of basic math symbols;

Mathematical Pi: usually used by (photo)typesetting software, this is a collection of six fonts whose glyph set is rather extensive;

and some more: let us also notice that many scientific software programs use proprietary fonts to display equations on-screen or print them on paper.[3] Not to mention the specific proprietary fonts used by some publishers.

In current (LA)TEX, a math font family needs to have at least three members: math italic (cmmi is the default), symbols (cmsy), and extensions for building different-sized symbols (cmex). Taking design consistency and glyph set exhaustivity into account, of the fonts listed above, we are effectively left with with three font families, alternatives which are both complete and unified (well, one less so than the others):

[1] Note that 'LaTeX' can be understood as having two relatively independent meanings: it's a program to typeset scientific texts, and it's also a standard in the electronic exchange of documents. This article is concerned with the former: producing documents which are to be printed and thereby benefit from typographic programs adapted to the purpose.

[2] This count, based on Unique IDs, is relatively outdated, as recent fonts IDs would imply that we've reached a count approaching 90,000!

[3] Among them, Mathematica provides a font set with a rather rich set of glyphs. U. Vieth has made TEX virtual fonts for them, along the lines of mathptm; see below.

CONJECTURE 1. — *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card}\Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

$$(1) \qquad x = 2\pi z \Longleftrightarrow \operatorname{card}\Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

$$(2) \qquad \prod_{j\geqslant 0} \left(\sum_{k\geqslant 0} f_{jk} z^k \right) = \sum_{k\geqslant 0} z^n \left(\sum_{\substack{k_0,k_1,\ldots\geqslant 0 \\ k_0+k_1+\cdots=n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product ffi, $\{\overbrace{g,\ldots,g}^{k\ g}, \overbrace{h,\ldots,h}^{\ell\ h}\}$ ($k+\ell$ elements) *taken in the basis* $(\vec{\imath},\vec{\jmath})$.

Moreover, eq. (1) yields $$\pm\sqrt{\begin{vmatrix} x_1-x_2 & y_1-y_2 & z_1-z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 1: The (LA)TEX default: Computer Modern.

- the standard fonts based on Knuth's CM[4]
- Lucida, a slightly more complete set from Bigelow and Holmes, is now quite extensive
- MathTime,[5] an alternative from Michael Spivak; nevertheless can't be as general as the previous two, since it wasn't designed to complement anything beyond the Times text font (the Times was never seen as the roman version of a family with sans serif, and typewriter versions). The current situation, where combining Times with Helvetica and Courier is seen as 'natural', is more the result of commercial suppliers making this combination available in most word-processing programs and in printers.

From this point on, I will take it as a given[6] that these three font families offer everyone a professional level of quality and consistency of style. The remainder of this article is for adventurous spirits or dissatisfied putterers, especially for those who have become jaded by the over-use of the currently available options. One way to describe the problem we face is "What can I do if I want to use a different font for the text, without spending a lot of time and energy designing the corresponding math symbols?"

2 Typographic limitations

There are three main features or characteristics which limit font combinations: color, style, and proportions. For two fonts to work together inobtrusively, these three traits should be as close as possible. This doesn't mean avoid contrasting fonts — just use contrasts with care. For example, you may want chapter titles to be clearly separate from the text, or have visually obvious heading levels (of course, such a contrast should not be used

[4] As I am primarily interested here in font *design* rather than implementation, I won't spend much time distinguishing fonts from various vendors, or in various formats. For instance, here I don't distinguish between Knuth's CM fonts and Knappen's EC, which is largely based on the former. The slanted CM smallcaps are from the EC font.

[5] Linotype's Mathematical Pi, which has no arrows or italics, is insufficient for use with TEX; we only consider it as a *complement* to MathTime.

[6] This view is shared by Berthold Horn[6], whose article in *TUGboat* provides useful details on TEX math fonts.

CONJECTURE 1.— *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \varnothing$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card}\Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture.— Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

$$x = 2\pi z \Longleftrightarrow \operatorname{card}\Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j\geqslant 0} \left(\sum_{k\geqslant 0} f_{jk} z^k \right) = \sum_{k\geqslant 0} z^n \left(\sum_{\substack{k_0,k_1,\ldots\geqslant 0 \\ k_0+k_1+\cdots=n}} f_{0k_0} f_{1k_1} \cdots \right) \tag{2}$$

(3) Look at the product ffi, $\{\overbrace{g,\ldots,g}^{k\ g},\overbrace{h,\ldots,h}^{\ell\ h}\}$ *taken in the basis* $(\vec{\imath},\vec{\jmath})$.
$$k+\ell \text{ elements}$$

Moreover, eq. (1) yields $$\pm\sqrt{\begin{vmatrix} x_1-x_2 & y_1-y_2 & z_1-z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 2: The same example as Figure 1, done in Lucida.

within paragraphs). We shouldn't forget that combining similar fonts was common practice in printing and publishing. Printers of lead type had far fewer font typefaces available to them than our computers usually provide: a 17pt Caslon in the title combined with Garamond for the text, or bold italic Plantin used with Granjon were perfectly reasonable — if you had no other choice! The first example is a deliberate attempt to startle the reader of today, when vendors pretend that their fonts are infinitely scalable: it's better to use a 17pt font at its design size than to scale the text font up, which is sure to yield something too bold, too round, and with too large an x-height.

2.1 Color

A font that is more or less bold or condensed determines the grayness or 'color' of a page of text. Other parameters — interline space, interword space, margins — all affect color. Keep in mind that the LaTeX default page makeup parameters assume CM fonts. Using another font family may require adjustments to some of these parameters. Grayness is determined

$$\textit{Let } x,\ y,\ z \in \mathbb{Z}\ \textit{; for } f,\ \alpha$$

Figure 3: Text done in Times, math in Computer Modern.

by the white spaces, which are therefore important parameters for typography. Variations in color are often inevitable in math: equations, for example, can change the interline spacing or force large white spaces. At the same time, though, in-line equations should have a minimum effect on the surrounding text.

A typical example would be a math article set with times.sty: since Times is a very "black" font, the material in math mode quite literally gives the impression that there's a hole in the page! FIG. 3 shows this, to a certain extent. Other than the perennial Times, books are often set with less dense fonts, such as Baskerville, Plantin, Minion, or Garamond. Depending on which one is used, these fonts have a color which is slightly darker than CM, while still being lighter than either Times or Lucida. It is

CONJECTURE 1. — *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathscr{A}$, $\mathscr{M}$, $\mathbb{C}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

$$x = 2\pi z \Longleftrightarrow \operatorname{card} \Omega_\alpha \mid \mathscr{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right) \tag{2}$$

(3) Look at the product $fffi$, $\{\overbrace{g, \ldots, g}^{k\ g}, \overbrace{h, \ldots, h}^{\ell\ h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath}\,)$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 4: Again the same example, done in MathTime.

the use of such font families that is at the heart of our problem.

2.2 Style

Font *style* is what I call the design specifics of a font's characters. It's not a quantifiable feature or trait — although, one can consult classifications such as the one by Maximilien Vox to identify fonts of relatively similar styles. Strictly speaking, CM is a "Didone" font, although it has more in common with fonts of the Century/De Vinne type than with Bodoni; considered Transitional Mécane (a hybrid category between Mécane and Didone, which isn't itself in the Vox AtypI classification). That's as far as any classification can help — nothing can replace education and experience. All the same, character design can be a significant obstacle to combining fonts. In particular, if we follow the standard practice of using italics in math and theorems, we run the risk of having two different styles in the same sentence, the proximity causing a rather jarring contrast, an effect which can be heighted since italics are often where design idiosyncracies are most obvious. FIG. 5 shows that it's not a simple problem.

afghkmpwyz, *afghkmpwyz*,
afghkmpwyz, *afghkmpwyz*,
afghkmpwyz,
afghkmpwyz, ***afghkmpwyz***,
afghkmpwyz, ***afghkmpwyz***.

Figure 5: Apollo, Baskerville, Computer Modern, Adobe Garamond, Lucida, Minion, Plantin, Times, and Utopia (all at 21pts).

2.3 Proportions

The last of the three features concerns character proportions. A font style establishes the relationships of the various dimensions of its face: x-height, height of uppercase letters, ascenders, descenders. For POSTSCRIPT fonts, dimensions are specified in

CONJECTURE 1. — *Let x, y, z, be integers; for* $\alpha \in \mathbb{N}$*, denote by* $\Omega_\alpha \subset \mathbb{N}$ *the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*)* $x^p + y^p = z^p$ *admits infinitely many solutions divisible by* α*. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*)*,*
- *more precisely,* $\operatorname{card}\Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

$$x = 2\pi z \Longleftrightarrow \operatorname{card}\Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j\geqslant 0} \left(\sum_{k\geqslant 0} f_{jk} z^k \right) = \sum_{k\geqslant 0} z^n \left(\sum_{\substack{k_0,k_1,\ldots\geqslant 0 \\ k_0+k_1+\cdots=n}} f_{0k_0} f_{1k_1} \cdots \right) \tag{2}$$

(3) Look at the product ffi, $\{\overbrace{g,\ldots,g}^{k\ g},\overbrace{h,\ldots,h}^{\ell\ h}\}$ ($k+\ell$ elements) *taken in the basis* $(\vec{\imath},\vec{\jmath})$.

Moreover, eq. (1) yields $$\pm\sqrt{\begin{vmatrix} x_1-x_2 & y_1-y_2 & z_1-z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 6: Again the same example, done with Mathptm (& dotlessj).

the afm file[7] in terms of 1000pts for each given character, referring to, respectively, the XHeight, CapHeight, Ascender, and Descender.

Each of these parameters can vary independently of the others, as a quick glance through any font catalogue will prove. French printing tradition, going back to Garamont and Granjon, favors what are called 'humanist' characteristics: a fairly small x-height, with uppercase letters below the height of the tallest ascenders, and with generous descenders. In contrast, twentieth-century faces typically have short descenders, ascenders that seem almost atrophied and reduced in size, for what appear to be reasons of efficiency, rationalization of paper savings... In the italic examples in FIG. 5 (all fonts are in the same size), Adobe Garamond and Lucida represent diametrically opposed concepts. Clearly, one shouldn't mix fonts with x-heights that vary too widely, especially in mathematics material, where alignments must default to precise positions (superscripts, for example). You can always bring two

Mixing italic fonts draws attention to their differences.

Figure 7: Garamond and Lucida scaled to the same x-height as Lucida at 20pts.

fonts of different x-heights together by changing the scale but the results can be unpleasant if their respective proportions are too divergent. The example in FIG. 7 demonstrates yet another factor: the slope of the italic characters (ItalicAngle).[8]

[7] A good qualitative description of what should be expected from the metrics of a font is provided in [10]. For a more technical approach, see [1].

[8] TEX is satisfied with slightly less specific information, which is stored in the tfm file, for its seven \fontdimen values: the value of an em (the size of a given font, implicit in the afm), the value of an ex (XHeight), and the tangent of ItalicAngle. The remaining dimensions concern spacing, whereas a POSTSCRIPT afm file specifies the width of the space character, but does not control the elasticity of an interword space.

CONJECTURE 1.— *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant*.

Evidence for the conjecture.— Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

(1) $$x = 2\pi z \Longleftrightarrow \operatorname{card} \Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

(2) $$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product ffi, $\{\overbrace{g, \ldots, g}^{k\ g}, \overbrace{h, \ldots, h}^{\ell\ h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath})$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 8: Text in Apollo, math in Computer Modern.

3 Customizing a suitable math font

In light of these negative aspects to the problem, we will discuss two methods which each provide a "solution". The examples have been tested, in that they provide a reasonable level of quality in documents containing mathematics. However, neither can pretend to address either the reliability or the quality of the three math font families discussed in our first section. Keeping in mind the remarks attached to its presentation, I would be willing to print a book using the second method (§ 3.3), but I'd only make photocopies if the first method (§ 3.1) had been used. In order of increasing difficulty in implementation, we'll start with Alan Jeffrey's mathptm option, then we'll examine how simple NFSS commands or a virtual font created via fontinst allows us to choose, character by character, each font used within a math environment.

3.1 Mathptm

The mathptm distribution includes virtual fonts created by using fontinst, as well as the style option mathptm.sty, which makes it possible to use the glyphs of Times in math mode with LaTeX. The font is a marvel in that it manages to simulate the majority of the 384 glyphs found in the three math font families, by accessing the Times and Symbol fonts available on any PostScript printer (the calligraphic uppercase letters, accessed via the \mathcal command, come out in Zapf Chancery); as a last resort, some characters are taken from Computer Modern. The style option modifies the LaTeX defaults by invoking these various math font families, adjusting spacing parameters in math mode, and modifying the size of the type body for first- and second-order exponents. This last operation is interesting, because it pushes the 'standard' PostScript fonts to their limits for typesetting mathematics. At 10pts, (La)TeX uses fonts at point sizes 10, 7, and 5, for normal text, super- and sub-scripts, and second-order super- and sub-scripts, respectively. Each of these sizes corresponds to a distinct font in the Knuth distribution, since it's necessary to make optical corrections in a 5pt font so that it's readable. PostScript printers have only one font (designed at a 12pt size) for each variant in the Times family. This means the only way to get a

5pt Times font is by applying a scaling but without optical correction, which in turn means the characters are difficult to read. Mathptm.sty redefines these sizes to 10, 7.4 and 6pts, which reduces — but does not eliminate — the visual problems. Mathematical Pi, Lucida, and MathTime will all show this flaw, hence the user will always have to adapt type body sizes with reference to readability. A few of the PostScript Multiple-Master fonts address the optical scaling issue, and while support for use with TEX is a bit tentative, I am convinced that within a few years, expert sets for these fonts will include all the refinements one could wish for.[9]

Mathptm is a free alternative to the MathTime fonts, but there are some drawbacks to it. The most obvious is that the Symbol font, which may be adequate for showing the characters available in the more popular word-processing programs, is decidedly smaller than the needs and possibilities available with TEX. For example, cmex has large expandable delimiters (the ones accessed via \big or \left) whereas Symbol only has the regular parentheses, and the elements needed to create large parentheses; other expanded characters are simply scaled versions of the Symbol character. The other problem is that the lowercase Greek characters are upright. Now, almost all letters inside math mode are presented in italics: upright lowercase Greek letters may require italic corrections, which is fairly bizarre. Generating a slanted version of Symbol (using the SlantFont operation in dvips for example), might work, but the result wouldn't be very good, especially if the slant was pushed to the values usually assigned to true italic fonts (roughly 15° vs. less than 10 for slanted fonts, to get something one could call 'acceptable'). Moreover, this would introduce yet another slope in math equations, which should be avoided as much as possible.

In summary, then, mathptm doesn't really offer a solution to the problem as outlined initially, but it does contain the kernel of two possible approaches to it: (1) a style option relying on NFSS commands to modify math fonts, and (2) creating (via fontinst) virtual math fonts which address specific requirements. As I study the mathptm virtual fonts, I am convinced that there is no other satisfactory alternative to symbol and extension fonts: for such fonts, the issues of style, color, and proportion are not present, so the point is to ensure that their design is consistent and of good quality. My own approach is to use members of one of the three basic families in terms of what works best — using the serifs of \prod as a guide, for example. The Computer Modern versions are adequate for the majority of cases I've run into. The problem of uniformity of typographic characteristics is crucial for alphabets (letters, in other words) such as the Roman and Greek italic letters in the math italics fonts, and then on to the uppercase calligraphic letters, located in the symbol fonts, and uppercase Greek letters, which should appear in OT1-encoded text fonts (such as cmr). For example, for a professional-looking text, one might prefer to use \mathcal to access the uppercase cursive characters in the rsfs font or the Commercial Script font (as shown in the examples on figures 11–13). It seems obvious to me that the preference will always be to choose the italic version of the text font for use in math mode. Below are two methods of achieving that goal.

3.2 Mathfont

I call mathfont.sty a 'generic' extension to access the necessary glyphs in math mode (its main features are discussed here, while the details are left for the reader to study).[10] The essentials are covered in the *LATEX Companion*. LATEX (essentially NFSS) introduces two concepts for fonts in math mode: *alphabets* and *symbols*. An alphabet is explicitly invoked by commands such as \mathbf. Assuming one has a text font, the math version of \mathbf can always be defined by means of a declaration such as:

```
\def\ED{\encodingdefault}% shorter!
\DeclareMathAlphabet{\mathbf}{\ED}%
  {\rmdefault}{b}{n}.
```

In this fashion, you can redefine \mathcal to access the ornamented letters one prefers. Additionally, if you want alphabets defined in this way to respond to the \boldmath command (to put mathematics material into boldface), you could do the following:

```
\DeclareMathAlphabet{\mathsf}{\ED}%
  {\sfdefault}{m}{n}
\SetMathAlphabet{\mathbf}{bold}{\ED}%
  {\rmdefault}{b}{n}
\SetMathAlphabet{\mathsf}{bold}{\ED}%
  {\sfdefault}{b}{n}.
```

We're more interested in symbols, but note that \mathversion already exists (\boldmath is the same as saying \mathversion{bold}). All fonts used in math mode can have a different version, depending on what is specified by \mathversion: for example, a textmathitalics or textmathupright version could be

[9] Provided someone feels the urge to produce the missing mathematical symbols ...

[10] It should be noted that a ready-to-use minimal adaptation of math italic for text italic can be found at: ftp://fourier.ujf-grenoble.fr/pub/contrib-tex.

CONJECTURE 1. — *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

(1) $$x = 2\pi z \Longleftrightarrow \operatorname{card} \Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

(2) $$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product ffi, $\{\overbrace{g, \ldots, g}^{k\ g}, \overbrace{h, \ldots, h}^{\ell\ h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath})$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 9: Text in Utopia, math in Computer Modern.

defined so that math italics would be accessed in a font invoked after \mathversion{textmathitalics}. This means it's possible to have several versions coexisting in the same document, just as it's possible to have several encodings for the text material. However, you have to be careful of TEX's limitations in this area: \mathversion can only be changed outside math mode, but never within an equation. Fonts invoked via this method must therefore be acceptable for such usage: if you've created a textmathupright version which replaces math italics by upright characters, these latter must be in a OML-encoded font in order to access such characters as lowercase Greek.

The following declarations introduce the four default symbol fonts:

```
\DeclareSymbolFont{operators}%
  {OT1}{cmr}{m}{n}
\DeclareSymbolFont{letters}%
  {OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}%
  {OMS}{cmsy}{m}{n}
\DeclareSymbolFont{largesymbols}%
  {OMX}{cmex}{m}{n}
\SetSymbolFont{operators}{bold}%
  {OT1}{cmr}{bx}{n}
\SetSymbolFont{letters}{bold}%
  {OML}{cmm}{b}{it}
\SetSymbolFont{symbols}{bold}%
  {OMS}{cmsy}{b}{n}
\DeclareSymbolFontAlphabet{\mathrm}%
  {operators}
\DeclareSymbolFontAlphabet{\mathnormal}%
  {letters}
\DeclareSymbolFontAlphabet{\mathcal}%
  {symbols}.
```

To obtain the results we want, we select the most suitable symbols font (or largesymbols) font which works the best. What concerns us here are the operators and letters. By default, LATEX uses the cmr operators font for:

1. digits 0–9
2. small delimiters (parentheses, brackets, etc.)
3. punctuation, including ; :
4. uppercase Greek letters
5. most accents
6. the + = signs

CONJECTURE 1. — *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

(1) $$x = 2\pi z \Longleftrightarrow \operatorname{card} \Omega_\alpha \mid \mathcal{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

(2) $$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product *ffi*, $\{\overbrace{g, \ldots, g}^{k\ g}, \overbrace{h, \ldots, h}^{\ell\ h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath})$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 10: Text in Utopia, with the mathfont option.

This extensive use of text characters in math mode is one of TEX's pitfalls (font changes are therefore very risky, particularly in plain).[11] While it may be natural to use the digits from the default text font, it's not likely that uppercase Greek letters will be found there. Parentheses and the + and = signs warrant a brief detour. Parentheses should be consistent with their larger versions, and thus should come from the text font and matching extension font, all within the same font family (just as cmr and cmex are part of the CM family). The = sign is rather critical in that it joins the combinations $\Leftarrow$ and $\Rightarrow$ to produce $\Longleftrightarrow$. Thus, it's really part of math characters; unfortunate that it's not part of a specific math font (the $-$ sign has the same function in simple arrows such as $\longleftrightarrow$ even though it's part of the symbols font).

What mathfont does is define a second set of operators, called *textoperators*, and then it tells LATEX to take the digits and accents from there (which is a bit risky if you're accenting letters in math mode that aren't from the same font ...). This yields:

```
\DeclareSymbolFont{textoperators}   {\ED}%
  {\rmdefault}{m}{n}
\SetSymbolFont{textoperators}{normal}{\ED}
  {\rmdefault}{m}{n}
\SetSymbolFont{textoperators}{bold} {\ED}%
  {\rmdefault}{b}{n}
\DeclareMathSymbol{0}{\mathalpha}%
    {textoperators}{`0}
  (...)
\DeclareMathSymbol{;}{\mathpunct}%
    {textoperators}{"3B}
  (...)
% Attention: only in OT1 encoding
\DeclareMathAccent{\hat}{\mathalpha}%
    {textoperators}{"5E}
  (...)
```

Thus, using fontmath.ltx as a guide, it's possible to create a new symbols font, selecting the characters that will be in it.

[11] This can only be addressed by the development of new font encodings, clearly differentiating text fonts (T1, for example) from text symbol complements (as in TS1) and math symbols (MC, MSP, currently being worked on by a TEX Users Group Technical Working Group).

The math italic font can be copied in the same way. By default, LaTeX uses the letters font (cmmi) for the following:

1. regular letters (without accents)
2. a few punctuation signs, such as , .
3. the italic Greek letters
4. some letter-type symbols that are very useful, such as \imath ($\imath$), \jmath ($\jmath$), \ell (ℓ), \partial (∂), some of the "harpoons" (e.g., \leftharpoonup ($\leftharpoonup$)
5. some of the relatively useless symbols, such as \smile ($\smile$);
6. the only 'accent' that's not in a text font: \vec ($\vec{}$)

If the selected font is a standard PostScript font, it will only include letters and punctuation signs — the rest have to be found elsewhere. For example, in mathfont.sty:

```
\DeclareSymbolFont{textletters}{\ED}%
  {\rmdefault}{m}{it}
\SetSymbolFont{textletters}{normal}{\ED}%
  {\rmdefault}{m}{it}
\SetSymbolFont{textletters}{bold}{\ED}%
  {\rmdefault}{b}{it}
\DeclareMathSymbol{a}{\mathalpha}%
    {textletters}{`a}
  (...)
\DeclareMathSymbol{A}{\mathalpha}%
    {textletters}{`A}
  (...)
\DeclareMathSymbol{,}{\mathpunct}%
    {textletters}{"3B}.
```

A word on the specific case of \imath and \jmath: the first is standard in PostScript fonts (under the name *dotlessi*), whereas the second is absent.[12] Using the base ($\vec{\imath}, \vec{\jmath}$) as a reference, it's clear that you can't use characters that are too different. As well, a word of warning about my decision to use a text font family *with its default encoding* — the choice was made purely as a way of limiting the amount of memory TeX would allocate to the font metrics. Unlike the other characters modified up to this point in the article, $\imath$ without a dot does not occupy the same slot in T1 or OT1 encodings. This method raises two additional problems:

- the number of font families declared at the same time is limited to sixteen. Each new declaration takes up one of these slots, so the method is not economic and carries certain risks.

- Math fonts and text fonts do not adhere to the same imperatives. We have to keep in mind that our initial problem revolves around the aesthetic impressions some glyphs have over others, whereas TeX doesn't really care about glyphs, just their 'metrics', via the tfm file. In math mode, each character is an atom, which must be placed relative to other such characters, according to its type (relation, delimiter, etc.). This is why \fontdimens 2, 3, 4 and 7 have a zero value in cmmi.[13] Although one can modify these global parameters dynamically from the (LA)TeX source (in the .fd file, for example), they would be attached to the font being loaded once only, which means it's not possible to call up the same font twice, using two different names, and assigning each one different parameters. Thus, to preserve the normal italics for text, the mathfont option produces atypical italics for math. The \fontdimen issue isn't too troubling since TeX suppresses spaces in math mode; at the same time, though, it's not possible to suppress kerning or ligatures between letters, which can lead to some odd results for something like Te or ffi: *Te ffi*. Similarly, TeX assumes that the side bearings (the lateral space which a designer adds to ensure that two characters of the same font don't touch) are as generous as those of its default fonts, which isn't really the general case. Super- and subscripts can end up looking like they're touching. If you use mathfont, you have to keep these points in mind: don't hesitate to include explicit kerning instructions (\mkern) to avoid inopportune ligatures and adjust the spacing.

The only way to get a math font, in terms of glyphs, similar to the one I've tried to obtain with mathfont (but uniform in terms of its metrics and independent of coding hazards) is to create a virtual font by using the same scheme, but making it more solid — and thus less flexible.

[12] The 'successful' examples presented here demonstrate three reasonable alternatives for *dotlessj*. (1) Since Utopia's $\imath$ is fairly similar to the $\jmath$ in Lucida, I used these two glyphs (of different origins) in Fig. 11. (2) This ruse is not possible for Apollo, so I simply edited the Apollo font and created a new character, copying j and removing the dot. (3) For Fig. 13, I was able to directly parameterize the PostScript fonts, using a *header* that Bernard Desruisseaux graciously provided, and thus magically removed the dot from the j and made it the same height as a virtual $\jmath$.

[13] Respectively, these \fontdimen establish the space to put between words, the maximum space to add or take away, and the special spaces after punctuation (as used in Anglo-Saxon typography).

CONJECTURE 1. — *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \varnothing$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathscr{A}, \mathscr{M}, \mathscr{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

(1) $$x = 2\pi z \iff \operatorname{card} \Omega_\alpha \mid \mathscr{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

(2) $$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product $f\,f\,i$, $\{\overbrace{g, \ldots, g}^{k\,g}, \overbrace{h, \ldots, h}^{\ell\,h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath})$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 11: Text in Utopia, math fonts based on Lucida and Computer Modern.

3.3 Virtual fonts

Creating a virtual font with fontinst [7] (see [5] for many concrete examples similar to the ones presented here) essentially comes down to understanding the \installfont command. This generates a vpl file, which is a human-readable equivalent of the virtual font (vf). The following shows how the two fonts we're using from the mathptm distribution are created:

```
\installfamily{OT1}{ptmcm}{}
\transformfont{ptmr8r}{\reencodefont{8r}%
                      {\fromafm{ptmr8a}}}
\installfont{zptmcmr}%
  {ptmr8r,psyr,latin,zrhax,kernoff,cmr10}%
  {OT1}{OT1}{ptmcm}{m}{n}{}

\installfamily{OML}{ptmcm}%
          {\skewchar\font=127}
\transformfont{ptmri8r}{\reencodefont{8r}%
                       {\fromafm{ptmri8a}}}
\installfont{zptmcmrm}
  {kernoff,cmmi10,kernon,unsetalf,%
   unsethum,ptmri8r,psyr,mathit,%
   zrmhax}%
  {OML}{OML}{ptmcm}{m}{it}{}.
```

As you can see, the \installfont command has eight arguments:

- the first is the name (for the tfm and vf files for the font being generated)
- the second argument contains the set of file names (with extension .mtx) needed for creating *metrics* for each character
- the third indicates the internal coding used by fontinst for the font in question
- the next four arguments specify the parameters which allow LaTeX to identify the font via the fd file, created by the \installfamily command
- the last argument makes it possible to configure the declaration contained in the fd file. This argument can be very useful for installing several virtual fonts that address optical scaling.

It's now possible to see that mathptm will install its operators font (replacing cmr in math) by using characters taken from ptmr8r, psyr and cmr10 (that is, Times Roman re-encoded as 8r for all the glyphs, Symbol, and Computer Modern roman). The order of these is important because all the glyphs required

CONJECTURE 1.— *Let x, y, z, be integers; for $\alpha \in \mathbb{N}$, denote by $\Omega_\alpha \subset \mathbb{N}$ the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*) $x^p + y^p = z^p$ admits infinitely many solutions divisible by α. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card} \Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant*.

Evidence for the conjecture.— Denoted by $\mathscr{A}$, $\mathscr{M}$, $\mathscr{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

$$x = 2\pi z \Longleftrightarrow \operatorname{card} \Omega_\alpha \mid \mathscr{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx \tag{1}$$

$$\prod_{j \geqslant 0} \left(\sum_{k \geqslant 0} f_{jk} z^k \right) = \sum_{k \geqslant 0} z^n \left(\sum_{\substack{k_0, k_1, \ldots \geqslant 0 \\ k_0 + k_1 + \cdots = n}} f_{0k_0} f_{1k_1} \cdots \right) \tag{2}$$

(3) Look at the product ffi, $\{\overbrace{g, \ldots, g}^{k\ g}, \overbrace{h, \ldots, h}^{\ell\ h}\}$ (*$k+\ell$ elements*) *taken in the basis* $(\vec{\imath}, \vec{\jmath})$.

Moreover, eq. (1) yields $$\pm \sqrt{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 12: Text in Apollo, math fonts based on Computer Modern.

by an OT1 font are present in cmr10, yet fontinst follows the order in which it finds the glyphs needed for the encoding: letters are thus acquired from Times or, if they aren't there, they have to be 'simulated', thanks to macros in the file latin.mtx; the Greek uppercase letters come from the Symbol font. However, as mentioned previously, it can be risky using some Times glyphs, such as `( ) [ ] + =`, so the zrhax.mtx file removes them from fontinst's memory so that they're selected from cmr10 instead. The kernoff.mtx file in turn suppresses the kerning that comes from cmr10; the resultant zptmcmrm font which is thus created will therefore be an *ersatz* font, providing symbols usable as operators without risk. The characters are accessed by the following declaration (which appears in mathptm.sty):

```
\DeclareSymbolFont{operators}%
  {OT1}{ptmcm}{m}{n}
```

The font zptmcmrm which replaces cmmi is obtained in the same way: you take all the glyphs from cmmi10 (but leave their kerning behind), the unsetalf and unsethum files remove the lowercase Greek letters, and the italics provided by Symbol and Times Italic (respectively), mathit plays the role of latin for the OML fonts, and zrmhax adjusts certain spacing parameters which would not be acceptable if this font were nothing but a regrouping of characters from different sources. For the same reasons as we saw with mathfont, the side-bearings, which are distinctly more restricted in Times than in Computer Modern, are enlarged; accent positions in math mode, which are controlled through a special mechanism: pseudo kern pairs, with the so-called \skewchar, are enhanced. Thanks to fontinst, it's possible to correct all the shortcomings of the mathfont option, i.e., by specifically using glyphs chosen for aesthetic reasons (but arranged in the standard LaTeX encoding), and by adjusting all the metric parameters (kerning, side-bearings, etc.), so they can be made to work optimally in mathematics. The only thing missing from these examples is the command *scaled*, which makes it possible to adjust to the same value the x-heights of the various fonts mixed into a single font. Another advantage not to be discounted with the 'virtual font' solution: since it yields fonts which can replace the 'original versions' of Computer Modern, it is also very easy to

use with *all* TEX dialects or formats (even *plain*). On the other hand, one could say that adjusting metric parameters is a subtle business, and should be left to a true typographer ...

The examples I've presented throughout this article were produced with a certain degree of haste — they are far from being optimal. It's not really feasible for me to distribute the virtual fonts I've been describing, because most are just the result of combining various bits of commercial fonts — which is why a finished package is not available. On the other hand, I will try to share the skills I've acquired with difficulty. I don't believe in a set of macros that can systematically generate virtual math fonts for, say, Palatino, Times, and New Century Schoolbook. These fonts are too different: it's impossible for any given symbol not to clash with any one of them. As well, there are problems adjusting the x-heights and side-bearings that simply can't be dealt with in a generic way.

I'll finish off this presentation with a concrete demonstration (used in *plain* TEX by the secretaries at the Fourier Institute). The text font is T1 Utopia Expert scaled down to the x-height of cmr10. Utopia, which has a dark color to it, doesn't really work with cmmi, although the symbols in cmsy/cmex don't clash once they've been scaled down. After a few attempts, I finally chose Lucida for the upper- and lowercase Greek letters, Utopia Italic for math italics, and Utopia Expert for *oldstyle* digits. This yields the following:

```
\installfamily{OML}{putluc}%
  {\skewchar\font=127}
\transformfont{putri8r}{\reencodefont{8r}%
                       {\fromafm{putri8a}}}
\installfont{zputlucm}%
  {kernoff,hlcrim scaled 804,kernon,%
   unsetalmf,unsetos,putri8r scaled 880,%
   putr8x scaled 880,utmathit,zrmuthax}%
  {OML}{OML}{putluc}{m}{it}{}
\installfamily{OT1}{putluc}{}
\transformfont{putr8r}{\reencodefont{8r}%
                      {\fromafm{putr8a}}}
\transformfont{hlcr7t}{\reencodefont
                              {OT1luc}%
                      {\fromafm{hlcr8a}}}
\installfont{zputluc7t}
  {putr8r scaled 880,putr8x scaled 880,%
   hlcr7t scaled 840,latin,zrhax,%
   kernoff,cmr10}%
  {OT1}{OT1}{putluc}{m}{n}{}
```

You can see that I've modified a few mathptm files, and have introduced a new unsetos to suppress the oldstyle digits, so that they come from the expert font rather than from cmmi.[14] One final example with Apollo. Since this is a lighter face, I decided to plunder Computer Modern for all the math symbols that don't exist in Apollo.

```
\installfamily{OML}{mapcm}%
  {\skewchar\font=127}
\transformfont{mapri8r}{\reencodefont{8r}%
                       {\fromafm{mapri8a}}}
\installfont{zmapcmm}
  {kernoff,cmmi10,kernon,unsetalmf,%
   unsetos,mapri8r scaled 1067,%
   mapr8x scaled 1067,apmathit,zrmaphax}%
  {OML}{OML}{mapcm}{m}{it}{}
\installfamily{OT1}{mapcm}{}
\transformfont{mapr8r}{\reencodefont{8r}%
                      {\fromafm{mapr8a}}}
\installfont{zmapcm7t}
  {mapr8r scaled 1067,mapr8x scaled 1067,%
   cmlatin,zrhax,kernoff,cmr10}%
  {OT1}{OT1}{mapcm}{m}{n}{}
```

4 Conclusion

I've illustrated different possible solutions in the above examples for Utopia and Apollo.[15] The principle behind these various illustrations has been the following: maintain the identical text each time, and change only the preamble, which takes care of modifying the fonts to be used via 'standard' NFSS (the equivalent of times.sty; see FIG. 8, 9), macros such as mathfont (FIG. 10), and ending up with composite virtual fonts, as described above (FIG. 11, 12). While the defects in mathfont are obvious enough (poor spacing around parentheses, the *ffi* ligature problem), you have to keep in mind that they can be fixed manually, which is a do-able operation in a document without a lot of math formulae and typeset by someone who knows what they're doing. After demonstrating the three comprehensive font systems available — Computer Modern (FIG. 1), Lucida (FIG. 2) and MathTime (FIG. 4), plus Mathptm (FIG. 6) — I have shown what you can get, starting from two text fonts of incompatible design with math characters from either Computer Modern or Lucida.

The Apollo example, although somewhat marginal (I don't see it used that often), does show the benefits of the approach I use. Its style is wildly

[14] This manoeuver doesn't really have any bearing on LATEX but it does allow the plain TEX \oldstyle command to work.

[15] A related discussion can be found in [2]. The task there was to modify not only the typography but also the layout of a LATEX book.

CONJECTURE 1. — *Let x, y, z, be integers; for* $\alpha \in \mathbb{N}$*, denote by* $\Omega_\alpha \subset \mathbb{N}$ *the set of prime integers p (called p-primes in the sequel) such that the following equation (known as* Frimas' last equation*)* $x^p + y^p = z^p$ *admits infinitely many solutions divisible by* α*. We conjecture:*

- $\Omega_\alpha \neq \emptyset$ (Ω_α *is* not *empty*),
- *more precisely,* $\operatorname{card}\Omega_\alpha > \mathsf{w}$ *where* w *is the well-known* WHYLLES' *constant.*

Evidence for the conjecture. — Denoted by $\mathscr{A}, \mathscr{M}, \mathscr{O}$, the famous *inferior constants of* WHYLLES, the three following formulae are *very* instructive:

(1) $$x = 2\pi z \Longleftrightarrow \operatorname{card}\Omega_\alpha \mid \mathscr{M} \quad \text{and} \quad \varphi(t) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-x^2/2} dx$$

(2) $$\prod_{j\geqslant 0} \left(\sum_{k\geqslant 0} f_{jk} z^k \right) = \sum_{k\geqslant 0} z^n \left(\sum_{\substack{k_0,k_1,\ldots\geqslant 0 \\ k_0+k_1+\cdots=n}} f_{0k_0} f_{1k_1} \cdots \right)$$

(3) Look at the product ffi, $\{\overbrace{g,\ldots,g}^{k\ g},\overbrace{h,\ldots,h}^{\ell\ h}\}$ ($k+\ell$ elements) *taken in the basis* $(\vec{\imath},\vec{\jmath})$.

Moreover, eq. (1) yields $$\pm\sqrt{\begin{vmatrix} x_1-x_2 & y_1-y_2 & z_1-z_2 \\ l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \end{vmatrix}} > 0.$$

Figure 13: Text in Minion MM, math fonts based on Computer Modern.

incompatible with Computer Modern yet its proportions and, above all, its color, are quite similar: once you remove the style incompatibility between them by using text italics in math, you get an undeniable uniformity and quality. Here's a list of fonts often used in books, which seem to me to lend themselves, without too much damage, to the games I've been playing with Apollo: Bembo, Adobe Garamond, Garamond Three, Granjon, Plantin *Light*, Times *Light*. Also possible, but probably without the same degree of uniformity, are Adobe Caslon, Galliard, or Baskerville. To complement Palatino, Melior, Stempel Schneidler, New Century Schoolbook, I'd think of Lucida. While Stone or Rotis could prefer MathTime symbols.

To conclude on a more pessimistic note: the French version of this article [3] was typeset in Minion — for me, one of the most beautiful fonts currently available, remarkably readable and elegant at the same time.[16] Today, I would choose it without hesitation for a good-quality journal. Unfortunately, the Minion design displays its acknowledgement of the Italian and French Renaissance too clearly. The initial version of this article had been prepared with the *Single Master* version (used by the journal *Libération*), which gave the page a relatively dark color, but not as dark as either Times or Lucida. And for this reason, none of the three basic fonts can complete it, even though MathTime is probably the least problematic. Just as this article was being finished, I installed the *Multiple Master* version of Minion, which makes it possible to incrementally vary the thickness, the width, and the optical size yet still maintain a consistent design. As we've seen, this last property is crucial for the readability of smaller point sizes (superscripts, for example), and it's one of this font's undeniable advantages.

I've tried to experiment with the thinnest and widest instances so that color and proportion converged as much as possible with those of Computer Modern.[17] It's interesting to note that Hilmar Schlegel reports getting quite satisfactory results by

[16] They say that Minion's on its way to becoming the 'Times of the 21st century', which is why I'm in a hurry to use it now before it becomes too passé!

[17] The complete interface for production of the French version of this article will eventually become available on CTAN, as an example. A pre-version is already somewhere on my home site: see `ftp://fourier.ujf-grenoble.fr/pub/contrib-tex/psfonts/adobe`.

using a similar method, but with a combination of a fairly bold and slightly narrowed Minion face with MathTime. FIG. 13 shows how this "works" quite respectably under 'real' conditions. Nevertheless, one can see that each glyph from the Computer Modern family is a surprise to the eye, and that there really is no alternative to it, at least regarding Greek letters.

Thanks — I went into this article without any idea where it would all end. Since I'm neither a programmer nor a typographer, nor a (LA)TEX guru (much less one in POSTSCRIPT), a certain number of unexpected roadblocks came up along the way. I'd like to thank everyone who helped me over these hurdles. In particular, I'd like to mention Jacques André, Bernard Desruisseaux and Hilmar Schlegel for their constructive criticisms and technical help, which made it possible for me to write this paper. Last but not least, it's a pleasure to thank Christina Thiele who undertook the present translation with patience & skill.

References

[1] Jacques André, Font metrics, *Visual and Technical Aspects of Types* (Roger D. Hersch, ed.), Cambridge University Press, 1993, pp. 64–77.

[2] Thierry Bouche. *Approximation of one of Henri CARTAN's books: first try*, `http://www.loria.fr/tex/fontes/maths/cartan-english.html`. The LATEX Navigator, Nancy, France, 1995.

[3] Thierry Bouche. *Sur la diversité des fontes mathématiques*, *Cahiers GUTenberg* **25** (1–24) novembre 1996.

[4] Michel Goossens, Frank Mittelbach, & Alexander Samarin. *The LATEX Companion.* Addison-Wesley, Reading, USA, 1994.

[5] Aloysius G. Helminck. *Contributions to fontinst*, on CTAN. 1994.

[6] Berthold K.P. Horn. *Where Are The Maths Fonts ?*, TUGBOAT Vol. 14 **3**, 282–284, 1993.

[7] Alan Jeffrey. *The fontinst package*, documentation accompagnying the software distribution (the paper in TUGBOAT 14/3 is obsolate). June 1994.

[8] Alan Jeffrey. POSTSCRIPT *Fonts in LATEX 2ε*, TUGBOAT Vol. 15 **3**, 263–268, 1994.

[9] Robert Bringhurst. *The Elements of Typographic Style.* 2nd ed. Hartley & Marks, Vancouver. 1996

[10] Lewis Blackwell. *Twentieth Century Type.* Calman & King, London. 1992

Hints and Tricks

'Hey — it works!'

Jeremy Gibbons

Welcome to *'Hey — it works!'*, a column devoted to (LA)TEX tips, tricks and techniques. In this issue, we have an article by Robert Tolksdorf, on automatically inserting or avoiding spaces after macros that expand to text (such as the macro `\TUB`, which generates '*TUGboat*'); this is based on a macro by Donald Arseneau in an earlier column in *TTN*. We also have an article by Pedro Aphalo on generating dashed lines of various kinds in LATEX.

I have decided to expand the scope of the column to include also METAFONT and METAPOST techniques, prompted by a recent question on the METAFONT mailing list. To get the ball rolling, this issue concludes with an article of mine on drawing double-headed arrows in METAPOST. Please send me any more little METAFONT or METAPOST snippets you might have, along with the usual TEX and LATEX ones.

◇ Jeremy Gibbons
CMS, Oxford Brookes University
Gipsy Lane, Headington
Oxford OX3 0BP, UK
jgibbons@brookes.ac.uk
http://www.brookes.ac.uk/~p0071749/

1 Smart spaced macros everywhere

The article *Italic correction everywhere* by Donald Arseneau in *TTN* 3,1:15 addresses the issue of inserting an italic correction automatically, if there is no punctuation following the italicized text.

A similar problem is the generation of spaces after a macro that generates text, such as the `\TUB` macro from the *TUGboat* document class. Consider the sentence "*TUGboat* uses the macro `\TUB` to generate '*TUGboat*'." The source for this sentence reads:

```
\TUB\ uses the macro \verb"\TUB"
to generate `\TUB'.
```

What one would like to avoid is the manually inserted `\␣` after the first `\TUB`. The following lines introduce the macro `\smartspace` that automagically inserts it when no punctuation follows the macro:

```
% smart insertion of space
% Robert Tolksdorf (tolk@cs.tu-berlin.de)
% Following Donald Arseneau,
% Italic correction everywhere, TTN 3,1
```

```
\def\smartspace#1{{\protect
  \aftergroup\smartspaceit#1}}
\def\smartspaceit{\futurelet\spta\sptest}
\def\sptest{\ifcat\noexpand\spta,\else\ \fi}
```

Now, we can define a macro `\TUGboat` by

```
\def\TUGboat{\smartspace{\TUB}}
```

and use

```
\TUGboat uses the macro \verb"\TUGboat"
to generate `\TUGboat'.
```

And hey, to quote Donald Arseneau, this works for $99\,^{44}/_{100}\%$ of the time only, as '`\TUGboat --`' shows. Someone tell me why![1]

◇ Robert Tolksdorf
Technische Universität Berlin
tolk@cs.tu-berlin.de

2 Dashed lines

Sometimes, for example when including data plots, it is necessary to include in the caption to a figure different dashed or entire line segments used to identify different lines, like this:

A (┈┈┈┈), B (— — —)

The size and location of these line segments should match the surrounding text. After reading Norbert Schwarz's 'Introduction to TEX' book, I wrote a very small package which I have been using for some time with LATEX. It is based on a command which can generate most commonly used dashed lines.

```
\def\dashedrule#1#2#3{{%
  % #1 is length of dash
  % #2 is length of gap between dashes
  % #3 is number of dashes

  \dimen1=#2 \divide\dimen1 by 2

  \def\@ruledash{%
    \rule{\dimen1}{0pt}%
    \rule[0.5ex]{#1}{0.4pt}%
    % line is 0.5ex above the baseline
    % and 0.4pt thick
    \rule{\dimen1}{0pt}}%

  \count1=0
  \loop%
  \ifnum\count1<#3%
    \advance\count1 by 1%
    \@ruledash%
  \repeat}}
```

How does it work? `\@ruledash` draws a single dash plus half a gap in front of it, and half a gap after it. A loop draws as many dashes, surrounded by half gaps, as indicated by the third argument. Using this command it is extremely easy to define different dashed line segments of equal length, as the example below shows for 3em-long line segments.

```
% length of line segment is (#1 + #2) * #3
\def\longdashes{\dashedrule{.8em}{.2em}{3}}
\def\mediumdashes{\dashedrule{.3em}{.2em}{6}}
\def\shortdashes{\dashedrule{.1em}{.1em}{15}}
\def\solidline{\dashedrule{3em}{0em}{1}}
\def\sparsedashes{\dashedrule{.5em}{.5em}{3}}
```

‘— — —’
‘------’
‘┈┈┈┈┈┈’
‘_____’
‘– – –’

◇ Pedro J. Aphalo
Faculty of Forestry
University of Joensuu
pedro.aphalo@joensuu.fi
http://cc.joensuu.fi/~aphalo/

3 Double-headed arrows

A recent request on the METAFONT mailing list was for help in drawing double-headed arrows. One correspondent provided the following definition of a macro to draw a path with an arrowhead at each end:

```
def draw_dbl_arrow text t =
  path p, q;
  p := t;
  q := subpath (0,.5) of p;
  drawarrow reverse q;
  q := subpath (.5,1) of p;
  drawarrow q
enddef;
```

For example,

```
draw_dbl_arrow (0,0){right} .. {right}(50,25);
```

produces:

This definition can be improved in several ways. For one thing, there is no need to use assignments like this. METAPOST[2] has a powerful expression language, and in particular you can use an expression as the argument to `drawarrow`:

```
def draw_dbl_arrow text t =
  drawarrow reverse (subpath (0,.5) of p);
  drawarrow subpath (.5,1) of p
enddef;
```

[1] Note that if you make the macro `\TeX` smart-spaced, then '`\TeX book`' no longer works as it used to!

[2] Although we use the term 'METAPOST' to refer to the language, everything in this article applies equally to the METAPOST and METAFONT systems.

For another thing, that '1' should be '`length p`', otherwise the macro will only work for a path of length 1.

Indeed, there is no need to draw just subpaths of `p`; there is no harm in drawing `p` itself twice:

```
def draw_dbl_arrow text t =
  drawarrow reverse p;
  drawarrow p
enddef;
```

In fact, the original poster asked for a triple-headed arrow, with two arrow heads at (for the sake of argument) the end of the path. If you can pick the right small value of `e`, you can achieve this by just drawing the path three times:

```
def draw_trp_arrow text t =
  drawarrow reverse p;
  drawarrow p;
  drawarrow subpath (0, length p - e) of p
enddef;
```

But what value to pick for `e`? You could just experiment, but different values will be needed for different paths to get consistent results. A better approach is to find the time at which a point traversing path `p` is a certain fixed distance (namely, the length of an arrow head, `ahlength`) from the end of `p`, and to draw the corresponding subpath of `p`. You can find that time using `intersectiontimes`, intersecting `p` with a circle of the appropriate size centred on the end of `p`.

```
def draw_trp_arrow text t =
  drawarrow reverse p;
  drawarrow p;
  path q;
  q := fullcircle scaled (2*ahlength)
       shifted (point (length p) of p);
  numeric tp, tq;
  (tp,tq) = p intersectiontimes q;
  drawarrow subpath (0, tp) of p
enddef;
```

On the same path as before, `draw_trp_arrow` gives

It is assumed that `p` is suitably well-behaved, that is, that it crosses the small circle just once.

(Again, it is possible to do it without those assignments, but then the argument to the third `drawarrow` gets rather unwieldy.)

◇ Jeremy Gibbons
Oxford Brookes University
jgibbons@brookes.ac.uk

Fo. another thing, that '1' should be '`length p`', otherwise the macro will only work for a path of length 1.

Indeed, there is no need to draw just subpaths of p; there is no harm in drawing p itself twice:

```
def draw_dbl_arrow text t =
  drawarrow reverse p;
  drawarrow p
enddef;
```

In fact, the original poster asked for a triple-headed arrow, with two arrow heads at (for the sake of argument) the end of the path. If you can pick the right small value of e, you can achieve this by just drawing the path three times:

```
def draw_trp_arrow text t =
  drawarrow reverse p;
  drawarrow p;
  drawarrow subpath (0, length p - e) of p
enddef;
```

But what value to pick for e? You could just experiment, but different values will be needed for different paths to get consistent results. A better approach is to find the time at which a point traversing path p is a certain fixed distance (namely, the length of an arrow head, `ahlength`) from the end of p, and to draw the corresponding subpath of p. You can find that time using `intersectiontimes`, intersecting p with a circle of the appropriate size centred on the end of p.

```
def draw_trp_arrow text t =
  drawarrow reverse p;
  drawarrow p;
  path q;
  q := fullcircle scaled (2*ahlength)
       shifted (point (length p) of p);
  numeric tp, tq;
  (tp,tq) = p intersectiontimes q;
  drawarrow subpath (0, tp) of p
enddef;
```

On the same path as before, `draw_trp_arrow` gives

It is assumed that p is suitably well-behaved, that is, that it crosses the small circle just once.

(Again, it is possible to do it without those assignments, but then the argument to the third `drawarrow` gets rather unwieldy.)

◇ Jeremy Gibbons
Oxford Brookes University
jgibbons@brookes.ac.uk

LaTeX

Default Docstrip Headers

LaTeX3 project team

Many LaTeX users now distribute packages in documented source form using the docstrip system. Docstrip allows a header to be placed on generated package files, suitable for copyright information or distribution conditions.

If the docstrip install file distributed with a package does not provide an explicit header, the docstrip system will add a default header to all generated files. Previously the default header was the following text:

```
This is file 'myfile.sty',
generated with the docstrip utility.

The original source files were:

myfile.dtx  (with options: 'package')

IMPORTANT NOTICE:

For the copyright see the source file.

You are *not* allowed to modify this file.

You are *not* allowed to distribute this file.
For distribution of the original source see the terms
for copying and modification in the file myfile.dtx.
```

Unfortunately the above conditions make it illegal to distribute the generated file, even if the full sources are included, thus making it impossible to include the package in 'ready to run' distributions such as the TeX Live CD.

Starting with the 1998/06/01 release of LaTeX the default header has been changed to allow such usage; it is now as follows:

```
This is file 'myfile.sty',
generated with the docstrip utility.

The original source files were:

myfile.dtx  (with options: 'package')

IMPORTANT NOTICE:

For the copyright see the source file.

Any modified versions of this file must be renamed
with new filenames distinct from myfile.sty.

For distribution of the original source see the terms
for copying and modification in the file myfile.dtx.
```

```
This generated file may be distributed as long as the
original source files, as listed above, are part of
the same distribution. (The sources need not
necessarily be in the same archive or directory.)
```

Note: This change *does not* affect any distribution that sets an explicit preamble in the docstrip install file. In particular it does not result in any changes to the distribution conditions placed on files generated from the base LaTeX distribution.

If you currently distribute a package on a public archive which does not specify a docstrip preamble (with the `\preamble` or `\usepreamble` commands in the docstrip install file) then this change **will affect you**.

- We hope that you will prefer the new default, which allows your package to be used in many of the more popular TeX distributions.
- However, if you prefer the more restrictive distribution conditions in the previous releases then you will need to update your install file to specify the command

  `\usepreamble\originaldefault`

  before the commands generating the affected package file.

We apologise that this change potentially requires package authors to update their files on the public archives, but the old default text caused great problems for many distributors and there was no way to change the default behaviour without affecting existing files using that default.

References

[1] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński, and Mark Wooding. The `docstrip` program. Part of the LaTeX distribution as file `docstrip.dtx`.

[2] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LaTeX Companion*, chapter 14. Addison-Wesley, Reading, Massachusetts, 1994.

◇ LaTeX3 project team
Johannes Braams
David Carlisle
Alan Jeffrey
Frank Mittelbach
Chris Rowley
and Rainer Schöpf
`latex-l@relay.urz.uni-heidelberg.de`

Editor's note: To join the mailing list for the LaTeX3 project, send email to `listserv@relay.urz.uni-heidelberg.de`. The body of the message should contain one line:

```
subscribe latex-l firstname lastname
```

LaTeX News

Issue 9, June 1998

New math font encodings

A joint working group of the TeX Users Group and the LaTeX3 Project is developing a new 8-bit math font encoding for TeX. It is designed to overcome several limitations and implementation problems of the old math font encodings and to simplify switching between different sets of math fonts, much as the LaTeX font selection interface has simplified switching between text fonts.

Since the work on this project relies entirely on volunteer work, we cannot give a specific release date yet. However, a prototype implementation already exists. This contains several sets of virtual fonts, some LaTeX packages and a kernel module; we hope to integrate it into the main LaTeX distribution for the next release.

Documents using only standard LaTeX commands for math symbols should not be affected by switching to the new math font encodings However, documents, classes or packages making specific assumptions about the encoding of math symbol fonts are likely to break.

Further information about the Math Font Group may be found on the World Wide Web at `http://www.tug.org/twg/mfg/`.

A new math accent

A new math accent, `\mathring`, has been added. This is a math mode version of the ring accent (˚) which is available in text mode with the command `\r`.

Extended `\DeclareMathDelimiter`

The command `\DeclareMathDelimiter` has been extended. Normally this command takes six arguments. Previously, when being used to declare a character (such as `[`) as a delimiter, a variant form was used with only five arguments. The argument specifying the default 'math class' was omitted. Now the full six-argument form may be used in this case. The extra information is used to implicitly declare the character via `\DeclareMathSymbol` for use when the symbol is not used with `\left` or `\right`.

The old five-argument form is detected and will work as before.

Tools distribution

The `multicol` package now supports the production of multiple columns without balancing the last page. To get this effect use the `multicols*` environment.

The `layout` package was partly recoded by Hideo Umeki to display page layout effects in a better way.

As suggested by Donald Arseneau, the `calc` package was extended to support the new commands `\widthof{`*text*`}`, `\heightof{`*text*`}`, and `\depthof{`*text*`}` within a `calc`-expression. At the same time we modified a few kernel commands so that `calc`-expressions can now be used in various useful places such as the dimension arguments to the `tabular` environment and the `\rule` command. For many other standard LaTeX commands this was already possible.

Support for Cyrillic encodings

We are very pleased that, after a lengthy period of development, a set of fonts, encodings and support files for using LaTeX with Cyrillic characters will soon be available.

Test versions of the 'LH' fonts for these Cyrillic encodings, based on the Computer Modern design, are available from CTAN archives in the directory `fonts/cyrillic/lh-test`. The LaTeX support files (by Werner Lemberg and Vladimir Volovich) are also available from CTAN archives in `macros/latex/contrib/supported/t2`

Default docstrip header

Many LaTeX users now distribute packages in documented source form using the docstrip system. Docstrip allows a header to be placed on generated package files, suitable for giving copyright information, or distribution conditions.

We have changed the default version of this header so that it allows stripped files to be distributed in ready-to-run installations such as the TeXLive CD. If you use the default header for distributing your files you should check that the new copyright text is acceptable to you. The file `docstrip.dtx` explains how to produce your own header if you wish to do so.

TEXNortheast Conference

TEXNortheast Conference: Final Report

Stephanie Hogue

Last March, the penthouse suite of the Loews New York Hotel was the site of U.S. TUG "history in the making". TEX users gathered for a non-annual-meeting, three-day conference. Held Sunday, March 22, through Tuesday, March 24, the TEXNortheast Conference was positioned both to follow the Seybold conference (held the preceding week) and to attract the local publishing community. With the theme "TEX/LATEX Now", the Conference promised — and delivered — practical information for those whose working lives revolve around TEX and LATEX.

The TEXNortheast Conference grew from a general discussion at the 1997 Annual Meeting in San Francisco, concerning a decline in U.S. membership and the needs of people utilizing TEX and LATEX in their jobs. A group of people attending the San Francisco meeting stepped forward to take on the task of organizing a conference for March 1998; they were later joined by several more volunteers. Serving in various capacities on the conference committee were:

Nancy Chien	Texas A&M University
Don DeLand	Integre Technical Pub. Co.
Susan DeMeritt	IDA/CCR La Jolla
Alan Hoenig	*Program Co-Chair*, CUNY
Anita Hoover	*Program Co-Chair*, University of Delaware,
Stephanie Hogue	The TypeWright
Mimi Jett	ICC Oregon
Cheryl Ponchin	IDA/CCR Princeton
Stacey Sensenig	Cadmus Journal Services/ TAPSCO
Heidi Sestrich	Carnegie Mellon University

Due to the time crunch (about three months to arrange the program) and the practical theme of the conference, a different approach to soliciting papers was adopted. The committee gathered a list of suggested topics and surveyed the membership for their responses, as well as more suggestions. The final topic list was then incorporated into the "Call for Papers". Additionally, committee members actively recruited potential speakers to present specific topics and to offer workshops. Several committee members agreed to give presentations and/or workshops themselves, in order to ensure that the conference met attendees' expectations.

The conference committee also approached commercial vendors of TEX implementations, offering scheduled times for each to demonstrate the "latest and greatest" features of their products.

Lance Carnes and Ed Lajarza	Personal TEX Inc.
Richard Kinch	TrueTEX Software
Barry Smith	Blue Sky Research

accepted the committee's invitation. Don DeLand also offered to demonstrate Scientific Word/Workplace on behalf of TCI/Brooks Cole. In addition to the vendors giving presentations, several providers of TEX-related services also had display tables:

Integre Technical Pub. Co.
Interactive Composition Corp.
TEXnology, Inc.

Blue Sky Research announced Textures' newest feature — *Synchronocity* between input and `.dvi` files — at the conference, and provided a Macintosh running a demonstration. The glass-walled vendor display area quickly became the center of activity for conference participants.

The "recruitment" of speakers and vendors resulted in three information-packed, albeit long, days, each devoted to a specific topic:

Sunday, "All About TEX" The conference opened with vendor demonstrations and, for those who brought laptops, the opportunity to install a trial copy of PCTEX, Scientific Workplace, Textures, or the new TEX Live 3 CD, to "test drive" during the conference. Talks on general TEX issues and virtual fonts by Alan Hoenig and a BIBTEX discussion with Oren Patashnik completed the morning session. Workshops filled the afternoon, with sessions on moving from LATEX 2.09 to 2_ε, using the amsmath package, customizing LATEX lists, and using packages that extend the `tabular` environment.

Monday, "WWW and Interactive TEX" The second day featured TEX in combination with other packages and on the Web. Presentations of TCI's EXP, IBM's techexplorer, PC TEX's new graphics features, Mathscape, and Mike Sofka's

talk on tagged DVI files (not for the faint-hearted!) occupied the morning. In the afternoon, Ross Moore presented his LaTeX2HTML package, and Michael Downes reported on his work to develop an environment that will *automatically* break long equations. The day finished with a LaTeX2HTML workshop.

Tuesday, "TeX in Publishing" The closing day was devoted to issues of professional publishing with TeX. Designing books, supporting multiuser macro packages, developing database publishing systems, and designing for the Web as well as print were discussed during the morning session. The final session covered custom-designing a legal document with TeX, controlling white space in TeX, and using alternative math fonts, and concluded with an overview of John Hobby's METAPOST language for producing PostScript graphics.

Each day during lunch, participants divided into groups for informal discussions of TeX/LaTeX issues and the future direction of TUG. On the final day, participants' names were drawn at random, and, thanks to the generosity of Personal TeX and TCI/Brooks Cole, copies of the latest versions of PC TeX, Scientific Workplace, and EXP were awarded. Other winners received advance copies of the **TeX Live 3** CD, thanks to the efforts of Sebastian Rahtz and Anita Hoover, and the support of the University of Delaware.

The total number of participants for the full, three-day conference was 57. One-day registrations were also accepted; attendance was highest on Tuesday (63). Ross Moore undoubtedly traveled the farthest — all the way from Macquarie University in Australia! The enthusiastic responses of the participants, some attending their first TUG conference, made it clear that the TeXNortheast Conference was a "hit". The most frequently heard question was, "When's the next one?"

If you missed the conference, you can check out the Web page:

`http://lib.stat.cmu.edu/~heidi/tug97.html`

or click on the link from TUG's home page (`http://www.tug.org`). The daily agendas, brief abstracts of presentations, and the list of participants are still available.

Several of the committee members have "re-enlisted" to help plan the 1999 Annual Meeting. We intend to continue some of the innovations which made the TeXNortheast Conference a success:

- Recruiting speakers for specific topics: The suggested topic list had more material than we could cover in three days; we will use the list to help design presentations and workshops for the 1999 meeting.
- Surveying the membership: Feedback from the survey respondents enabled us to design a program that people wanted to attend; we hope for an even bigger response to the next survey.
- Parallel sessions: Although we were not able to arrange them at this conference, we think parallel sessions would allow us to accommodate those who need practical, how-to sessions, as well as those who prefer more theoretical discussions.
- Using laptops: While the use of laptops will require more technical support, it was apparent that this approach could lower the cost of providing hands-on workshops.

The TeXNortheast Conference Committee wants to thank everyone who helped to make this conference such a success, especially the speakers and vendors who put together terrific presentations and workshops on very short notice. We particularly appreciate the developers' efforts to explain their packages in layman's terms and to provide concrete examples. No one present at Michael Barnett's Mathscape presentation will soon forget his helpful and hilarious explanation of three-dimensional math.

While the committee planned to impart practical information to users, we were pleasantly surprised by developer Ross Moore's comment that he, too, learned something from the workshops. It seems he found helpful information about how some of the packages are applied by users in actual work situations. We look forward to making the 1999 Annual Meeting an even more valuable exchange of information and ideas.

◇ Stephanie Hogue
The TypeWright
801 Highland Road
Lansdale,PA 19446 USA
shogue@typewright.com

TEXNortheast: Workshops and additional papers

Summaries follow for workshops as well as for papers which have been published elsewhere, or for which no final text was received by the TEXNortheast Program Committee.

Workshops

Moving On: LATEX 2.09 to LATEX 2ε

Anita Z. Hoover

Prerequisites: Little or no experience in LATEX 2ε and most familiar with LATEX 2.09 conventions.

Description: Learn the basics to convert a document from LATEX 2.09 to LATEX 2ε. The focus was on

1. Discuss the differences between LATEX 2.09 and LATEX 2ε;
2. New features in LATEX 2ε;
3. Standard classess, packages, and options; and
4. Custom packages.

◇ Anita Z. Hoover
University of Delaware
anita@udel.edu

More Multiline Equation Environments

Stephanie Hogue

Prerequisites: Basic knowledge of standard LATEX math environments, including `eqnarray` and `array`.

Description: This workshop was an introduction to the multiline equation environments of the amsmath package for LATEX 2ε, which supersedes the amstex package. The following environments were discussed:

- `gather`, `multline`: environments without alignment across lines;
- `split`, `align`, `flalign`, `alignat`: environments with one or more alignments across lines.

The discussion included guidelines for breaking equations, according to the AMS. Enhancements to equation numbering were also addressed.

This was *not* an exhaustive presentation of the amsmath package. Complementary material on font issues in amsmath was presented in Anita Hoover's workshop "Moving On: LATEX2.09 to LATEX 2ε".

◇ Stephanie Hogue
The TypeWright
shogue@typewright.com

Customizing LATEX Lists

Donald W. DeLand

Prerequisites: Intermediate LATEX 2ε for Authors workshop, or solid understanding of LATEX fundamentals.

Description: The `\list` mechanism is the basic building block of most non-sectioning LATEX environments. This workshop reviewed the generic LATEX environments that use `\list`, and how they are constructed. The following more advanced topics were covered in detail:

1. Changing default indents, labels, and vertical spacing using `\list` parameters and localized definitions of `\makelabel`.
2. Adding an optional argument to `\begin{enumerate}` to "clear for widest label" by using `\@ifnextchar` and linking the `\leftmargin` to the `\labelwidth`.
3. Using `\newcounter` and `\refstepcounter` to write theorem-like environments *without* using `\newtheorem`.
4. Tricks of the trade and aside comments:
   (a) Adding design elements using `\item`
   (b) Marking "optional" list items (e.g., in exercises or sections)
   (c) Boxing a theorem or definition
   (d) Enumerating horizontally rather than vertically
   (e) Why `\hangindent` and `\hangafter` don't work within a `\list`

◇ Donald W. DeLand
Integre Technical Publishing Co.
deland@cs.unm.edu

Beyond Tabular

Stephanie Hogue

Prerequisites: Basic knowledge of standard LATEX `tabular` environment.

Description: This workshop was an overview of several packages which provide enhanced features for tabular material. The following packages were presented:

- array: provides some new preamble options in addition to those found in the `tabular` environment;
- tabularx: automatically calculates *column* widths for a table of specified width;
- longtable: automatically breaks a long table across pages;

- dcolumn: provides a new column type for specifying a decimal-aligned column.

◇ Stephanie Hogue
The TypeWright
shogue@typewright.com

"WYSIWYG" LaTeX: EXP and Scientific Word/Workplace/Notebook

Donald W. DeLand

Two WYSIWIG applications — Simon Smith's EXP and TCI's Scientific Word/Workplace/Notebook — allow authors to create LaTeX documents without learning LaTeX. This talk reviewed and demonstrated the major features of both programs, and explored some of their limitations with respect to document design, user interface, and LaTeX compatibility.

EXP is a "scientific word processor" whose word-processing features are easy to learn, but EXP documents need to be set up in a particular way to guarantee a smooth transition to LaTeX. Although EXP is easy to use, its automatic numbering mechanisms, lack of macro support, and inability to handle large tables or import non-EXP documents make it cumbersome to work with.

One major strength of TCI's Scientific Workplace is its built-in support for Maple, a popular computer-algebra system. Scientific Workplace also includes a style editor that lets the user customize numerous design elements, then process the document via LaTeX for outputting. There is a great deal of confusion, however, as to what the relationship is between Scientific Workplace and LaTeX. Workplace uses LaTeX as its output (print) engine, but it does not generate a "clean" LaTeX document.

◇ Donald W. DeLand
Integre Technical Publishing Co.
deland@cs.unm.edu

Making Web Sites using LaTeX2HTML

Ross Moore

LaTeX2HTML is an extremely flexible tool for creating Web pages. Indeed it is best used when requiring technical information to be presented as a 'Web' of linked HTML pages.

One immediately encounters questions like:

- How many HTML pages?
- How much information should go on each page?
- How to link pages for easy access to related pieces of information?
- Indexing, Table-of-Contents and other Navigation aids.

The aim of this workshop was to get some familiarity with the way LaTeX2HTML tackles these issues, using configuration variables and command-line switches.

The LaTeX source provides the information presented, but there are many options available to affect the appearance and arrangement of the resulting Web pages.

◇ Ross Moore
ross@mpce.mq.edu.au

Presentations not included in this issue

Virtual Fonts

Alan Hoenig

TeX makes special demands on the fonts that it works with. Although this presents no problem for fonts (like Computer Modern) that were created explicitly for use by TeX, what do we TeX users do if we want to use any of the hundreds of beautiful fonts provided by mainstream digital foundries? The concept of virtual fonts provides this mechanism — for this and much more, as this talk will demonstrate. Discussion will center about available virtual font tools and some simple virtual font projects.

Editor's note: This article appeared in *TUGboat* **18** (2) pp. 113–121.

◇ Alan Hoenig
ajhjj@cunyvm.cuny.edu

Breaking Equations

Michael Downes

Some flaws in the way TeX and LaTeX handle displayed equations are of such long standing that they are scarcely noticed any more except by beginning users — for example, the fact that `\left ... \right` constructs cannot span multiple lines, if an equation must be broken into more than one line. Other flaws that have to do with relatively subtle typographical issues go unnoticed by most users — for example, the fact that in multi-line equations `\abovedisplayshortskip` isn't applied when applicable, and intra-line shrink isn't used when available.

This is a report on a new LaTeX package called "breqn" that substantially eliminates many such problems. One of its main goals is to support automatic linebreaking of displayed equations, to the extent possible within the current limitations of TeX and LaTeX.

Editor's note: This paper appeared in *TUGboat* **18** (3), pp. 182–194.

◇ Michael Downes
mjd@math.ams.org

Designing Books with TEX in Mind

Donald W. DeLand

This paper presents an overview of TEX's structure and how that structure impacts the implementation of book designs. Most book designs cannot be implemented using only TEX's internal components; rather, design implementation usually involves a combination of TEX and PostScript, and further depends on the specific font encodings and PostScript drivers used by the operating system and TEX implementation being used. The programmability of TEX combined with the flexibility of PostScript can be powerful. On the other hand, TEX predates PostScript, so the two do not always merge gracefully.

Specific design issues covered here include selecting fonts for use with math, using graphics as design elements, limitations in setting multicolumn text, and a discussion of how TEX's paragraph-building and page-breaking mechanisms impact marginal text and color usage. In addition, this paper presents some examples of how TEX's programmability can be used to automate or simplify design elements that could only be handled manually in other typesetting or desktop systems.

◇ Donald W. DeLand
Integre Technical Publishing Co.
deland@cs.unm.edu

Custom Legal Documents for the Auto Loan Exchange

Douglas Lovell

The Auto Loan Exchange is a project of IBM Research which connects automobile dealerships directly to lenders and credit bureau reporting services for rapid approval and funding of automobile loans. We have used TEX to typeset the loan contract and related documents required to complete the loan and close the automobile purchase.

In many ways, TEX was the perfect choice to satisfy our document needs. We have been able to eliminate the preprinted forms stocked by dealers and instead, print complete contract documents customized for each loan. We will discuss the unique document requirements of this internet commerce application and describe our TEX-based solution.

Editor's note: This article appeared in *TUGboat* **18** (3), pp. 175–181.

◇ Douglas Lovell
dcl@us.ibm.com

Hops, Skips, and Jumps: White Space

Joe Weening

An important part of the appearance of a document is the proper use of white space. Obeying well-established traditions of typography helps the reader to understand the document better. Failing to follow these rules may cause confusion and draw the reader's attention away from the content of the document.

TEX tries to insert the proper amount of white space wherever it can, but it sometimes gets it wrong. It is then up to the author of the document, or someone else editing the TEX file, to find and correct these errors.

In this talk we will explain TEX's rules for inserting white space, describe cases in which they don't work correctly, and explain how to get TEX to insert the right amount of space. We will include examples from TEX's horizontal mode, vertical mode, and math mode.

◇ Joe Weening
jweening@ccrwest.org

Introducing METAPOST

John Hobby

METAPOST is a picture-drawing language very much like MetaFont except with PostScript output. I will give a brief overview of the METAPOST language and discuss drawing and filling, dashed lines, using TEX and LATEX output, and the graph-drawing package.

◇ John Hobby
hobby@research.bell-labs.com

mathscape — Combining Mathematica and TEX

Michael P. Barnett
Department of Chemistry,
Princeton University,
Princeton, N.J. 08540
michaelb@princeton.edu
http://www.princeton.edu/~allengrp/ms

Preliminaries

Millions of mathematical formulas are typeset annually. Most of the numbers we see in print are produced by computer. So are the indexes and catalogs issued by database publishers. Charts and diagrams and other products of computer graphics have replaced manually drafted copy. But most of the formulas in mathematics, engineering and science publications are still derived and coded by hand.

The `TeXForm` function in Release 2 of Mathematica [1], and some more extensive resources in Release 3 [2], provide a bridge between symbolic computation and computer composition. The author's mathscape system was designed to strengthen the bridge. Written in Release 2 of Mathematica, it is in ongoing use by the author, and it has produced several hundred typeset pages of heavily mathematical material already. It subsumes work reported previously as `bilo` and `forTeX` [3]. It produces a document from a control file containing:

- statements that Mathematica evaluates for inclusion in the output,
- formatting information and other statements to be executed silently, flagged by the `#` symbol,
- text coded in LaTeX, with each record flagged with an `*`, or in a `text` environment between `# beginText` and `# endText` markers.

Then, within a Mathematica session, the mathscape package is loaded, and the mathscape statement `autorecord[`*controlFileName*`]`:

- makes Mathematica read the control file and convert its contents to the LaTeX coded representation of the document that is being created,
- invokes LaTeX to convert this to a `dvi` file,
- invokes a preview program, and
- prints the typeset product if requested.

In this way, the document can be crafted interactively. Graphics can be incorporated with ease.

The system was started to meet some major needs of research publication. The production of problem sets and worked examples for teaching has been addressed extensively. So has the production of tables of formulas for reference. A tutorial introduction to mathscape and a systematic review are available [4].

The production of the following boxed output illustrates the control file conventions.

$$y^2 - x^2$$

is converted by `Factor` to:

$$(y - x)(y + x)$$

Here, formatting is needed to override the default arrangement $-x^2 + y^2$ and $(-x + y)(x + y)$ imposed by Mathematica. mathscape converts the immediate result v of a Mathematica evaluation to `prep[`v`]`. `prep` is initialized to `Identity` and reassigned dynamically, in the present case to a function that reverses every `Plus`. The portion of the input that produced the contents of the preceding box is:

```
# prep = toEach[Plus][reverse]
s = y^2 - x^2
* is converted by \verb|Factor| to:
s // Factor
```

mathscape supports a large open-ended class of functions, typified by `toEach[Plus]`, that "target" particular portions of an expression. This can be identified by head, e.g., `toEach[Plus]`, `toEach[log]`, as a Mathematica pattern e.g., `toThe[_Integer+_]`, by part name, e.g., `toTheLhs`, `toTheNumerator` or, as in `to[Plus][containing[x], outermost]`, by head and criterion, or by pattern and criterion.

Playing through to TEX

mathscape passes elementary algebraic expressions to the Mathematica `TeXForm` function for conversion to corresponding TEX code. Greek letters, the names of all the special symbols in the TEX vocabulary and some other unparameterized objects, e.g., `strut`, are denoted by the TEX control sequence names without the `\`. The names of binary operators (e.g., `oplus`) are given appropriate mathematical properties, too. Function expressions are used

for parameterized objects, e.g., `hat[x]`, `rule[`*rise*`][`*width*`,`*height*`]`, `overbrace[`*tag*`][`*expr*`]` that map into TEX codes in just a few simple ways.

Other names can be used in the body of a calculation and then changed to the TEX names by replacement rules assigned to `prep`. The statement `newSymbol[`v`]` makes `mathscape` append v to the list of identifiers for unparameterized TEX codes. Symbols can be appended to the lists of other control sequence names by further functions that write the definitions to the output.

The built-in Mathematica names and the lowercase names, e.g., `Cos`, `cos`, for the typographically "cos-like" functions are converted to TEX sequences that provide the conventional omission/inclusion of parentheses and placement of exponent, as in:

```
cos[x], cos[x]^2, cos[x+y]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ $\cos x\,,\ \cos^2 x\,,\ \cos(x+y)$

(We use the $\rhd\!\!\rightarrow$ and $\overset{resp}{\rhd\!\!\rightarrow}$ symbols between single or multiple verbatimized input expressions and the typeset products.) In the output, parentheses are put around the arguments of functions that do not have special typographic status. Thus:

`f[x], g[u,v]` $\overset{resp}{\rhd\!\!\rightarrow}$ $f(x),\ g(u,v)$

Special bracketing is illustrated by:

`enbr[x], f[enbr[x]]` $\overset{resp}{\rhd\!\!\rightarrow}$ $[x],\ f[x]$

`enpr[enpr[x]], f[ompr[x]]` $\overset{resp}{\rhd\!\!\rightarrow}$ $((x)),\ fx$

`ensp["|", ">"][x, y]` $\rhd\!\!\rightarrow$ $|x,\ y>$

`sapr[x/y]` $\rhd\!\!\rightarrow$ $\left(\frac{x}{y}\right)$

Further `en` and `sa` functions provide other fixed-size and self-adjusting bracketing symbols. Typically, these are introduced after the body of a symbolic computation by targetting expressions in `prep`.

The infix treatment of binary operators, relationship symbols and arrows in the output, is shown by:

`otimes[x, oplus[u,v,w]]` $\rhd\!\!\rightarrow$ $x\otimes(u\oplus v\oplus w)$

`ll[a,b,c]` $\rhd\!\!\rightarrow$ $a\ll b\ll c$

`not[prec][u,v]` $\rhd\!\!\rightarrow$ $u\not\prec v$

`rightarrow[a,b,c]` $\rhd\!\!\rightarrow$ $a\to b\to c$

`arrowoo[u,v]` $\rhd\!\!\rightarrow$ $u\rightarrowtail v$

The conventions for single and multiple subscripts and superscripts, on the right and/or left of a symbol are illustrated by:

```
x@sub@1, x@sup@enpr[m@sub@1], P@subsup[n, m]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ $x_1,\ x^{(m_1)},\ P_n^m$

`x@subscriptSequence[a,b]` $\overset{resp}{\rhd\!\!\rightarrow}$ $x_{a,b}$

```
E@lsub@r, E@lsubsup[r, epsilon]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ ${}_rE,\quad {}_r^{\epsilon}E$

The conventions for decorations, ties, rules and composites are illustrated by:

```
hat@x, breve@Psi, widetilde@enpr[tilde@A]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ $\hat{x},\ \breve{\Psi},\ \widetilde{(\tilde{A})}$

`underline[x+underline[y]]` $\rhd\!\!\rightarrow$ $\underline{x+\underline{y}}$

```
f[u] + overbrace["time\ dependent"][
                              g[t,u] + g[t,w]]
```

$\rhd\!\!\rightarrow$ $f(u)+\overbrace{g(t,u)+g(t,w)}^{time\ dependent}$

`rule[5pt][30pt, 1pt]` $\rhd\!\!\rightarrow$

`atop[a, b], above[1pt][a, b]` $\overset{resp}{\rhd\!\!\rightarrow}$ $\begin{matrix}a\\b\end{matrix}$, $\frac{a}{b}$

`stackrel[F, "="], ddrel[arrowcc, a, b]`

$\overset{resp}{\rhd\!\!\rightarrow}$ $\overset{F}{=}$, $\overset{a}{\underset{b}{\rhd\!\!\rhd}}$

`overlay[vee, wedge]` $\rhd\!\!\rightarrow$ $\curlyvee$

The effects of some simple catenation functions are shown by:

`sequence[a, b, c, d]` $\rhd\!\!\rightarrow$ $a,\ b,\ c,\ d$

`catenation[X, scriptscriptstyle[path], Y]`

$\rhd\!\!\rightarrow$ $X_{path}Y$

`markedCatenation[cdots][a, b, c]`

$\rhd\!\!\rightarrow$ $a\cdots b\cdots c$

Fonts styles and sizes are specified by TEX names. Also, `sizedFont[1]`, ... alias `tiny`, ... Thus,

```
rm[a b^2], bf[a b^2], sansSerif[a b^2]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ $\mathrm{ab}^2,\ \mathbf{ab^2},\ \mathsf{ab}^2$

```
boldmath[a b^2], boldmath[cal[ABCD]]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ $\boldsymbol{ab^2},\ \boldsymbol{\mathcal{ABCD}}$

```
tiny[a b], sizedFont[3][a b]
```

$\overset{resp}{\rhd\!\!\rightarrow}$ ${}_{\mathrm{a\ b}},\ ab$

`mathscape` uses TEX primitives in the basic alignment process, too. Every display is built using `hbox`, `vbox`, `hboxTo`, `vboxTo`, `hspace`, `vspace`, `newlength`, `addtowidth`, `newbox`, `phantom`, `setbox`, `copy`, `wd`, `ht`, `dp`, and related constructs that translate directly to TEX or to local macros.

Varying the style

Alternative notations often exist for the same mathematical expression. mathscape lets the user change these freely. Thus, logical expressions are set in &|¬ notation by default. The assignment `logicStyle=2` changes this to the $\wedge\ \vee\ -$ notation. `logicStyle=1` restores the default.

Square roots introduce a more general tactic. Following the action of `prep`, `sqrt[`z`]` is converted to `style[sqrt, defaultSqrtStyle][`z`]`. Initially, the style parameter is 1, giving the radical notation $\sqrt{z}$. Changing it to 2 and 3 give $z^{1/2}$ and $z^{\frac{1}{2}}$ respectively. In general, `useStyle[`n`]` converts f`[`z`]` to `style[`f, n`][`z`]`. It is used to mix styles within a single expression, as in the production of:

$$(1-\sqrt{\delta})^{1/2}$$

from

```
# prep = to[sqrt][1][useStyle[2]]
sqrt[1 - sqrt[delta]]
```

Fractions are built up, with the numerator and denominator of just the outermost fractions in the `displaystyle` mode, when `defaultFractionStyle` is 1. Style 2 puts all the numerators and denominators in `displaystyle`. Styles 3 and 4 give shilling and reciprocal notations. Styles 1.1, 1.2, ..., and 2.1, 2.2, ... strengthen the fraction bar and lengthen the shilling slash. For powers, style 2 gives radical notation, e.g., $\sqrt[3]{x}$, when the exponent is a fraction.

Representations

We represent derivatives, integrals, matrices, sums and many other composite mathematical objects in a way that facilitates mechanical operations and allows flexible styling in the typeset output. The handling of partial derivatives, shown next, is typical.

```
D$[x][y], D$[x, 2][y], D$[x,y,z][phi]
```

$$\overset{resp}{\rhd\!\!\rightarrow}\ \frac{\partial y}{\partial x},\ \frac{\partial^2 y}{\partial x^2},\ \frac{\partial^3 \phi}{\partial x \partial y \partial z}$$

mathscape contains extensive suites of procedures to manipulate expressions represented by "compound heads", such as `D$[x]`, `Dt$[x]` (for a total derivative), `sum[i, j, k]`, `integral[x, 0, infinity]`, and `matrix[m,n,M,N]`. Style is controlled by the `setOptions[D$, placement -> subscript]` statement and its counterparts. These create intermediate `style[...][...]` expressions, that for the current `D$` example, leads to subscript placement of the variables of differentiation, as in $\phi_{x,y,z}$.

Environments

By default, mathscape centers the typeset Mathematica statements in a field that is `widthForMath` wide. The commands `alignLeft`, `alignRight` and `alignCenter` are put in `#` statements to change the alignment. `leftIndent` and `rightIndent` control the indentions. The `displayBoth` command produces verbatimized input and conventionally styled output. `pairHorizontally` makes the output run-on, and `pairVertically` makes it start a new line. The commands `displayInput` and `displayOutput` display just the input and output, respectively. The input can be modified before evaluation and/or before display, by actions that the user specifies.

Within an `alignOnEvalSym` environment, begun and ended by appropriate `begin...` and `end...` statements, all the displays, containing input and output are aligned on the $\rhd\!\!\rightarrow$ and $\overset{resp}{\rhd\!\!\rightarrow}$ symbols. The arrows are placed at the middle of the print region, by default. This is overridden by assigning a value to `inputField`.

Consecutive tags are created in the `tagging` environment. By default, these are parenthesized undivided Arabic numerals, i.e., (1), (2), ... In general, the tag consists of the left marker, `tagPrefix`, `tagSeparator`, `tagNumber`, and the right marker. `tagStyle`, e.g., `letter`, `roman`, `Letter`, determines the style of the sequence number. The markers are combined in `tagMarker`. `tagDown` uses the present prefix, separator and tag number to prefix the subordinate sequence numbers that start again at 1. `tagUp` restores all the tagging parameters in force before tagging down. `tagSide` defaults to `right`, and can be reassigned to `left`.

The `alignOnEqual` environment aligns on the first `=` symbol in the concomitant displays. These may be separated by text. The left and right fields have equal width by default. This is overridden by assignment to `leftWidth`. The environment is an alias for `alignOnRelSym`, which treats all the relationship symbols and `Infix` operators as equivalent.

The `aligningItems` environment is used in:

```
# beginAligningItems; itemWidth = 25pt;
  leftIndent = sequenceGap = 0pt;
  itemsPerLine = 6; itemAlignment = right;
  bar = rule[10pt, 0.2pt]
* Fill in the blanks, in this list:
Table[Prime[Prime[n]], {n, 12}] //
 ReplacePart[#, bar, {{1},{2},{6},{9}}]&
* and in this:
{14, 34, bar, 59, bar, 125}
# endAligningItems;
```

This produced:

Fill in the blanks, in this list:

__	__	11	17	31	__
59	67	__	109	127	157

and in this:

14	34	__	59	__	125

The `runOnGroup` and `tabbedRunOnGroup` environments can be used in a variety of ways. The following simple example

$$\sum_i s_i \qquad \sum_{i=j}^{k} s_i \qquad \prod_{i=j}^{k} s_i$$

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

is produced by

```
# beginRunOnGroup; runOnStyle = compressed;
  continuationSymbol = "";
sum[i][s@sub@i]
sum[i,j,k][s@sub@i]
prod[i,j,k][s@sub@i]
# turnRunOnGroup
matrix[{1,2},{4,5}]
matrix[{1,2,3},{4,5,6}]
# endRunOnGroup;
```

In a `runOnGroup`, space between items on each line may be `compressed` or `expanded`. In `runOnGroup` and `tabbedRunOnGroup`, items may be tagged left or right, or untagged. Each group may be tagged left or right, or untagged, independent of item tagging. `continuationSymbol` defaults to ",". We set it to an arrow when successive items trace a reduction.

The next display shows another tracing tactic. `pipe` generalizes composition, so as to allow rules.

```
# newBinaryOperator[lplus, "+"];
  continuationSymbol = "rightArrow";
  s = lplus[a, times[b, c]];
  cm = toThe[times][Reverse];
  ca = toThe[lplus][Reverse];
  markWithAction;
  prep = pipe[toEach[_String][
   StringReplace[#,
    {"(cm)" -> "{\\cal C}_m",
    "(ca)" -> "{\\cal C}_a"}]&],
   List -> catenation]
s // pipeList[cm,ca]
```

$\rightarrowtail$

$$a + b \times c \xrightarrow{\mathcal{C}_m} a + c \times b \xrightarrow{\mathcal{C}_a} c \times b + a$$

`texTab[`*lcrString*`] [{`*lineData*`}]` plays through to the `tabular` environment. `hline` and `cline` symbols and `multicolumn` heads are wrapped into the data, using `prep`, to form the *lineData* list.

The `boxedPair` environment creates TEX files consisting of the codes for verbatimized input and the fully processed output. By default, these are input to `frameboxe`s joined by an $\rightarrowtail$. An option defers this to later `input` statements in the text.

The `textExpansion` and `runOnMath` environments embed evaluated results in the run-on text.

Interactive development

`autorecord` is recursive. A lengthy mathscape document is developed, typically, by writing separate control files for the successive parts, and invoking these from an overall control file. Optional arguments omit the xdvi step, convert to PostScript, invoke ghostview or xpsview, and print. The recursivity is used by `boxedPair`.

The `bypass` environment and `autobreak` function facilitate incremental testing. By conditionalizing the `beginBypass` and `endBypass` statements, different versions of a document, e.g., terse and detailed, can be produced from the same file. The `silentExecution` environment is used to set up variables and operators which are taken for granted in the printed exposition. The `evaluation` environment, in which work usually is conducted, is exited to allow the output of statements without execution.

The Mathematica graphics shell script `psfix` has been modified to omit boilerplate. New shell scripts wrap ghostscript and dvips to compensate.

Restructuring

The rearrangement and abbreviation of mathematical expressions is extremely important. `reverse`, used earlier, belongs to an extensible suite of procedures for these purposes. Several are used to form the next display from an equation that was saved in ordinary Mathematica style from a previous run. They all suspend `Orderless`ness of `Plus` and `Times` and encase the final result in `HoldForm`.

$$\sum_{n=0}^{\infty}\sum_{l=0}^{\infty}\sum_{m=0}^{\infty}[4\epsilon m A_{l,m,n-1}L_l(u)L_{m-2}(v) - 4\epsilon m^2 A_{l,m,n-1}L_l(u)L_{m-2}(v) + << 361 terms >> - 4(n+1)\epsilon m^2 A_{l,m,n+1}L_l(u)L_{m+2}(v)] \times L_n(w) = 0$$

This is produced by

```
# alignLeft; turnIndent = 1pc;
  prep =
   toTheLhs[
    to[Plus][outermost][
     showTerms[{1, 2, -1}]],
    toEach[_Integer + _][
     sortByAbsence[_Integer]],
    allowFurtherSorting,
   to[Times][outermost][
    splitBeforeFactor[2, times]],
   allowFurtherSorting,
   to[Plus][outermost][
    toTerms[containing[v]][
     sortByAbsence[v]],
    splitBeforeTerm[4,, "\\left."],
    splitBeforeTerm[3],
    splitBeforeTerm[2, "\\right."], sabr],
   disallowFurtherSorting,
   A[l_, m_, n_] ->
    A@subscriptSequence[l, m, n],
   L[n_, x_] -> L[sub[n]][x], e -> epsilon]
eqn[4.13]
```

The functions and rules in the arguments list of `toTheLhs` are executed consecutively, just like those of `pipe`. All the targetting functions act this way.

The two procedures `sortByAbsence`$[v_1, v_2, \ldots]$ and `sortByPresence`$[v_1, v_2, \ldots]$ meet many needs. These wrap `sortByCriteria` which works by selecting subsequences that satisfy the successive criteria instead of repeated swapping.

`splitBeforeTerm`$[n]$`[`s`]` and the corresponding `After`, `Factor`, `Element` and `Equal` expressions can specify continuity symbols, e.g., $\times$, and codes to balance stretchable brackets.

The procedure `showTerms[{`*indices*`][`s`]` and the similar `Factors`, `Elements`, `Arguments` procedures are used for `Plus`, `sum`, `Times` and `prod` expressions, and lists, matrices and arbitrary functions. Optional arguments control the depiction of of omitted items.

`allowFurtherSorting` removes `Orderless`ness and any `HoldForm`s. `disallowFurtherSorting` imposes `HoldForm` and restores `Orderless`ness.

Numerous situations arise that can be handled by adapting the general principles used in the procedures of this section, e.g., forcing the expressions that Mathematica ordinarily returns as $-u-v$ and z^{1-m} into $-(u+v)$ and $1/z^{m-1}$.

Because ease of understanding is our objective, mathscape contains substantial suites of procedures for convenient cross referencing between statements, and for fine-tuned factoring, distribution and collection. Graphics provides a powerful supplement in many ways. The abstract shows a depiction of a class of sparse matrices, that occur in an electronic energy calculation. Zero and non-zero elements are displayed as spaces and dots, respectively. Symbolic computation, graphics and typesetting come together in the production of diagrams and the synthesis of text throughout scholarly publication.

Past, present, future

The production of readable copy from the numerically represented results of symbolic computation motivated some of the earliest work on electronic typesetting. Formulas, produced by simple array manipulation were converted mechanically to the code of a paper tape driven photo-mechanical typesetter, for work in theoretical chemistry and planetary theory [5].

mathscape, started about six years ago, has gone through a few name changes, but has not undergone any structural change in the last three years. Its application to a variety of material has highlighted the need for the resources it provides. By enabling the mechanical production of readable discourse, this kind of work gives a fresh incentive to the formal study of mathematical derivation.

Acknowledgements

This work is part of the research program of the L.C. Allen Theoretical Chemistry Group. The author is grateful to Professor Allen for his support, and to K.D. Alexander and K.R. Perry of ICGL (now MECA) for their assistance during its development.

Bibliography

1. S. Wolfram, *Mathematica, A system for doing mathematics by computer*, Addison-Wesley, Reading, MA, 1991.
2. S. Wolfram, *The Mathematica Book*, Cambridge University Press, New York, 1996.
3. M. P. Barnett and K. R. Perry, Symbolic calculation for electronic publishing, *TUGboat* **15** (3) 285 – 292, Nov. 1994, and papers cited therein.
4. M. P. Barnett, *Symbolic Calculation for Electronic Publishing*, `http://www.princeton.edu/~allengrp/ms/scep.ps`, 1997.
5. M. P. Barnett, *Computer Typesetting, Experiments and Prospects*, MIT Press, Cambridge, MA, 1965, and papers cited therein.

Appendix

The main account [5] of mathscape contains numerous examples produced in the `boxedPair` environment. The TEX files for a selection of these were reset separately, converted to PostScript, and `input` to construct this Appendix.

The helium calculation

This page shows a summary of an automated check and extension of Pekeris' classical calculation of the electronic structure of helium like atoms. An `autorun` session produced a detailed narrative of both the conventional mathematical activity and its mechanization. Intermediate results were written out for subsequent computational use. The summary was produced from these.

The calculation involved partial differential equations, changes of variable, infinite series expansion, special functions of mathematical physics, determinants, and multiple integrals. Part of the calculation carried expressions that run to hundreds of terms. At several points, lengthy equations were broken into sets of smaller equations of specified form, for display and manipulation, using further mathscape procedures.

Graphics was used to plot numerical results conventionally, and to display the structure of a matrix as mentioned earlier. Also, the published version of a very lengthy formula was scanned, the image dissected, and the pieces imported as pictures between the corresponding pieces of the newly calculated result, for visual comparison. Some are shown in [3, 4].

We begin with the Schrödinger equation for a 2-electron atom with nuclear charge Z.

$$\frac{\partial^2\psi}{\partial r_1^2}+\frac{2}{r_1}\frac{\partial\psi}{\partial r_1}+\frac{\partial^2\psi}{\partial r_2^2}+\frac{2}{r_2}\frac{\partial\psi}{\partial r_2}+2\frac{\partial^2\psi}{\partial r_{12}^2}+\frac{4}{r_{12}}\frac{\partial\psi}{\partial r_{12}}+\frac{r_1^2-r_2^2+r_{12}^2}{r_1 r_{12}}\frac{\partial^2\psi}{\partial r_1\partial r_{12}}+$$
$$\frac{r_2^2-r_1^2+r_{12}^2}{r_2 r_{12}}\frac{\partial^2\psi}{\partial r_2\partial r_{12}}+2(E+\frac{Z}{r_1}+\frac{Z}{r_2}-\frac{1}{r_{12}})\psi=0 \qquad (1)$$

This is in standard texts. It is converted to the perimetric coordinates (2) where $\epsilon=\sqrt{-E}$.

$$u=\epsilon(r_2-r_1+r_{12}),\qquad v=\epsilon(r_1-r_2+r_{12}),\qquad w=2\epsilon(r_1+r_2-r_{12}) \qquad (2)$$

We use the equation for $\partial(u,v,w)/\partial(r_1,r_2,r_{12})$ and the consequent equations for the $\partial^2/\partial r_1^2,\ldots$ in terms of the $\partial^2/\partial u^2$. Hence:

$$4\epsilon^2\left\{u(2uv+2v^2+2uw+2vw+w^2)\psi_{uu}+\ll 6\text{ terms}\gg+2(2u^2+2v^2-w^2)\psi_w\right\}+$$
$$\left\{E(u+v)(2u+w)(2v+w)-2\epsilon(2u+w)(2v+w)+8\epsilon Z(u+v)(u+v+w)\right\}\psi=0 \qquad (3)$$

The wave function ψ is written as:

$$\psi=\mathrm{e}^{-(u+v+w)/2}F(u,v,w) \qquad (4)$$

Substitution in (3) gives an equation for F that is, in abbreviated form:

$$\{4Z(u+v)(u+v+w)-(2u+w)(2v+w)\}F+$$
$$2\epsilon\left\{u(2uv+2v^2+2uw+2vw+w^2)F_u+\ll 6\text{ terms}\gg+\right.$$
$$\left.(4u^2+4v^2-2u^2w-2v^2w-2w^2-uw^2-vw^2)F_w-2F(u+v)(u+v+w)\right\}=0 \qquad (5)$$

F is expanded as a triple series in Laguerre functions of u,v,w.

$$F=\sum_{\{l,m,n\}\geq 0}A_{l,m,n}L_l(u)L_m(v)L_n(w) \qquad (6)$$

Hence (8). The coefficient of each A contains Laguerre functions and their first two derivatives.

$$\sum_{\{l,m,n\}\geq 0}[-4\epsilon(u+v)(u+v+w)L_l(u)L_m(v)L_n(w)+\ll 8\text{ terms}\gg+$$
$$4\epsilon w(2u^2+2v^2+uw+vw)L_l(u)L_m(v)L_n''(w)]\ A_{l,m,n}=0 \qquad (7)$$

Occurrences of $L_l(u)$ and its derivatives times u and u^2 are converted to terms in $L_{l+\lambda}(u)$, $|\lambda|\leq 2$, using simple recurrence formulas. Terms containing v and w are treated correspondingly, giving a summand that contains (u,v,w) only as arguments of undifferentiated Laguerre functions.

$$\sum_{\{l,m,n\}\geq 0}[nL_l(u)L_m(v)L_{n-2}(w)+\ll 234\text{ terms}\gg-4L_n(w)L_{l+1}(u)L_{m+1}(v)+$$
$$\ll 127\text{ terms}\gg-n^2L_l(u)L_m(v)L_{n+2}(w)]\ A_{l,m,n}=0 \qquad (8)$$

Occurrences of $L_l(u)$ and its derivatives times u and u^2 are converted to terms in $L_{l+\lambda}(u)$, $|\lambda|\leq 2$, using simple recurrence formulas. Terms containing v and w are treated correspondingly, giving a summand that contains (u,v,w) only as arguments of undifferentiated Laguerre functions.

$$\sum_{\{l,m,n\}\geq 0}[nL_l(u)L_m(v)L_{n-2}(w)+\ll 234\text{ terms}\gg-4L_n(w)L_{l+1}(u)L_{m+1}(v)+$$
$$\ll 127\text{ terms}\gg-n^2L_l(u)L_m(v)L_{n+2}(w)]\ A_{l,m,n}=0 \qquad (9)$$

The coefficients of $L_{n+\nu}(w)$ are collected for each $v=-2,\ldots,2$. The summation is split into 5 parts corresponding to the different ν. These are re-indexed and combined, to give:

$$\sum_{n=0}^{\infty}\sum_{l=0}^{\infty}\sum_{m=0}^{\infty}[4\epsilon m\,A_{l,m,n-1}L_l(u)L_{m-2}(v)-4\epsilon m^2A_{l,m,n-1}L_l(u)L_{m-2}(v)+$$
$$\ll 361\text{ terms}\gg-4\epsilon m^2(n+1)A_{l,m,n+1}L_l(u)L_{m+2}(v)]\ L_n(w)=0 \qquad (10)$$

The dependences on v and u are treated similarly, to give:

$$\sum_{n=0}^{\infty}\sum_{m=0}^{\infty}\sum_{l=0}^{\infty}[4\epsilon lA_{l-2,m,n}+\ll 362\text{ terms}\gg+4l^2Z\,A_{l+2,m,n}]\ L_l(u)L_m(v)L_n(w)=0 \qquad (11)$$

Orthogonality of the Laguerre functions gives a 33-term recurrence formula for the $A_{l,m,n}$.

$$4(l+1)(l+2)\{-Z+\epsilon(1+m+n)\}A_{l+2,m,n}+\ll 31\text{ terms}\gg+$$
$$2mn\{1-2Z+\epsilon(2l+n+1)\}A_{l,m-1,n-1}=0 \qquad (12)$$

Let (l_j,m_j,n_j) be the j'th triple in the sequencing (12), where $w_j=l_j+m_j+n_j$ and $j<k$.

$$w_j\leq w_k;\qquad n_j\leq n_k\ \text{ if }\ w_j=w_k;\qquad l_j<l_k\ \text{ if }\ w_j=w_k\ \text{ and }\ n_j=n_k \qquad (13)$$

In symmetric states, $A_{l,m,n}=A_{m,l,n}$, so we write $B_k=A_{l_k,m_k,n_k}$, where $\{l_k,m_k,n_k\}$ is the kth triple in the sequence that also satisfies $l_k\leq m_k$. The restriction $l+m+n<=q$ gives the q'th approximation to wave function and energy. $q=1$ takes the first 10 A's in the sequence (12). These map into $B_1,\ldots,B_7$. The equations formed from (11) for these by setting the $B_k=0, k>7$ require the following determinant in $\xi=Z-\epsilon$ to be zero.

$$\begin{vmatrix}
5-16\xi & -4+4\xi & -6+28\xi & 1 & 2-4\xi & -8\xi & 2-4\xi \\
-4+4\xi & 15-48\xi+24Z & 2+8\xi-12Z & -12+16\xi-8Z & -10+60\xi-24Z & -8\xi+8Z & 0 \\
-6+28\xi & 2+8\xi-12Z & 26-144\xi+32Z & -4\xi+4Z & -12+16\xi & -12+104\xi-16Z & -14+72\xi-28Z \\
1 & -12+16\xi-8Z & -4\xi+4Z & 31-90\xi+64Z & 4+20\xi-28Z & 0 & 0 \\
2-4\xi & -10+60\xi-24Z & -12+16\xi & 4+20\xi-28Z & 54-336\xi+192Z & 4+32\xi-40Z & 2+4\xi-8Z \\
-8\xi & -8\xi+8Z & -12+104\xi-16Z & 0 & 4+32\xi-40Z & 34-320\xi+96Z & 8-24\xi+8Z \\
2-4\xi & 0 & -14+72\xi-28Z & 0 & 2+4\xi-8Z & 8-24\xi+8Z & 25-208\xi+104Z
\end{vmatrix} \qquad (14)$$

In terms of the normalizing factor $\mathcal{N}$, the first approximation to the wave function is:

$$\psi_1=\frac{\mathrm{e}^{-(u+v+w)/2}}{\mathcal{N}_1}[B_1L_0(u)L_0(v)L_0(w)+B_2L_0(u)L_0(v)L_1(w)+$$
$$B_3\{L_1(u)L_0(v)L_0(w)+L_0(u)L_1(v)L_0(w)\}+\ll 3\text{ terms}\gg+B_7L_1(u)L_1(v)L_0(w)] \qquad (15)$$

Expansion of the determinant followed by some simple rearrangement leads to:

$$\xi=0.3125+\frac{1}{Z}(0.808039-7.07288\xi+14.0571\xi^2)+\ll 4\text{ terms}\gg+$$
$$\frac{1}{Z^6}(0.000735782+\ll 6\text{ terms}\gg-100.288\xi^7) \qquad (16)$$

For helium, $Z=2$, and numerical solution gives $\xi=0.2961$ for the lowest root, whence ϵ. Given ϵ, the B_j are determined relative to an arbitrary scaling factor. B_1 is set to 1, and the equations that led to 13 are solved numerically. Hence:

$$B_1=1,\quad B_2=0.03859,\quad B_3=-0.04876,\quad B_4=0.002969,\quad \ldots \qquad (17)$$

We replace the Laguerre functions in (14) by explicit polynomials in (u,v,w), and replace these coordinates by (r_1,r_2,r_{12}), by reference to (1). Hence the wave function in the form:

$$\psi_1=\frac{\mathrm{e}^{\epsilon(-r_1-r_2)}}{\mathcal{N}_1}[d_1+d_2(r_1+r_2)+d_3(r_1^2+r_2^2)+d_4r_1r_2+r_{12}\{d_5+d_6(r_1+r_2)\}+d_7r_{12}^2] \qquad (18)$$

where the d_i are linear combinations of the Bs.

$$d_1=B_1+B_2+2B_3+B_4+2B_5+2B_6+B_7,\quad d_2=-2\epsilon(B_2+2B_4+2B_5),\quad \ldots \qquad (19)$$

The normalizing factor is found from the volume integral $\int\psi^2d\tau=1$, using:

$$\int f\,d\tau=\frac{\pi^2}{32\epsilon^6}\int_{u=0}^{\infty}\int_{v=0}^{\infty}\int_{w=0}^{\infty}(u+v)(2u+w)(2v+w)f\,du\,dv\,dw \qquad (20)$$

whence

$$\mathcal{N}_1=\frac{\pi}{2\epsilon^3}(4B_1^2-5B_1B_2+\ll 22\text{ terms}\gg+52B_6B_7+55B_7^2) \qquad (21)$$

The radial density distribution is found from:

$$\rho(r_1)=8\pi^2\left(\int_{r_2=0}^{r_1}\int_{r_{12}=r_1-r_2}^{r_1+r_2}\psi^2r_1r_2r_{12}\,dr_2\,dr_{12}+\int_{r_2=r_1}^{\infty}\int_{r_{12}=r_2-r_1}^{r_1+r_2}\psi^2r_1r_2r_{12}\,dr_2\,dr_{12}\right) \qquad (22)$$

Some formulas for reference

Table 1. $S_q = \sum_{k=1}^{n} k^q$

q	S_q	q	S_q
1	$(n^2+n)/2$	2	$(2n^3+3n^2+n)/6$
3	$(n^4+2n^3+n^2)/4$	4	$(6n^5+15n^4+10n^3-n)/30$
5	$(2n^6+6n^5+5n^4-n^2)/12$	6	$(6n^7+21n^6+21n^5-7n^3+n)/42$
7	$(3n^8+12n^7+14n^6-7n^4+2n^2)/24$	8	$(10n^9+45n^8+60n^7-42n^5+20n^3-3n)/90$
9	$(2n^{10}+10n^9+15n^8-14n^6+10n^4-3n^2)/20$		
10	$(6n^{11}+33n^{10}+55n^9-66n^7+66n^5-33n^3+5n)/66$		

Problem sets and worked solutions

Fold this worksheet, factor the expressions and check your answers

1. $54n^2+3fn-77f^2$ _____ $(11f+9n)(6n-7f)$
2. $70m^2-83mu+18u^2$ _____ $(10m-9u)(7m-2u)$
3. $30i^2+59ip-56p^2$ _____ $(10i-7p)(3i+8p)$
4. $35p^2+34kp-33k^2$ _____ $(11k+7p)(5p-3k)$

⋮ ⋮ ⋮

Consider the thermal decomposition of a sample of H_2O_2. The temperature is 22° C. The pressure is 773 torr. The volume of gaseous product is 6.01 liter. Calculate the mass of the sample.

Answer : moles of gas $= \dfrac{\text{pressure} \times \text{volume}}{\text{gas constant} \times \text{temperature Kelvin}} =$

$= \dfrac{(773 \text{ torr}) \times (6.01 \text{ liter})}{(62.36 \text{ liter torr/ deg mol}) \times (295 \text{ deg})} = 0.253 \text{ mol};$

Hence : mass of sample $= \dfrac{\text{molecular mass} \times \text{number of moles of gas}}{\text{mole factor}} =$

$= \dfrac{(34 \text{ gm}) \times (0.253)}{(0.5)} = 17.2 \text{ gm}.$

0.103 mol of $CaCO_3$ undergoes thermal decomposition. The pressure is 795 torr. The temperature is 24° C. Compute the volume of gaseous product.

Answer : moles of gas $=$ mole factor $\times$ moles in sample $=$

$= (1) \times (0.103 \text{ mol}) = 0.103 \text{ mol};$

Hence : volume $= \dfrac{\text{gas constant} \times \text{moles of gas} \times \text{temperature Kelvin}}{\text{pressure}} =$

$= \dfrac{(62.36 \text{ liter torr/ deg mol}) \times (0.103 \text{ mol}) \times (297 \text{ deg})}{(795 \text{ torr})} = 2.4 \text{ liter}.$

This ruled table was produced in an experimental reconstruction of portions of the reference work commonly known by the names of the authors Gradshteyn and Ryzhik. The entire first section of indefinite algebraic integrals has been derived anew — many of the citations in the monograph are unhelpful or inaccessible. The process of mechanization provided several useful prototype derivations and new insights of wider application.

The factoring example, like many others in [5] was produced by working back from the solutions. These were formed by random choice of the letters used to name the variables. The coefficients also were random, within a limited range, and rejected if the expanded expression would contain coefficients outside a particular range.

The gas law example is part of a much larger set. The procedure accepted a sequence of n-tuples that specified the property to be found (*e.g.* pressure, number of moles), the compound undergoing decomposition, the units, the values of the given variables, within acceptable ranges, the sentence order, and certain words and phrases. This work is in direct line with an earlier project of the author sponsored by the NSF under their CAUSE initiative some years ago.

The envelope examples

• Example 1: Plot the family of lines $y = m^4 + 2mx$ and its envelope. The canonical and derivative equations are, respectively,

$$y - m^4 - 2mx = 0 \qquad (1.1) \qquad\qquad 2(2m^3 + x) = 0 \qquad (1.2)$$

The parametric form of the discriminant is

$$x = -2m^3 \qquad (1.3) \qquad\qquad y = -3m^4 \qquad (1.4)$$

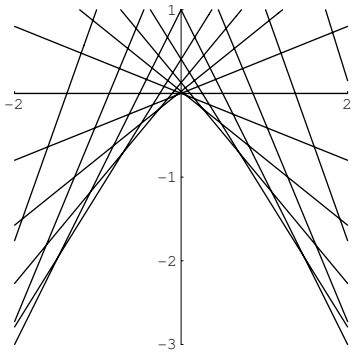


Figure 1a

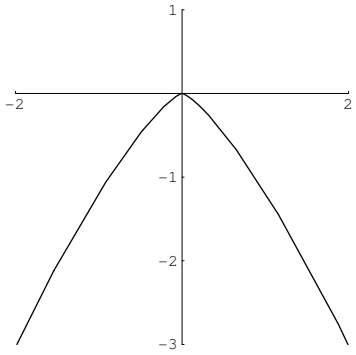


Figure 1b

The existence of an envelope is shown by inspection of

$$F_x = -2m, \;\; F_y = 1, \;\; F_{mx} = -2, \;\; F_{my} = 0 \qquad (1.5)$$

$$F_{mm} = -12m^2, \;\; \begin{vmatrix} F_x & F_y \\ F_{kx} & F_{ky} \end{vmatrix} = \begin{vmatrix} -2m & 1 \\ -2 & 0 \end{vmatrix} = 2 \qquad (1.6)$$

• Example 4: Plot the family of parabolas $y^2 = a(x - a)$ and its envelope. The canonical and derivative equations are, respectively,

$$a(a - x) + y^2 = 0 \qquad (4.1) \qquad\qquad -2a + x = 0 \qquad (4.2)$$

The direct form of the discriminant has the two solutions

$$y = -\frac{x}{2} \qquad (4.3a) \qquad\qquad y = \frac{x}{2} \qquad (4.3b)$$

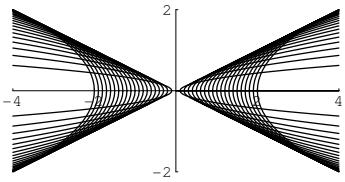


Figure 4a

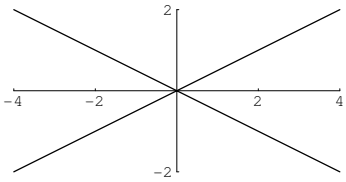


Figure 4b

The existence of an envelope is shown by inspection of

$$F_x = -a, \;\; F_y = 2y, \;\; F_{ax} = -1, \;\; F_{ay} = 0 \qquad (4.4)$$

$$F_{aa} = 2, \;\; \begin{vmatrix} F_x & F_y \\ F_{kx} & F_{ky} \end{vmatrix} = \begin{vmatrix} -a & 2y \\ -1 & 0 \end{vmatrix} = 2y \qquad (4.5)$$

Envelopes have long been of interest in popular mathematics and education. mathscape was used to produce graphically illustrated worked solutions to the exercises on this topic in a problem book that was widely used in the former Soviet Union. Each example begins with the generic equation for a family of curves. The problem is to determine whether the family has an envelope and, if it does, to find the equation and to plot it. The first step finds the "discriminant equation." Sometimes, this is best found in direct form, in other instances parametrically. It may have one or more solutions. Direct, implicit or parametric plotting may be optimal for the envelope.

The process was encapsulated in a single, heavily conditionalized control file. The data for each example consisted of the noun that identified the members of the family (e.g., "line", "curve"), the generic equation, and the choices needed to navigate the alternative paths.

The work was done by Artur v. Solecki, as an undergraduate project in a computer graphics course that the author taught.

$$1.\quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} . \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} \boxed{1\times 5+2\times 7} & \boxed{1\times 6+2\times 8} \\ \boxed{3\times 5+4\times 7} & \boxed{3\times 6+4\times 8} \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$2.\quad \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix} . \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} \boxed{3\times 5+4\times 7} & \boxed{3\times 6+4\times 8} \\ \boxed{1\times 5+2\times 7} & \boxed{1\times 6+2\times 8} \end{pmatrix} = \begin{pmatrix} 43 & 50 \\ 19 & 22 \end{pmatrix}$$

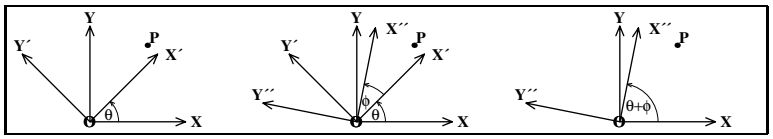
$$3.\quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} . \begin{pmatrix} 6 & 5 \\ 8 & 7 \end{pmatrix} = \begin{pmatrix} \boxed{1\times 6+2\times 8} & \boxed{1\times 5+2\times 7} \\ \boxed{3\times 6+4\times 8} & \boxed{3\times 5+4\times 7} \end{pmatrix} = \begin{pmatrix} 22 & 19 \\ 50 & 43 \end{pmatrix}$$

$$4.\quad \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix} . \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} \boxed{5\times 1+7\times 2} & \boxed{5\times 3+7\times 4} \\ \boxed{6\times 1+8\times 2} & \boxed{6\times 3+8\times 4} \end{pmatrix} = \begin{pmatrix} 19 & 43 \\ 22 & 50 \end{pmatrix}$$

Compare the starting matrices and the results in examples 1 and 2.
Make the corresponding comparisons for examples 1 and 3, and for examples 1 and 4.

This depiction of a matrix multiplication illustrates the use of fonts to show "where things come from" in a derivation. The entire set of four traced multiplications is parameterized on the eight starting matrix elements, enabling the rapid production of further examples of numerical and symbolic matrix operations. In teaching a course on mathematics for humanists some years ago, the author found it helpful to use worked examples of two-step linear transformations, expressed in terms of verbal matrix elements, e.g., the number of locomotives (coaches) per starter (advanced) train set, and the number of nuts (bolts) per locomotive (coach), and the corresponding product elements.

The axis diagrams are part of an explanation of rotation matrix multiplication, that uses symbolic calculation to generate the associated equations. Diagrams and associated matrix equations are used, too, in the connectivity matrix treatment of n-step path counts in a directed graph.



Rationalize the denominator in:

$$\frac{\sqrt{x-y}-\sqrt{x+y}}{\sqrt{x-y}+\sqrt{x+y}} \tag{1}$$

Multiply the numerator and the denominator by the numerator, and expand.

$$\frac{(\sqrt{x-y})^2-2\sqrt{x-y}\sqrt{x+y}+(\sqrt{x+y})^2}{(\sqrt{x-y})^2-(\sqrt{x+y})^2} \tag{2}$$

Use $(\sqrt{a})^2 = a$ and $\sqrt{a}\sqrt{b} = \sqrt{ab}$.

$$-\frac{2x-2\sqrt{(x-y)(x+y)}}{2y} \tag{3}$$

Simplify:

$$\frac{\sqrt{x^2-y^2}-x}{y} \tag{4}$$

Consider the geometric series:

$$S(n) = \sum_{i=0}^{n} x^i \tag{1}$$

Multiply throughout by x and restructure the right hand side.

$$xS(n) = \sum_{i=0}^{n} x^{i+1} = \sum_{i=1}^{n+1} x^i = \sum_{i=0}^{n} x^i + x^{n+1} - 1 \tag{2}$$

Subtract (2) from (1).

$$S(n) - xS(n) = 1 - x^{n+1} \tag{3}$$

Solve for $S(n)$.

$$S(n) = \frac{1-x^{n+1}}{1-x} \tag{4}$$

The next few examples illustrate different styles of discourse. The displays may be expressions or statements (in mathematical, not Mathematica, terminology). They may be joined by text or relationship symbols, such as = or >, or by arrows.

In the rationalization example, the identities embedded in the explanatory sentence are applied mechanically, as an example of the avoidance of possibly inconsistent results and narrative.

In the geometric series example, the referencing between equations also is performed mechanically by mention of the tag. This uses the implied rule formation feature of mathscape.

(1) Definition : $A \subseteq B$ iff $x \in A$ implies $x \in B$.

(2) Suppose $A \subseteq B$ and $B \subseteq C$.

(3) Then $x \in A$ implies $x \in B$ and $x \in B$ implies $x \in C$.

(4) Hence $x \in A$ implies $x \in C$.

(5) Consequently $A \subseteq C$.

{1}	(1) $A \Rightarrow B \vee C$	P
{2}	(2) $B \Rightarrow -A$	P
{3}	(3) $D \Rightarrow -C$	P
{4}	(4) A	P
{1, 4}	(5) $B \vee C$	*law of detachment*$(4,1)$
{2, 4}	(6) $-B$	*modus tollendo tollens extended*$(4,2)$
{1, 2, 4}	(7) C	*modus tollendo ponens*$(6,5)$
{1, 2, 3, 4}	(8) $-D$	*modus tollendo tollens extended*$(7,3)$
{1, 2, 3, 4}	(9) $A \Rightarrow -D$	*c.p.*$(4,8)$

The successive examples of simple algebraic operations in the display below this paragraph were formed by a single assignment to `prep` followed by the pairs `{Expand, (1+x)^2}`, ...

1. *Expand* : $(1+x)^2$. *Answer* : $1+2x+x^2$.
2. *Factor* : x^2-y^2. *Answer* : $(x-y)(x+y)$.
3. *Cancel* : $\dfrac{x^2+2xy+y^2}{x^2-y^2}$. *Answer* : $\dfrac{x+y}{x-y}$.

The production of the proof of transitivity of the $\subseteq$ operator (above left) involved the conversion of functional expressions to sentence form. Both this example and the logic proof (left) can serve as prototypes for quite large classes of application.

Alignment and tags: some more examples

Items can be labelled collectively and individually. The Legendre functions of degrees 0–3 of the first and second kinds follow.

5.7.1. (a) 1 (b) x

(c) $\dfrac{3x^2-1}{2}$ (d) $\dfrac{5x^3-3x}{2}$

5.7.2. (a) $\dfrac{1}{2}\log\dfrac{1+x}{1-x}$ (b) $-1+\dfrac{x}{2}\log\dfrac{1+x}{1-x}$

(c) $-\dfrac{3x}{2}+\dfrac{3x^2-1}{4}\log\dfrac{1+x}{1-x}$ (d) $\dfrac{-15x^2+4}{6}+\dfrac{5x^3-3x}{4}\log\dfrac{1+x}{1-x}$

The outer horizontal lines are `itemWidth` long and the central line is `runOnGap` long.

$1+x$ (1) $1+2x+x^2$ (2)

$1+3x+3x^2+x^3$ (3) $1+4x+6x^2+4x^4+x^4$ (4)

The centering allows for the tags and the `runOnGap`.

Items can be labelled collectively:

(5.7.1) $\displaystyle\int e^x x^2\,dx \;\to\; e^x x^2 - 2\int e^x x\,dx \;\to\; e^x(-2x+x^2)+2\int e^x\,dx \;\to\; e^x(2-2x+x^2)$

(5.7.2) $\displaystyle\int \cos^2 x\;dx \;\to\; \frac{2x+\sin 2x}{4}$

The examples on the left show grouped items tagged individually and/or collectively, and variations in the tag style. The examples below show multi-expression bracing, and alignment on single and multiple relationship symbols.

$$\left.\begin{aligned}(1+x)^2 &= 1+2x+x^2\\ (1-x)^2 &= 1-2x+x^2\\ (1-x)(1+x) &= 1-x^2\end{aligned}\right\}\quad(1)$$

$$\begin{aligned}(1+x)^2 &= 1+2x+x^2 && (1)\\ 2x^4 &\geq x^4 && (2)\\ 1 &< 2 && (3)\\ a+b &\hookrightarrow c+d && (4)\end{aligned}$$

For $x > 1$,

$$e^{x^2} > e^x > x > \log x$$

For $0 <= \theta <= \pi/2$,

$$0 \leq \sin^2\theta \leq \sin\theta$$

TEX and LATEX on the Web via IBM **techexplorer**

Robert S. Sutor
Interactive Sci. Publishing Group
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598 USA
sutor@us.ibm.com

Samuel S. Dooley
Interactive Sci. Publishing Group
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598 USA
dooley@watson.ibm.com

Abstract

The IBM **techexplorer** Hypermedia Browser™ is an application for the interactive publication of scientific and technical documents. The original project started as an experiment at IBM Research to see how an implementation of a subset of TEX, LATEX, and $\mathcal{AMS}$-LATEX could be extended to support interactive viewing of documents for a computer algebra system. This interactivity is accomplished via support for hypertext, multimedia, user-defined pop-up windows and menus, and a modular architecture that allows connections with other applications and Java applets. IBM's **techexplorer** provides an alternative to HTML-based solutions for presenting scientific and technical documents on the World Wide Web and is being used for scientific journals, and educational courseware and textbooks.

The Intorductory Edition of **techexplorer** operates as a Netscape Navigator Plug-In and is available for several platforms, including Windows 95/NT, Sun Solaris, and IBM AIX. In addition to being able to display full documents using the supported LATEX language, **techexplorer** also implements the new Mathematical Markup Language being prepared by the HTML Working Group of the World Wide Web Consortium. In this paper we will give an overview of **techexplorer** and detail how it can be used to deliver mathematical articles, books and course materials via the World Wide Web. Future directions regarding our plans for opening the architecture of **techexplorer** and how that relates to the authoring of scientific and technical documents for the Internet will also be discussed.

– – ∗ – –

Introduction

The World Wide Web provides one of the greatest opportunities that the publishing industry has seen in this century, as well as one of its most perplexing challenges: how to produce interactive electronic alternatives to printed textbooks and journals that take advantage of the unique characteristics of electronic media and of the Internet in a way that is both intellectually engaging and economically viable. This challenge is even more rewarding in the arena of scientific and technical publishing, where the complexities of mathematical layout and the richness of the information contained in technical documents cause special problems not encountered with other kinds of documents, but also give rise to exciting possibilities for creating truly interactive materials that are useful for distributed and distance learning, interactive courseware, and electronic journals.

However, until recently the publication of documents containing a high degree of technical content on the World Wide Web has been extremely awkward, due to the absense of HTML support for mathematical notation. While TEX and LATEX have become a widely accepted standard for publishing scientific and technical documents, authors and publishers have had no convenient way of electronically disseminating documents written in this form. As a result, we have had to make compromises in various ways as authors to allow our materials to take advantage of the possibilities of the Internet, either by using tools for converting TEX/LATEX markup into HTML, or by using static images (GIF, PDF, etc.) for mathematical notation. Such conversions result in documents of poor visual quality, that fail to adapt well to a wide range of display and printer hardware, and that fail to preserve the rich semantic information present in technical documents.

While the browser development and Internet standards communities have long acknowledged the shortcomings of HTML for the presentation of mathematical notation, early efforts to extend HTML

with additional primitives to address the needs of the technical publishing community (such as HTML 3.0, as well as early versions of HTML Math) have been largely unsuccessful, due to the relatively specialized nature of mathematical notation. The approach now being used in more recent efforts is to have the Internet community support XML as a general extension mechanism for HTML, and allow the technical publishing community to define the Mathematical Markup Language (MathML) as an XML application. This alternative holds greater promise for the future, but it will be some time before this approach can be fully supported by software developers.

The IBM **techexplorer** Hypermedia Browser™ provides an alternative to HTML-based solutions by dynamically rendering a large subset of TEX and LATEX markup without converting the original document source to an alternative markup or binary format. Instead, when **techexplorer** reads a LATEX document, it parses the original document source into an internal object-oriented representation that mirrors the visual structure of the document. Such a representation enables high-quality dynamic rendering at varying screen resolutions and sizes, rapid document reflow and redisplay, and the opportunity to implement various extensions to the language that facilitate the use of hypertext and the inclusion of multimedia content.

This object-oriented approach allows **techexplorer** to support a much richer model of active mathematical content. When a mathematical expression appears in the text of a document, **techexplorer** can represent both the visual appearance of the expression and the underlying semantic mathematical content being presented by the visual notation. This capability allows **techexplorer** to support interactive mathematical manipulations within a document that operate not on the visual appearance of the document, but on the underlying structure of the mathematical expressions it contains. Using this capability, we have used **techexplorer** to develop interactive courseware that supports symbolic problem solving and intelligent graphical exploration, on a platform that allows interactive technical documents to be delivered over the Internet.

Project history

techexplorer is our second generation TEX-based hypertext system. The original application, known as HyperDoc, was developed as a viewer for documents for the computer algebra system now known as Axiom and now distributed by the Numerical Algorithms Group, Ltd. (NAG) Axiom is a sophisticated system for performing mathematical computation that offers two- and three-dimensional graphics, a hypertext help package, and various forms of output, including LATEX and Fortran. HyperDoc was developed in the late 1980s for Unix platforms, and serves as Axiom's hypertext front end for viewing text intermixed with the results of computations. The Axiom/HyperDoc link enabled users to open a workspace by clicking on Axiom input, or start a graphics manager by clicking on a graphic.

The first challenge we faced when extending HyperDoc was that it accepted a non-standard dialect of LATEX, and so our initial efforts were directed toward improving HyperDoc to support standard LATEX. However, HyperDoc still did not render many common TEX/LATEX control sequences, and what is perhaps most surprising, it had very weak support for displaying mathematics. Some early implementation choices made adding such support very difficult and hindered porting HyperDoc to other platforms such as Microsoft Windows. Although HyperDoc had excellent connectivity to Axiom and to the graphics manager, there was no way to update a document in-place with the results of a computation or with a modification to a graph. For these reasons, in 1994 we embarked on developing a completely separate implementation of a large subset of standard TEX and LATEX, with extensions to support the interactive viewing of documents.

That project's efforts resulted in an early version of **techexplorer** (then known as "Saturn") that produced familiar output from TEX/LATEX source. Our approach to orthogonally extending TEX and LATEX with control sequences for hyperlinking, multimedia, and interaction with mathematical software ensured **techexplorer**'s compatibility with TEX and LATEX markup for printed documents. An early version of this standalone edition of **techexplorer** is used as the front end for NAG's Axiom for Windows product.

Our decision to provide a Windows 3.1 and later a Windows 95/NT implementation of **techexplorer** was prompted by the sophisticated set of available tools geared toward rapid `C++` code development, a rich set of user interface components, a robust implementation of interprocess communication via Object Linking and Embedding (OLE), and a potentially large user community. Our aim was to leverage these technologies and quickly develop a framework for rendering and interacting with mathematical documents.

It became evident in early 1996 that we could augment our core technology to deliver interactive scientific and technical documents over the World

Wide Web via the Netscape Navigator plug-in interface. The first version for Windows 95 was made publicly and freely available in May 1996. This "Introductory Edition" of the **techexplorer** Plug-In allows authors and publishers to effectively expand the reach of their articles, books, and journals by making them available on the Internet. As the community of **techexplorer** users began to grow, we realized that a UNIX edition of the plug-in was a high priority for our colleagues in the scientific community. In September 1997, we released our first "Preview Release" of **techexplorer** on a UNIX platform on IBM alphaWorks.

In parallel to the work on the core TEX/LATEX technology, we realized early on that the dynamic rendering and object-oriented representation used in **techexplorer** could provide an excellent framework for an interface for active mathematical documents such as interactive textbooks. Combined with the group's earlier expertise in computer algebra system development, we felt that the creation of an electronic textbook for linear algebra would be a natural application of the **techexplorer** technology. As a result, a stand-alone version of **techexplorer** was used as the framework for an electronic version of the textbook *Linear Functions and Matrix Theory* by Bill Jacob, that will be appearing as the first volume of the forthcoming Springer Interactive CourseWare Series. This interactive textbook combines **techexplorer**'s dynamic document model with the powerful symbolic computation facilities of Axiom and with a collection of Java graphical exploration tools to allow a reader to interact with the course material in a number of novel ways.

Today, the Interactive Scientific Publishing Research Group at IBM Research continues to distribute the **techexplorer** Plug-In, Introductory Edition for Windows 95/NT, IBM AIX 4.1 and SUN Solaris 2.5, with more platforms planned for the future. The **techexplorer** product line will continue to evolve as we create new tools and technologies for the Internet delivery of scientific and technical journals, reports, textbooks, and courseware.

techexplorer overview

In creating **techexplorer**, we set out to implement a majority of the standard LATEX control sequences and environments, as well as a substantial collection of the commonly used features from plain TEX. Support for these features, especially during the early development stages of the introductory edition of **techexplorer**, has traditionally been user and application driven. At the time of this writing, virtually all of the standard LATEX commands are available; the **techexplorer** user guide lists the LATEX and TEX commands that are supported, as well as the **techexplorer** extensions that have been added. In addition to the support for standard LATEX, full support for $\mathcal{AMS}$-LATEX is planned for the near future. Parsers for subsets of SGML, XML, and MathML presentation tags have also been implemented that produce the same object-oriented representation used by the LATEX parser. As a result, markup written in these languages can be embedded in LATEX documents, and vice versa, and rendered using **techexplorer**.

When **techexplorer** parses a document, an internal tree structure of objects is created that represents the document contents. Thus when the user clicks on the display screen, **techexplorer** has enough information to identify the object in the structure hierarchy under the position of the mouse cursor. This means, in particular, that **techexplorer** can provide hypertext links or maintain status messages that are updated as the cursor passes over different objects in the document. Different flavors of **techexplorer** links can:

- navigate to another location, either in the current document or in a different document, possibly in a different frame;
- start an application;
- play an audio or video clip from a URL;
- pop up various kinds of dialog boxes for user input;
- display fully-formatted text in a pop-up window;
- display one or the other of two expressions; or
- send input to another application, and place any output generated as a result into the current document.

Documents delivered over the World Wide Web are easier to navigate if they are broken up into reasonably sized sections with a rich collection of hyperlinks. For many documents, this creates a natural tree hierarchy. The commands `\aboveTopic`, `\previousTopic`, and `\nextTopic` allow the reader to move up, left or right, respectively, in this hierarchy, as it is defined by the document author. When these commands are defined in a document, the default document context menu (produced by clicking the right mouse button) allows the reader to jump quickly to the corresponding sections. Commands on the document context menu also allow the reader to navigate forward and backward in the dynamic sequence of sections visited in the current session. All of these commands are also available from the **techexplorer** toolbar.

Color support is very important for high quality on-screen display and so **techexplorer** implements the commands `\color`, `\textcolor`, `\colorbox`, `\fcolorbox`, `\rgb`, `\pagecolor`, as well as the **techexplorer** extension `\colorbuttonbox`. We also provide the `\includegraphics` command to embed images in either GIF or JPEG format in a document. The `\backgroundimage` command is a **techexplorer** extension that allows the author to set the image displayed behind the text in a window.

Pop-up menus, also known as context menus, are very useful for making selections. Within an electronic version of a large textbook, for example, it is relatively straightforward to define context menus that can be used to easily navigate from section to section with the book, or within smaller sub-sections of the current section. As an example, here is a menu definition from section 3.2 of *Linear Functions and Matrix Theory*:

```
\newmenu{theorem-menu-3.2}{
    \labelLink{theorem-3}{Theorem 3}
}
\newmenu{section-menu-3.2}{
    \labelLink{sec-3.2}{Section 3.2}
    \hrule
    \usemenu{theorem-menu-3.2}{Theorems}
    \hrule
    \labelLink{sec-3.2.1}{Elementary Operations}
    \labelLink{sec-3.2.2}{The Augmented Matrix of a System}
    \labelLink{sec-3.2.3}{Equivalent Systems of Equations}
    \labelLink{sec-3.2.4}{Gaussian Elimination}
    \labelLink{sec-3.2.5}{Echelon Form Systems}
    \labelLink{sec-3.2.6}{Solutions to Echelon Form Systems}
    \labelLink{sec-3.2.7}{Row-Echelon Matrices}
    \labelLink{sec-3.2.8}{Two Remarks About Parameters}
    \hrule
    \labelLink{section-3.2.problems}{Problems for 3.2}
    \hrule
    \docLink{lfmttoc.tex}{Book Contents}
}
```

The `\hrule` commands put separator lines in the menu. Most of the menu selections are hypertext links within the section, but there is one submenu that lists the theorems in the section. (In this case, there is only one theorem.) The `\usemenu` command is used in menus to create submenus as above, but it is also used within the text to associate a menu with a particular piece of text.

An interesting and sometimes controversial subject is how **techexplorer** deals with fonts. On Windows 95 and Windows NT, **techexplorer** uses TrueType fonts directly, while under UNIX, **techexplorer** uses PostScript fonts. METAFONT fonts are not currently supported. The user can select any TrueType or PostScript font under Windows or UNIX, respectively, for use with `\rm`, `\bf`, `\it`, `\tt`, etc. Special symbols are obtained from various font collections that may be available from other sources, such as the Monotype Math fonts shipped with Lotus SmartSuite, the WordPerfect math fonts, or the Lucida fonts sold by Microsoft. For UNIX we derived several symbol fonts from the Computer Modern and $\mathcal{AMS}$ Symbol fonts created by BlueSky Research and Y&Y and placed in the public domain under the auspices of the American Mathematical Society. In 1998, we plan to release TrueType versions of these derived symbol fonts for use under Windows. Eventually we expect to open the font model to allow authors to map a symbol to an arbitrary character in a font of their choice.

Authoring

The Introductory Edition of the **techexplorer** Hypermedia Browser™ is available for free download via the IBM **techexplorer** web site.[1] This version contains all the core TEX/LATEX parsing and rendering features, and can be used to rapidly distribute scientific and technical documents on the World Wide Web. **techexplorer**'s compatibility with standard LATEX means that these documents can be developed in the most natural markup language for mathematical notation and disseminated directly without major modification and without translation into another format. The **techexplorer** extensions to LATEX provide a gentle upgrade path that allows authors to incrementally include hypertext and multimedia extensions to their documents as time and resources allow.

Several options are available for increasing the interactivity of electronic versions of LATEX documents using the **techexplorer** extensions. A **techexplorer** document can be used as an interface to other programs. Depending on the nature of the application, the integration between the document and the application can range from a model where the document is used merely to launch the external application, as in the Introductory Edition, to one where there is a much higher degree of active mathematical information being communicated from the document to the application and vice-versa, as in the upcoming Springer Interactive CourseWare.

For those authors comfortable with Java programming, Java applets can also be incorporated into LATEX documents using the **techexplorer** extensions, either by invoking stand-alone Java programs using the more general application link, or by using the **techexplorer** extension command `\javaLink` to invoke a Java method directly. In the future, we plan to expose more of the underlying **techexplorer** document model to provide a consistent interface for external Java methods to manipulate and script **techexplorer** document contents.

[1] `www.software.ibm.com/techexplorer`

Conclusions

Our primary goal for the **techexplorer** technology is to create a set of publicly available interfaces that allow **techexplorer** to serve as the "glue" for developing novel interactive scientific and technical documents. By using **techexplorer**, such documents will be able to leverage Internet programming languages, as well as specialized programs from a wide-range of scientific software applications.

Clearly, with the continued growth of the World Wide Web and increased investments in scientific and educational content, traditional scientific and technical markup languages will play a central role in online dissemination. The deployment of interactive scientific and technical documents using enhanced versions of these languages will be central to the success of the next generation of technical publishing. We in the Interactive Scientific Publishing group at IBM plan to continue enhancing the **techexplorer** products to support and to define professional quality scientific online publishing in this next generation.

Real Life LaTeX: Adventures of a TeX Consultant

Amy Hendrickson
TeXnology Inc.
57 Longwood Avenue
Brookline, MA 02146
USA
amyh@ai.mit.edu

Fortunately, the life of a LaTeX consultant can be varied and the activities diverse. In my fourteen years working with Donald Knuth's wonderful language, I've spent time teaching LaTeX, writing special-purpose macro packages for, among other things, database publishing, tables that continue for hundreds of pages, training slides, software documentation, and PDF production, in addition to my major activity — writing and supporting multiuser macro packages for publishing companies. The programming capabilities of the language are immense, and it has been fun exploring a tiny part of its possible applications.

In this paper I'd like to share some observations, especially in the areas of designing and supporting multiuser macro packages, the use of PostScript in design, and some of the capabilities of LaTeX as a generator of PDF.

Preparing and Supporting Multiuser Macro Packages

The first multiuser book macro packages that I authored were written in the early 1980s; my first journal macro package was the original version of RevTeX, a widely distributed macro set used by the American Physical Society, which I wrote in the late 1980s. Since then I've written *many* more, and am currently supporting more than thirty journal styles and four book styles that I've written for three different publishing companies. Here are some concepts I've learned in the process.

Designing the Macro Package. First of all, conceptually, there are some critical differences between preparing a macro package to be used once, and one that is to be used by many people over a series of years. Planning ahead is crucial for the multiuser package, which must be both flexible and inclusive, as well as matching the specifications of the publishing company for appearance and functionality, since it will:

- Be used by many authors on many platforms, and even in many countries.
  - Must be flexible enough to accomodate different versions of LaTeX and differing PostScript font naming conventions.
  - Must include capability for all ordinary LaTeX commands since some author will want to use one of them.
  - Must be as easy to use and document as possible.
- Must be easy as possible to change and support.

A nontrivial set of requirements!

Desirable attributes. In addition, there are other considerations which may not be ironclad requirements yet which make the macro set useful and desirable:

- Keep commands similar to the ones used by standard LaTeX. This will mean less documentation, and fewer problems for authors.
- Include commands that are not found in the general LaTeX distribution but which are generally useful, such as lettered equations, continued captions, lettered captions, and other convenient additions or alterations to the general distribution form of LaTeX.

Process rather than product. Conceptualizing the package as a process rather than a product is helpful since, in reality, the authors or publishing company will very likely want to change the style slightly or will request additional features.

To do this, we want, first and foremost, to keep the code as simple and clean as possible. Comments should be added where necessary, to help understand why a command was written in a particular way, to make it easier to make changes to it later.

Organizing the main macro set into parts according to function, and listing the various parts may take more time when writing the code but in the long run it will make it easier to find the part that needs to be changed. Examples of this are a part for theorem environments, a part for specific font calls, or a part for equations, each designated

and numbered, with a numbered list near the top of the file to make it easier to find the particular part.

Another way I have found to simplify the package and its maintainance is to have a single main macro file which will work with either LaTeX2.09 or LaTeX 2ε. This means that when a change needs to be made, it can be made to one file, which can then be copied and distributed as both `filename.sty` and `filename.cls`. The contents of each file are identical, but the filename ending will satisfy the requirements of LaTeX2.09, which is looking for a `.sty` file; and LaTeX 2ε, which is looking for a `.cls` file. Here is the switch which I build into the main macro file:

```
\newif\ifll
\expandafter\ifx\csname LaTeXe\endcsname\relax
  %  We see that LaTeXe has not been
  %  defined so LaTeX2.09 is being used
\else
  % LaTeX2e is defined, so set ll true,
  % LaTeX2e is being used.
\global\lltrue\fi
```

This means that we can test to see if the file is being used with LaTeX2.09 or LaTeX 2ε, and make definitions in those places where the conventions for the two forms of LaTeX diverge. For instance, when setting font family sizes:

```
\ifll
  %% Provide font family in LaTeX2e form:
\renewcommand{\normalsize}{%
   \@setfontsize\normalsize\@xpt\@xiipt
   \abovedisplayskip 10\p@
....
\else
  %% Provide font family in LaTeX2.09 form:
\gdef\@normalsize{%
\@setsize\normalsize{12pt}\xpt\@xpt
...
\fi
```

Another example shows how options may be used, whether the author is using LaTeX2.09 or LaTeX 2ε:

```
\ifll \let\dooptions\ProcessOptions
\else
\let\dooptions\@options\fi
\dooptions
```

There are many parts of the code where this switch is not necessary, but for those parts where it is, this branching innovation definitely makes maintaining and redistributing the macro package easier.

Making a Flexible PostScript Font File

It is a major nuisance that PostScript font names are not identical across TeX implementations. Karl Berry's naming system is helpful but, unfortunately, it isn't universally used. So, the best solution I've found is to

1. Have a separate PostScript font file that can be used for final production but doesn't need to be used by the author who is not willing to go to the trouble of customizing it. The document will then be printed in ComputerModern for the author, but translated to PostScript in the final production process.
2. For those authors willing to modify the PostScript font file, make it as easy as possible to do so.

Near the top of the PostScript font file the author will read instructions and then see the font names that need to be changed:

```
% You may need to rename these fonts to match
% the names of the .tfm files on your system.
% If you look at the directory where the .tfm
% files are stored you should be able to make
% the appropriate substitution.
% Some TeX implementations, such as TeXtures,
% will show you the available fonts when you
% click on the correct menu item.
%
% You may write in the name your system uses
% if you don't find it already written below.
%
% Change the definitions below,
% if necessary ====>

% Times-Roman
%% the Berry names:
\def\timesroman{ptmr}
\def\timesbold{ptmb}
\def\timesitalic{ptmri}
\def\timesbolditalic{ptmbi}

%% Another possibility:
%\def\timesroman{Times}
%\def\timesbold{TimesB}
%\def\timesitalic{TimesI}
%\def\timesbolditalic{TimesBI}
...
(Similar for Helvetica and Courier,
or other special font names)
...
%% <==== End of changes needed.
%% Please do not make changes below this point.
%% !!!!!!!!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The authors should not have too difficult a time making this modification. We can then use the definition later in the file, after adding `\space` to the end of the font definition:

```
%% Times-Roman
\xdef\timesroman{\timesroman\space}
\xdef\timesbold{\timesbold\space}
... and similar xdef for other fonts names
```

And then we can use them for all the special use fonts that are necessary, without the author having to be at all aware of these commands:

```
\font\titlefont= \helvetica at 16pt
\font\titlethanksfont=\helvetica at 8pt
\font\cccfont=\timesroman at 7pt
\font\subtitlefont= \helvetica at 12pt
\font\specialsectionfont= \universebold at 18pt
\font\affilfont=\timesitalic at 8pt
\font\emailfont=\timesroman at 8pt
\font\communicatedfont=\timesitalic at 8pt
...
```

Macro Package Distribution

Perhaps this is obvious, but the macro packages are typically distributed from an ftp or Web site. Authors are directed to a site by their publishing company and then download the files. A `readme.txt` file can explain the function of each of the files. This system has many advantages, including the fact that the macro set can easily be changed and a new set of macros or documentation dropped into the ftp or Web site. The authors can be instructed to download the files at the time that they do their book or article so that they are sure to have the current versions.

Supporting multiuser macro packages

The complete macro package typically will include a sample file demonstrating every command that is unique to the package, and options which the user may have, as well as a template file with the commands listed in correct order so that the user may copy it and fill in the arguments to at least start his/her paper or book. The final set of files, which are very important to the success of the package, are the documentation files.

Documentation. Frankly, I don't like to read documentation, and I bet you don't either. However, we need to be able to get the information somehow, or transmit the information if we are writing a macro package.

My method, which I hope is helpful to authors, is to provide many examples of code and results: *show* rather than *tell*. I believe that this makes it easy for the author to see what command to use, by comparing their needs to the examples of typeset text, and then examining the code needed to produce that text. The author downloads the documentation file, runs LaTeX on it, and can print it on their own printer.

Another helpful technique is to provide the documentation file in PDF form. Since one of the main problems is getting people to read the documentation, having it presented in attractive PDF form with color and hypertext-linked table of contents, bookmarks, and index, helps authors get started using the package. They may view the PDF file before they have figured out what the various parts of the package are used for, and even, perhaps, before they have figured out how to run LaTeX on the `.tex` form of the documentation. The PDF file can sit on the publisher's Web site, and the author can read it with a Acrobat Reader enabled browser program.

Offering Author Support. Authors of the complex technical material that is usually typeset with LaTeX are undoubtedly very smart, but not necessarily very familiar with LaTeX. Often they are motivated to use it for their book or article, and are quite reassured to know that they can ask a question or modify the macro set to their liking. That is one reason for offering TeXnical support.

Another is that authors may be stuck on one small problem, which they can work out easily with a little help. Typical of this kind of problem is the author who can't figure out how to get the PostScript font file to work on his system. Another very common sticking place is the use of BibTeX, which is often troublesome, but yields with the help of a few suggestions.

Some authors totally refuse to read the documentation. This is aggravating to the person doing support, but the author usually can be directed politely into performing this reasonable task.

Many authors want additional capabilities. If you have not made all the normal LaTeX commands available, you will very likely hear about it, and have requests to make a command like, for instance, `\thanks{}` work in all kinds of unlikely places.

Many also want some new environment or feature. If your contact at the publishing company thinks that it is worthwhile to provide this new capability for the author, then a decision needs to be made if this would be generally useful. If so, add it to the general macro package and to the documentation; otherwise make a special version of the macro file for that particular author.

Finally, sometimes an author may discover a bug in the macros or documentation. Of course, we try our best to avoid this, but it does happen. In this case, we must change the macro file and/or documentation, and drop it back onto the ftp site so that subsequent users don't experience the same problem.

A Plea For Good Design

Many books and articles done in LaTeX use, to be charitable, a timid design. Some publishing companies distribute books done with the standard distribution LaTeX book style. Anyone who values handsome typesetting and understands the capabilities of LaTeX will find these books painful to behold, knowing that they are totally unnecessary.

No excuse for the techie look! First of all, even if the author submits his book in the default LaTeX book style, the publishing company can supply a macro file which will reinterpret the marked up commands, re-run LaTeX on the file, and, with minimum effort, produce a book with a handsome, professional appearance. Second, as far as I am aware, there are no limitations in implementing *any design* when using the combination of LaTeX and PostScript.

TeXnical Capabilities: Using PostScript with LaTeX

The possibility of combining LaTeX and PostScript code in the same macro package opens up many more options for the book designer who, without the knowledge of this potential, might be much more conservative in their design choices.

How it is done. Since we usually print books and journal articles done with LaTeX by converting them to PostScript with a driver program, we can also include raw PostScript commands in the macro file, which can then be passed, unchanged, by the driver program to the final PostScript file. As well as allowing us to add PostScript graphic effects to a macro file, there is also the capability of writing a macro which will include PostScript code which may be altered according to arguments given to the macro.

Here is a rather trivial example, but it demonstrates the principle of using LaTeX information to produce PostScript code. Once you understand that this will work you might imagine many other uses for what is essentially building PostScript code on the fly.

A LaTeX-PostScript macro can be written to position a PostScript grey or colored screen behind a particular area of text. The text is picked up as a macro argument, set in a box to be measured, and the results passed to the PostScript code, which will form a screen of the correct size, which can then be positioned underneath the given text.

First, a definition using PostScript code, designed to be used within another LaTeX macro:

```
\def\printbluescreen#1#2{%
\hbox to\hsize{\vbox to#1pt{\vss
\special{language "PS", literal
"/ChartCheckPoint save def
newpath
 0 0 moveto
 0 #1 rlineto %up
 #2 0 rlineto %over
 0 -#1 rlineto %down
closepath
0.8 0.99 0.99 setrgbcolor %% lt blue
fill
ChartCheckPoint restore
"}% end special
}}}
```

`\printbluescreen` is used in the second part of a two-part macro: the first part begins a box and the second part ends the box. This gives us a box containing the text found between the two macros which we can then measure. The results can be used as the first argument of `\printbluescreen`:

```
....
\printbluescreen{\the\boxht}{\the\pagewidth}
....
```

where `\boxht` is a manipulated version of the height of the test box, and `\pagewidth` is a manipulated version to the width of the text.[1] Each time the macro is used, a new dimension for the `\boxht` may be used, changing the PostScript commands to exactly fit the space behind the given text, in effect making PostScript code on the fly.

We can also have a normal LaTeX macro call, something like `\chapter{}`, for instance, and produce a graphic effect written in PostScript, when the macro for chapter titleblocks includes raw PostScript code that can be altered depending on the argument given to `\chapter{}`.

One of the most interesting designs I've implemented was for documentation of toolbox software packages published by The MathWorks. It had normal chapter titles but also a bar that would

[1] We need to manipulate the dimensions because the PostScript code is expecting a number and assuming that it means that number of points. Supplying a LaTeX dimension will produce a number followed by 'pt', i.e., 25.0pt, when what we need is 25.

appear in the margin, with a short version of the chapter title running sideways in a colored block, topped by the chapter number printed upright. This graphic effect would also change position depending on the chapter of the book, starting at the top of the page and gradually moving down. This striking effect was produced with a combination of LaTeX macros and PostScript code, in which information was passed from LaTeX to the PostScript code used to form the graphic.

Looking Forward: The Basic Wonderfulness of PDF

Many of you are already familiar with the Adobe Acrobat program that produces and reads PDF files. Its cross-platform and hypertext abilities, and easy user interface, make it an attractive way to distribute on-line journals and books, either on a CD or over the Web. However, we in the LaTeX world are especially fortunate when it comes to using this program because:

1. The program that produces PDF, the Acrobat Distiller, processes PostScript files. Since most LaTeX documents are translated to PostScript routinely, that means that they are ready to be distilled with no extra effort, other than being sure to use outline fonts, and a driver program run on the `.dvi` file to produce a `.ps` file.
2. The Acrobat pdfmark commands can be added to the LaTeX file, passed through the driver program unchanged, to be used by the Distiller program. This allows pre-linking of any appropriate material, as well as other features, such as generating Acrobat bookmarks automatically, changing colors of specific parts of the document, and controlling many other aspects of the final PDF file.

Examples of passing LaTeX information to PDF include the possibility of prelinking the Table of Contents, List of Tables, and List of Figures. The viewer can then click on any item in one of these environments and pop to the page listed. Similarly, cross-references, bibliography citations, and footnotes can by colored and hypertext-linked automatically, as can indices. Graphics can be added to every page, if desired, and be linked to the Table of Contents and to the Index, to make it easy for the viewer to access either of these sections of the document. Graphics can be used in specific cases and linked to the appropriate referant. For instance, a printed question might appear in the margin of a document — 'Need more information?' — and the user who clicks there would be sent to an appropriate appendix or other source of information. Another possibility for complex technical documents would be to have links to a glossary, so that the first time a term is used it would be highlighted and the user could click on it to jump to the appropriate glossary entry.

Color is free. Usually book publishers are concerned about adding color to their books because of the added cost, understandably, since each color makes the conventional printing process substantially more expensive. When using PDF, the cost of adding color is no longer an issue, so escaping from dreary black and white becomes a no-cost option. Pdfmark commands to set a particular color may also be inserted into a LaTeX macro file, so that particular parts of the document will appear in the chosen color automatically. There is tremendous potential for LaTeX/PDF book and journal production used for on-line distribution, database publishing, and many more applications.

An example of an application of LaTeX-PDF which I prepared recently was for a company that uses LaTeX for the over 400-page documentation of their statistical software package. They asked me to provide macros to produce a PDF form of their documentation for on-line help for their software.

The user of their software will now be able to click on the correct entry in the 'help' menu to access a PDF version of the documentation without leaving the original program. The Acrobat Reader program will pop up with the PDF file containing the full version of the printed documentation. When the user has found the bit of information that they need they can return to the original program which continues running in the background.

This means that instead of using only the usual RoboHelp files that must be written by the company separately from the documentation, and which would necessarily be a subset of the complete printed documentation, users will now have access to the complete documentation, with hypertext linking in the complete index, table of contents, and glossary. Graphics at the top of the page will allow users to easily jump to the contents or the index. This would seem to be a very attractive method of producing on-line help for those software companies that use LaTeX to produce their documentation.

Another innovative use of TeX and PDF can be seen in the University of Akron mathematics

professor David Story's online Calculus Tutorial, and Algebra Review. You will find them at

```
Home Page:
   http://www.math.uakron.edu/~dpstory/
e-Calculus:
   http://www.math.uakron.edu/
            ~dpstory/e-calculus.html
An Algebra Review in 10 Lessons:
   http://www.math.uakron.edu/
            ~dpstory/mpt_home.html
e-mail:
   dpstory@uakron.edu
```

I consider PDF production the cutting edge of the LaTeX world, and I look forward to exploring its potential, as I expect you will too.

Happy TeXing!

◇ Amy Hendrickson
TeXnology Inc.
57 Longwood Avenue
Brookline, MA 02146
USA
amyh@ai.mit.edu

Typesetting with TEX and LATEX

Alan Hoenig
17 Bay Avenue
Huntington, NY 11743
ajhjj@cunyvm.cuny.edu

(This presentation appears in a considerably expanded form as chapter 1 of my book *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, and More* published just this year by the Oxford University Press.)

By *typesetting*, we mean the ability to place elements of a document on a page according to generally accepted principles which most people seem to agree look best and make it easiest to read and comprehend the document. It's surprisingly difficult to do that — a typesetter has to decide how best to break paragraphs into lines, how to hyphenate words, how to leave space for footnotes, how to prepare indexes and the other detritus of scholarly publishing, and provide the optimum space between elements on the page (among many other things). The spacing issue is particularly critical for technical documents. Formulas make extensive use of arcane symbols which have different appearances and spacing depending on context. Consider, for example, how the placement and spacing of the ordinary numeral '2' changes in

$$2x \qquad x^2 \qquad e^{-x^2}$$

and how the spacing surrounding minus sign changes in

$$x - y \quad \text{and} \quad -x + y$$

Other symbols may change depending upon whether the equation appears in text '$\int x\,dx$' or display mode:

$$\int x\,dx.$$

Furthermore, if a computer system is going to control the typesetting, we expect more of it than from a mere human. We may expect, for example, to be able to label an equation in some logical way and then refer to it later by this label in our source document. It would be up to the typesetter to resolve these labels and references and replace the labels by properly formatted label numbers.

The TEX system has been freely available since the mid-80s or so and accomplishes all of the above tasks (and more) in a particularly effective manner. TEX is the creation of Donald E. Knuth of Stanford University, who has placed all the source code for TEX in the public domain. The logo 'TEX' is related to the Greek root '$\tau\epsilon\chi$' from whence come words like 'technology'. If pronounced properly, the face of your listener may become slightly moist (but no one complains if you say 'tek').

The purpose of this survey is to acquaint readers with the aspects of the TEX cycle necessary to produce handsome papers and books. This presentation should *not* be regarded as a be-all-and-end-all tutorial, since (like many other mature and sophisticated software systems) lengthy books are not enough to do full justice to it.

The TEX production cycle

Why is 'typesetting' not the same thing as 'word processing'? Typically, a word processor allows editing of the document, but in its impatience to display the results immediately onscreen (most word processors are aggressively WYSIWYG in behavior), certain niceties are sacrificed. These niceties — fine control of spacing, word placement, hyphenation, and so forth — are never ignored in TEX.

It's useful to consider the TEX production cycle by comparing it with that of word processors. In a word processor, the program assists you in preparing the document, after which it is printed. TEX relies on three steps.

1. We use a *text editor* to prepare the *source document* — the document file which consists of the text and data of your document together with the TEX formatting commands. Let's suppose this file is called `myfile.tex`.

2. We run `myfile.tex` through the TEX program. If all goes well, this generates a file in which the typesetting commands are made explicit using a generic printer description language independent of any particular printer; it is *device independent*. TEX names this file `myfile.dvi`. Just like a computer program source file with syntax errors, if there are any errors, we return to step 1 and correct them before continuing.

3. Finally, we need the assistance of a special *device driver* customized to the printer. It's the

driver's task to translate the generic `dvi` commands into the form the printer understands.

The advantages of creating a `.dvi` file are that we can print the document on any printer (at least, any printer for which device drivers exist) and rest assured that the output is identical on each device (except for raster resolution).

Macros; logical document design

TEX has been called an assembly language for typesetting. This means that there are plenty of primitive commands to control fine points, but these commands may not be entirely appropriate for creating a new section head or aligned equation. As a result, TEX has a rich and powerful macro creation facility. It's possible (as we will see later) to string primitive commands and pre-existing macro commands together to create new, custom typesetting commands.

Remember that the TEX production cycle means that we prepare a source file which is fed into TEX at a later point. This plus the nature of the macros means that TEX supports the notion of *logical document design*. We can embed components of the document by means of tags which can be defined or redefined depending upon context. One example suffices. Here's a theorem.

Theorem There is no royal road to typesetting. Computer typesetting is a surprisingly complex task.

This was typeset by means of inserting

```
\theorem There ...
```

in the source document. It may happen that it is more appropriate to display that theorem as

THEOREM *There is no royal road to typesetting. Computer typesetting is a surprisingly complex task.*

The same command string will accomplish this *provided* that only the macro definition of `\theorem` needs be changed. The implications are enormous — we can design our document so that it will properly printed for any set of particular formatting requirements provided only that we change the particular definitions of the macros. Many strategies exist for facilitating this use of definitions.

A first TEX document

The "steps" for generating a TEX document are well-defined, but there are sufficiently idiosyncratic implementations of TEX floating around so that it may be necessary to adapt these procedures to a local adaptation. By the way, some readers may be interested in the TEX dialect called 'LATEX'; as we will see, LATEX is the same as TEX, so these procedures follow for a LATEX document as well.

1. Use a *text editor* to create the *source document* for subsequent processing by TEX. The source document is the document file — text and typesetting commands. Take care not to use a word processor. These programs aim to do the formatting themselves, and tend to do so by inserting non-Ascii characters into the document file. Quite apart from the fact that TEX (or LATEX) needs no help with the typesetting, these binary characters will only confuse TEX. (If it is necessary to use a word processor, make sure to save the document in some way so as not to include the word processing formatting information.)

2. Run this source file through the TEX program. the simplest form of the command to do that is

   ```
   tex myfile
   ```

   where the source file has the name `myfile.tex`. LATEX users will use the command

   ```
   tex &lplain myfile
   ```

   (Unix users may have to enter the ampersand as `\&`.)

   As in any compilation process, TEX may uncover errors. (Warnings may be ignored, at least at this stage.) Return to step 1 to correct these errors, and re-run it through TEX. Repeat this process until all errors have been dealt with (or until there is enough of a document to print.)

   The result of a successful TEX compilation is a new file with a `dvi` extension. In this example, we would have a new file `myfile.dvi`.

3. With the document in hand, it can be printed or previewed on screen. In each case, appropriate *device drivers* are necessary to properly render the document on screen or on paper.

As we see, the TEX process is actually a concerted action between several programs in addition to TEX — a text editor, a device driver, and a screen previewer. Many implementations of TEX may merge several or all of these into one integrated module.

The TEX document: input conventions

Although it is not practical or possible to deal with all or even a completely useful subset of all TEX (or LATEX) commands in this article, it is possible to summarize the keyboard conventions that any TEX typesetting must adhere to.

White space. TEX normally regards all white space as equivalent, where we include carriage returns, tabs, and of course spaces in this category. Furthermore, multiple spaces are generally equivalent to a single space. **Important exception:** we signal the end of one paragraph and the beginning of another by skipping a line in the source file; that is, we enter two hard carriage returns in a row. (But three or more consecutive carriage returns is still equivalent to a pair of carriage returns.)

Once in a while, spaces are special in that we don't want a line broken between two words or word groups. For example, in a discussion of World War I, it would look silly if a line broke between 'World War' and 'I'; it would be too confusing to the reader. To guarantee that the line break won't happen at that point, we replace the space with the tilde character `~`. If this *conditional space* is typeset in the middle of a line, it appears as a regular space. Aspiring TEX typists should develop the habit of typing things like `King Henry~VIII`, `Dr.~Knuth`, and `pages~44--55` to protect the manuscript from unwarranted line breaks. Note that whereas `Henry~VIII` will work as advertised, `Henry ~ VIII` will not. Here is one instance where users need to be careful.

Characters. We generate most characters by simply entering the character in the source document. That is, we type

```
Oh!  What a beautiful morning.
```

to get

Oh! What a beautiful morning.

See below for exceptions to this; certain special characters need be entered in a special way.

But TEX is smarter than that. Certain character pairs are replaced by special glyphs. For example, if we type `‘‘` or `’’` we get true "quotes." With its special attention to details, TEX will replace certain character combinations such as `fi`, `fl`, and `ff` by the *ligatures* fi, fl, and ff (provided these ligatures are present in the current font).

TEX does a similar thing with hyphens and dashes. We can type `-`, `--`, or `---` to put -, –, or — in our documents. (And we will see later that the mathematical minus sign — yet a different dash — can be gotten using the hyphen character in mathematics mode.)

By the way, no user should *ever* put an explicit line-break hyphen in a word which has to be split at the end of a line. TEX's hyphenation algorithm takes care of such should a word break be necessary.

In summary, we type

```
‘‘Oh, the selfish shell-fish---that
lobster mobster---tasted
best when basted west,’’ quoth Aaron
while reading
pages~12--33 of his cookbook.
```

to typeset

"Oh, the selfish shell-fish — that lobster mobster — tasted best when basted west," quoth Aaron while reading pages 12 – 33 of his cookbook.

TEX formatting instructions and commands. TEX is very good about applying default typesetting parameters to text, but there will be many times when you wish to actively control the printed appearance of your document by issuing commands to TEX. Since the entire file must contain Ascii characters only, TEX has decided to reserve the meanings of certain characters to itself. The tilde `~` is one such special character. To typeset an actual tilde ~ in the document, you must enter a short command to do so.

These characters

```
\  #  $  %  ^  &  ~  {  }
```

have special meanings to TEX. The backslash `\` is TEX's *escape character* — it escapes the normal meaning of the following bit of text. This character generally begins all of TEX's commands. For example, to typeset an ampersand &, you would type `\&`. Typesetting commands for some of these symbols are formed in the same way. (All the symbols can be typeset, but for some, addtional TEX expertise is needed.)

TEX can do àçc̈éñṫṣ as well. If you need them, check your main manual. We can get the Spanish punctuation marks ¿ and ¡ by typing `?‘` and `!‘`.

We will discuss the special TEX characters bit by bit, but commands generally begin with the escape character. The escape character can be followed one single non-letter, or by an arbitrary sequence of letters, terminated by a space. Examples of commands from the first category include `\&`, `\1`, `\$`, and `\"`. Examples of the second category include `\TeX`, `\L`, `\noindent`, `\vskip`, and `\futurelet`. Note that TEX is case sensitive, so the command `\TeX` (which typesets the TEX logo) is different from the (nonsense) commands `\tex` and `\TEX`.

These rules lead to our first piece of TEXarcana. As part of TEX's digestive process, it is smart enough to know that when a non-letter follows an escape character (the backslash), the command name consists only of a single letter. Hence, anything following that command, such as a space, is

typeset as you expect. To get the sequence, '& &', type `\& \&`.

The situation is subtly different when a command name follows the backslash. For now, TEX has no way *a priori* to know the length of the command name. It reads your file, and terminates the command name when it encounters a space or a new command. Consequently the space following a command is 'eaten up' by TEX — it serves not introduce a space into the document but rather to delimit the command. But since at other times, spaces typed in the document file *do* generate a space, it's easy to see why newcomers are easily confused.

Anyway, to illustrate the point, suppose you wanted to typeset 'TEX TEX'. The way *not* to do it would be by entering `\TeX \TeX` into the source, for the interior space terminates the initial `\TeX` command and is therefore eaten alive. (`\TeX \TeX` typesets as TEXTEX.) Since multiple spaces count as a single space to TEX, the solution is *not* to insert addtional spaces. The following list, which suggests several ways out of the impasse, also hones your beginning TEX skills.

1. Use the TEX command which explicitly generates an interword gobber of space. This *control space* command is `\␣`, and so your source should like `\TeX\␣\TeX` if you use this method.
2. Terminate the command by inserting some command which does not print anything. The empty group `{}` is one such; thus, we could type `\TeX{} \TeX`.
3. Simply surround the command in its own group: `{\TeX} \TeX` is one appropriate way to do this.

A working TEX system

A complete TEX system is actually a concert between several different component pieces of hardware and software. There are at least three different but necessary pieces of software.

First is a version of TEX for a particular computer and operating system. At this time, there are versions of TEX available for every reasonable computer. In the unlikely event that there isn't, it's possible to customize TEX by doing a reasonable amount of spade work yourself. The TEX program is in the public domain (and in electronic form), and all you need to do is make whatever changes (if any) are called for and recompile the TEX source code in a robust Pascal compiler that works on your system. (Most likely, you will translate the original Pascal WEB source to a C source program using the freely available `web2c` utility, and then use a robust C compiler to compile TEX.) TEX was originally written in Pascal, and depending on how you "pretty print" the listing, it amounts to between 20,000 and 30,000 lines of code. TEX exercises all the dark corners of any compiler, so you need a compiler that has itself been thoroughly debugged. (There is a white lie of omission in this account. All this source code is written in WEB, so some mastery of this WEB system must be acquired.)

Do we need a special version of LATEX to match our hardware? The core LATEX files, which "sit" on top of TEX, are ASCII files, and we can easily transfer ASCII files from one computer to another. However, proper behavior of LATEX will require us to install LATEX and in that process to create a special binary format file for use by our computer. In general, binary files may not be transferred from one type of hardware to another (but format files are easy to construct).

Next is a *text editor*. This was discussed earlier and is necessary for preparing the document source file.

Finally there are the *device driver* and *screen previewer*. TEX's output is a file containing commands to typeset all the letters, rules, and special symbols in the document. Unfortunately, different printers obey distinctly different sets of such commands. Therefore, TEX employs a generic, no-frills, *device-independent* language in which to express these commands. That's why the output file from TEX has the extension `dvi`, to suggest *device-independence*. In this way, the TEX program is relevant to virtually any hardware setup, but it does mean that we need yet another program, a so-called device driver. The purpose of this program is simply to translate TEX's generic, device independent typesetting commands into commands that our particular printer understands.

Everybody will want to arm themselves with a *screen previewer*, a special-purpose device driver. Remember, TEX is not WYSIWYG, and we frequently want to see what the TEX document will look like without going to the bother of printing it out. (This may be because we share printing facilities in some computer center, or because your printer takes a long time to deliver a single page. Anyone who has tried generating TEX output on a dot matrix printer knows that feeling.) A video monitor is just a special purpose printing device, and it is usually a straightforward matter to write a device driver to paint the image of the page on a monitor screen.

It makes sense to choose hardware on the basis of TEX software. For example, you'll want to make *really* sure that the printer is one for which

a device driver exists. (Or else make sure that the printer is one that will *emulate* — imitate — a supported printer. For example, there are many laser printers for sale, each with its own protocol for generating printed images. There are relatively few laser printer device drivers available. Among these few with support are the Hewlett-Packard laser printers, which many other laser printers emulate well. Since there are several Hewlett-Packard laser printer drivers available, an HP-like laser printer may be a safe bet.)

TEX also runs well on printers that understand the PostScript page description language. This PostScript language is another means for creating device-independent files, because the mechanism for rendering the PostScript document resides in the printer itself. Consequently, we need a special PostScript printer in order to take advantage of the PostScript technology. (That there are hundreds of beautiful digital PostScript fonts is another inducement to use PostScript.) Special `dvi`-to-PostScript postprocessors translate a `dvi` file to a PostScript equivalent. Many such programs are available from any number of vendors. Fortunately, one of the best, dvips by Tomas Rokicki, is freely available.

Getting TEX

Although the TEX software is "free" — within the public domain — it often takes work to port it to a particular computer. This is true of implementations for the original IBM PC and for the Apple Macintosh, for example. In any case, device drivers and screen previewers were never part of the original TEX package. Consequently, some firms sell their implementations for TEX.

An interesting recent phenomenon is the availability of several public domain TEX implementations for microcomputers. One or more such implementations exist for all kinds of personal computer, including IBM-type computers, Apple Macintosh, Amiga, Atari, and Acorn. Some of them may even be as good or better than commercial products. Of course, when we use a public domain version, we are on our own. Companies have "helplines" for users who find themselves in trouble. With rare exceptions, no such lifelines exist for users of public domain TEXs. Public domain implementations are available from user groups (TUG, Dante, GUT, and so on), from Internet archives, and from special TEX CDROMs.

When acquiring a TEX "package," make sure it's complete. In addition to the TEX executable, the associated ancillary files TEX needs, various important input files, and the latest version of important macro packages, we must make sure additional utilities are part of the suite even if your current plans don't include using them. I have in mind here the METAFONT program (and the MP program if possible), various TEXware utilities (of which `tftopl`, `vftovp` and their inverses `pltotf` and `vptovf` are probably the most important), and the METAFONTware utilities. The TEX program should be version 3.1415 or higher, and the METAFONT program should be version 2.71 or higher.

Unique TEXs. Although each implementation of TEX is typographically equivalent to any other, a few are worthy of special notice by virtue of some distinguishing feature. A few of these special TEXs are worthy of mention here.

The em-TEX software collection is especially interesting — it's a complete implementation of TEX, METAFONT, all TEXware and MFware programs, printer drivers, previewers, and documentation (in English and German) for PC-DOS and OS/2 operating systems, and it's all free. Several executables of TEX and METAFONT are provided, from "small" to "huge" versions. (These designations refer to the speed and/or the amount of material these TEXs and METAFONTs can process.) Eberhard Mattes is the man behind this prodigious effort, but there are those who wonder if "Eberhard Mattes" doesn't refer, like the fictional "Nicolas Bourbaki," to a dedicated group of workers. In any case, this material is all available free for downloading from any CTAN site, and some user groups make it available to their members for a nominal fee. Furthermore, there is a special em-TEX Internet list, so this is one important instance where public domain software does have some support. Over the last several years, Mattes has proven to be a conscientious developer, providing bug fixes in a timely way and keeping up with the latest master source files of TEX, METAFONT, and their friends.

Tom Rokicki is well-known in TEX circles for his dvips post-processor (it converts `dvi` output to a form suitable for rendering on a PostScript printer). Less well known but just as impressive is TEXView, his version of TEX for the NextStep operating system. NextStep, which runs on the Intel-486 architecture among others, is a flavor of Unix (BSD 4.2) onto which has been grafted a very convenient windowing system. NextStep contains a version of Display PostScript, which means that TEX documents that incorporate PostScript graphics and PostScript fonts may be previewed effortlessly (including color). Because Unix is a multitasking system, an author

can run a document through TEX, begin the previewing process, and continue editing. One odd feature has proven invaluable — the ability to measure actual distances on a page with clicks of a mouse. It's surprising how often it is possible to fix a bug in a TEX file by simply knowing how much extra or missing space there is. Most of the TEXView enhancements can be found incorporated into the `web2c` TEX kits for Unix platforms. (As a result of various corporate acquisitions, NextStep is now called OpenStep.)

Lightning Textures, by Blue Sky Research, is a version of TEX for the Macintosh. Its distinguishing feature is its ability to show TEX output produced simultaneously as text is keyed in. (And look for its newly-arrived sibling "Synchronicity.") The freely available InstantTEX for NextStep (originally by Dmitri Linde and now maintained by Gregor Hoffleit) provides the same functionality — it's great for debugging macros. InstantTEX is freely available from the `ftp` site `peanuts.leo.org` and others, in the area

`pub/comp/platforms/next/Text/tex/apps`

and its mirrors. The file will have a name like `InstantTeX.3.11d.NIHS.b.tar` and there is an accompanying "`readme`" file as well.

AucTEX is not an implementation, but an editing enhancement available to Unix users of the Emacs editor. With it, Emacs becomes highly TEX-aware, making available a large number of shortcuts, command completion, automatic indentation, special outlining, online documentation, the ability to customize (provided you can program in Lisp, the language in which it is programmed), and a good bit more. Kresten Krab Thorup is its author, and it is available from any CTAN site.

Inking the page

Neither TEX nor LATEX (nor any other macro package) actually paints the page. TEX simply creates a file, the `dvi` file, which precisely records the position of each character, rule, and other graphic element in the document. Other programs take responsibility for using this information to place ink on the page. Remember, the TEX program needs only to know how much space is required for each character, rule, and typographic element on the page.

The characters of a font. It's the province of the device driver (or previewer, just another type of device driver) to deal with the characters of a digital font. Information in the `dvi` file tells it where to position each character, and then the driver paints each character where it is supposed to be.

Bitmap fonts. How does the driver know these shapes? One way to store shape information is within a collection of *pixel files*. Computer printers generate their shapes by putting lots of tiny dots next to each other in such a way that they form patterns which our eyes resolve into letters and graphs. Office laser printers, for example, are capable of placing as many as 600 dots per inch (or more) to the left or right and up or down. Only the dots needed to create a character are printed, and the human eye smoothes out any jaggedness as it perceives the image. Pixel files, or *bitmap fonts*, provide instructions as to which dots to blacken and which to leave blank.

There are lots of pixel files because TEX needs a different file for each font of "type" at each size and at each magnification. What's the difference between the *size* of a font and the *magnification* of a font? Type designers of old took care to slightly redesign each font of type at different sizes. Subtle issues of spacing require that the proportions between thick and thin strokes, the size of the white areas within some letters, and so on be readjusted at each size. *The TEXbook* makes this point early on, on page 16, which shows the difference between 10-point type and 5-point type magnified two hundred percent. That and the following demonstrations have members from the Computer Modern Roman families. Computer Modern Roman type at a 10-pt design size (type size or just *size*) is referred to as `cmr10`.

Although TEX has been designed to appreciate this subtle difference in type sizing, many computer typesetting systems do *not*. Therefore, prevailing electronic fonts of type construct different type sizes by taking a single font and magnifying it by whatever amount necessary to get the size you want. That is, there is generally no difference between *type size* and *magnification* unless you work with Computer Modern types. (Recently, there are some indications that the non-TEX world may be beginning to perceive the importance of this distinction. Adobe's Multiple Master technology is one sign of this trend.)

The many different font files that normally come with TEX often confuse users, but now we know why they are necessary. TEX *does* distinguish between size and magnification, and TEX's ability to make this distinction calls this diversity into being.

On DOS systems (and any others that impose rigid lengths on possible file names), the necessity for having many different fonts at different sizes and magnifications calls an intricate directory structure

into being. All `cmr10` fonts at any magnification have the name `cmr10.pk`. The magnification is distinguished by creating different directories to hold pixel files with like magnifications. For fonts rendered to print on a 300 dpi printer, fonts at magnification 1000 (normal size) appear in directories named something like

```
\tex\fonts\dpi300
```

while for 120% magnification, since $360 = 1.2 \times 300$, the directory will be

```
\tex\fonts\dpi360
```

Scalable fonts and a PostScript postscript. PostScript technology has come to dominate the world of computer typesetting and desktop publishing. Files fully (and carefully) prepared in PostScript are device-independent and are generally ASCII (so they can be freely transferred across computer platforms). "Device independence" means any PostScript device. (With the Level 2 enhancement, PostScript supports some binary encoding, so such files may no longer be pure ASCII. Such files may still be moved across platforms as Adobe took great care to ensure this.)

Since PostScript files are independent of the printer resolution, fonts for PostScript cannot rely on bitmap descriptions, which are inherently tied to a printer's resolution. Each character in a PostScript font is given by a mathematical description of the outline of that character (hence, the appellation 'outline font'); this description is not dependent on any printer resolution. Nevertheless, any raster printing device is, in the final analysis, a set of bitmap images, but it's the job of the special PostScript printer to resolve the outline descriptions to their bitmap equivalents.

PostScript fonts are called *scalable fonts* because the technology scales these outline descriptions up or down to get different sizes.

TEX and PostScript technology coexist quite companionably. To render a `dvi` file on a PostScript printer, the `dvi` file must be translated to the PostScript language. A variety of dvi-to-PostScript converters exist for this purpose; one such, and one of the most highly regarded, is the freely available dvips by Tomas Rokicki. (It has been compiled for virtually every computer platform.) The end product of the translation is a series of statements in the PostScript language, which are either transmitted immediately to the printer or saved to a special file with a `ps` extension.

Beware — these `dvi` postprocessors are savvy enough to embed bitmap descriptions into the output `.ps` file in the proper format for printing on a PostScript printer. Only now the file has become device-dependent — it will print properly only on the printer for which the bitmaps were created. PostScript does try to scale the bitmap image, but an inevitable loss of quality accompanies this operation. Thus, if you plan to print the document later via phototypesetter, you'll have to regenerate the `.dvi` and `.ps` files with the proper bitmaps.

Document files

What do TEX commands look like? TEX reserve ten characters for their own use. Otherwise, when TEX encounters a letter or symbol such as an "A" or "9" it interprets the symbol as a command to typeset that symbol (to typeset an uppercase A or the numeral 9). We call the character that alerts TEX to an immediately following command the *escape character*, but this character bears no relation to the key marked "escape" or "esc" on many keyboards.

We issue explicit commands to TEX by means of one of several hundred commands beginning with the backslash, the usual escape character for TEX. Immediately following the backslash are one or more characters, which may be followed by additional information that the command needs. For example, the command

```
\noindent
```

suppresses indentation at the beginning of a paragraph while

```
\vspace{1in}
```

instructs LATEX to skip one inch of vertical space.

It's important that special reserved characters begin or introduce special formatting instructions, because you never know when you might want to include the word "noindent" or "vspace" within the document. By the way, there are special commands to typeset any of the reserved symbols, so it is possible (and easy) to typeset a backslash or a dollar sign within the document.

TEX contains a rich set of several hundred or so commands. Although by themselves they do just about anything we might wish, their real strength flows from the ability to string commands together to form new commands for special typesetting purposes.

We might create our own personal `\newchapter` command to begin a new page, skip down a third of the page, center the chapter title, skip a quarter of an inch, suppress indentation on the first paragraph, and set the first two words of the chapter in a small caps font. These new commands are called *macros*.

Creating a macro is a lot like writing a computer program. TEX possesses several commands to test conditions and perform action on the basis of the test, to perform looping, and to handle input/output operations. Tricky macros can take a long time to write and test, but more often than not, it's possible to write simple macros on the fly that make typesetting much easier.

TEX lets us place all our personal macro definitions into a separate *style file*. The document would contain special instructions for TEX to read and assimilate those macros before typesetting. Publishers, for example, could exploit this by making generic style files available to their authors while commissioning a designer to prepare a style file to implement a specialized book layout. The typeset document file remains the same, and only the macro definitions change. LATEX exploits this philosophy to the hilt.

Learning and joining

Although the numbers of users of TEX are (as of yet) dwarfed in magnitude by users of famous commercial products, the growth in use of TEX is astonishing. For consider this: an idiosyncratic product, passed along by word of mouth, has no public-relations dollars behind it. The reaction of users learning it was "free" was often the same — if it is so good, why isn't anyone selling it?

But as these words are written, it has become clear that there will always be a prominent place for TEX. Although there is widespread perception that TEX may be difficult to use well, no desktop publishing package is particularly easy. Furthermore, TEX has been able to perform certain nice things from the beginning that some famous commercial programs are still struggling over, and this has won the hearts of several mainstream publishers. Scientists the world over will never relinquish TEX — how else will they *unambiguously* capture on the page their technical expressions? Furthermore, the many millions of pages of LATEX and TEX electronic manuscript already in existence require TEX's continued presence.

Joining the TEX community. TEX, METAFONT, and their friends form a rich set of tools. Many workers have spent many happy hours adapting TEX to various specialized tasks or to creating special front ends that might be just what you were looking for. Although the many computer networks form a platform for the communication of this news, the various user groups provide a more formal forum for the exchange of important news. Not only do the larger of these groups publish their own newsletters, but they often sponsor annual meetings.

The TEX Users Group — TUG — is the original user group organization. Originally an offshoot of the American Mathematical Society, it is now an independent organization. TUG serves as a clearinghouse for all information on TEX, and members receive the journal *TUGboat*, the transactions of the TEX User Group.

Other user group organizations have since arisen. New user groups, particularly behind the former Iron Curtain, are constantly being formed. (The attention of English-speaking readers must certainly be drawn to UK TUG, whose journal *Baskerville* is well worthwhile.)

Alternatives to Computer Modern Mathematics

Alan Hoenig
17 Bay Avenue
Huntington, NY 11743
ajhjj@cunyvm.cuny.edu

TEX users early on hustled to do everything with TEX that was available to other publishing software. It has long been possible to use all kinds of graphics and fancy fonts with TEX. One hole remains in this scenario, and that is the ability to use fonts other than Computer Modern for scientific typesetting. This is a real puzzle, when you consider that the whole reason for TEX in the first place was scientific typesetting. In the accompanying discussion, we attempt to fill in this hole by presenting several strategies for non-CM mathematical typesetting.

Naïvely, you might wonder why we just can't replace the Computer Modern text fonts of a document by some other fonts for a brand-new look. If we combine Computer Modern math with Times Roman text (or something comparable), the variables look too anemic and thin compared with the text letters, and in an extended document this incompatible contrast between text and math grates on the reader. You may find that some differences between math and text are acceptable—after all, math *is* different from prose—but these variations are somehow too disparate.

This article, drawn from the more extended discussion found in chapter 10 of my book *TEX Unbound* (1998, Oxford University Press), contains the results of some experiments showing how to use TEX to generate technical documents using many handsome fonts. We will be creating series of virtual fonts to do the typesetting for us.

We will discuss several strategies for mathematical typesetting:

- replacing Computer Modern Roman by Monotype Modern;
- the use of commercial and other math fonts that can then be integrated with text fonts. Although vendors may supply macro and style files to perform this integration, we will explore virtual font approaches. The four special sets of raw math fonts include MathTime, Euler, Lucida New Math, and Mathematica fonts; and
- using variations of the usual Computer Modern bitmap math fonts whose parameters have been adjusted so they more closely match their accompanying text fonts.

Computer Modern math plus new text fonts

It may be typographically dangerous to willy-nilly change the text font of a document while retaining Computer Modern math, but it is possible to choose a font that does blend well with Computer Modern. Computer Modern fonts were designed using Monotype Modern No. 8A as a model. The digital font most resembling these fonts is Monotype's Modern font, widely available from digital font vendors.

You should install the fonts as per the usual procedures. Then, LATEX users should add the command

```
\renewcommand{\rmdefault}{mmo}
```

where `mmo` is the Berry name for the Modern font family. No changes need be made to the math font declarations, as we shall continue to use the Computer Modern math fonts.

New math raw fonts

In an ideal world, math fonts would be 100% compatible with text fonts. For the math fonts available to TEX, this statement is true only when Computer Modern fonts are selected, when Times Roman is used with MathTime or the Mathematica fonts, or when Lucida Bright is used with Lucida New Math. However, if reasonable compromises are permitted, a much wider selection of font matches is available.

The x-height is the dominant physical feature of a font, since so much of a document is lowercase. In our math fonts, we take pains to match the x-height of the math fonts with that of the text types. Moreover, it makes more sense to scale the math to the text, rather than vice versa, since there is almost always more text than math in a paper. Presumably, the 10-pt size for a text font is the optimal size for that font.

The MathInst utility

One way to install math fonts is via *MathInst*, written by me. *MathInst* creates an entire font environment for typesetting. At the moment, *MathInst* knows about four math fonts, the MathTime, Lucida, Euler math, and *Mathematica* math fonts. *MathInst* consists of a Perl script, and several additional files needed by *fontinst*. The main purpose of these Perl scripts is to match a designated text family with a set of math fonts to create new math virtual fonts.

MathInst produces lots of output. First are the fonts themselves which combine the given math fonts with a family of text fonts. In case an author has provided the names of other fonts, such as

- a sans serif *family* of fonts;
- a typewriter font;
- a calligraphic font;
- a fraktur font;
- an uppercase bold Greek font; and
- a blackboard bold font,

MathInst will make them available as well (the uppercase Greek bold font will be used to create bold math fonts). In all cases, *MathInst* takes great care to size all fonts against the text fonts to make sure that all lowercase alphabets are visually compatible.

It's not enough to be provided fonts—they must be integrated into a set of macros for easy use by an author. For LaTeX authors, *MathInst* provides a new package file that performs this integration and makes new commands available for the specialty fonts (Fraktur, blackboard bold, etc.) if these fonts are available.

Authors using `plain` will find a new style performing this same integration. These authors will need to make sure Damian Cugley's `pdcfsel` font selection macros are available. (They can be downloaded from the `macros/plain/contrib/pdcmac` area of any of the CTAN archives.)

Two test files are also provided—one for LaTeX and one for plain—so authors can see examples of the new font selection commands at work. Finally, a log file records information about the virtual font process, including scale factors, to make it easy to rerun the process using override values of the scale factors if necessary.

Installation *MathInst* itself is found on CTAN in the `fonts/utilities/mathinst` area. To install it, follow the detailed instructions that are part of the package. You will also need the `pdcfsel` font selection scheme for plain TeX mentioned above, and the text and math fonts themselves. You'll also need *fontinst*, version 1.5 or greater. Also, make sure the Perl executable appears on one of your system's path directories.

The *raw* math fonts consist of a series of outline font files plus the associated `afm` files. It's easy to install these fonts. Here are the steps appropriate for traditional systems. The same steps apply to TDS systems, but it will be necessary to be more specific about the paths for the files.

1. Place the math font files with the other scalable fonts.
2. Place the `afm` files with your other `afm` files.
3. Make sure a proper entry exists for each math font in the `map` file for your `dvi` postprocessor.

Only the last point requires additional comment. For example, for the *dvips* `psfonts.map` file for a traditional TeX system, we need entries like

```
%% MathTime fonts...
mtsy    MTSY    <mtsy.pfb
mtex    MTEX    <mtex.pfb
rmtmi   RMTMI   <rmtmi.pfb

%% Lucida New Math fonts...
lbma LucidaNewMath-Arrows <lbma.pfb
lbme LucidaNewMath-Extension <lbme.pfb
lbms LucidaNewMath-Symbol <lbms.pfb
lbmi LucidaNewMath-Italic <lbmi.pfb
lbmo LucidaNewMath-AltItalic <lbmo.pfb

%% Mathematica Math fonts...
Math1   Math1 <Math1.pfb
Math2   Math2 <Math2.pfb
Math3   Math3 <Math3.pfb
Math4   Math4 <Math4.pfb
Math5   Math5 <Math5.pfb
```

Note that these fonts are proprietary; please respect the licenses under which these fonts are sold or distributed.

Alone of the special math fonts, the Euler fonts are in the public domain. In the 1980s, the American Mathematical Society commissioned Hermann Zapf to draw a set of alphabets suitable for mathematical typesetting. The Society has since graciously made these beautiful alphabets available for free. The first major use of these fonts was to typeset the book *Concrete Mathematics*.

These fonts were implemented by METAFONT, and proper installation consists in placing the `tfm` and `pk` with their mates on your system. However, we will often be scaling these slightly to match various text fonts, and rather than regenerate many new bitmap fonts, it may be easier to use the scalable versions of these fonts, also available for free (courtesy of Basil Malyshev; they may be found in the `fonts/cm/ps-type1/bakoma` section of CTAN—but there are certain licensing conditions). In this case, these fonts will also need entries in your `map`

file. These entries on a traditional TEX system should look something like

```
%% Euler fonts...
euex10   euex10  <euex10.pfb
eufb10   eufb10  <eufb10.pfb
eufb5    eufb5   <eufb5.pfb
eufb7    eufb7   <eufb7.pfb
eufm10   eufm10  <eufm10.pfb
eufm5    eufm5   <eufm5.pfb
eufm7    eufm7   <eufm7.pfb
eurb10   eurb10  <eurb10.pfb
eurb5    eurb5   <eurb5.pfb
eurb7    eurb7   <eurb7.pfb
eurm10   eurm10  <eurm10.pfb
eurm5    eurm5   <eurm5.pfb
eurm7    eurm7   <eurm7.pfb
eusb10   eusb10  <eusb10.pfb
eusb5    eusb5   <eusb5.pfb
eusb7    eusb7   <eusb7.pfb
eusm10   eusm10  <eusm10.pfb
eusm5    eusm5   <eusm5.pfb
eusm7    eusm7   <eusm7.pfb
```

Note that the Euler fonts come in a variety of weights and sizes; `m` and `b` represent medium and bold weights, and `f`, `r`, and `s` the fraktur, roman, and symbol fonts.

Installing text fonts The *MathInst* utilities expect the text font family (and the sans serif fonts too) to be installed using the Berry font-naming scheme. One way to do this is via my *vfinst* utility or via the PSNFSS files. These text fonts *must* be installed using the original OT1 TEX encoding.

Running MathInst Once the *MathInst* software has been properly installed, you execute a module by switching to a *MathInst* work directory like `mathinst/work` and issuing a command like

```
../mathinst mt mbv
```

This command installs a MathTime family of math fonts. If your computer doesn't seem to understand this command, issue the wordier incantation

```
perl ../mathinst mt mbv
```

instead. Then, follow the further instructions that appear on the computer screen.

New math virtual fonts with MathInst

MathInst uses the two-character designations

`mt` MathTime
`lu` Lucida New Math
`eu` Euler
`ma` Mathematica

to refer to the various math fonts. The new math font families use the *z naming convention*, whereby the font family name for the new math fonts uses the two-character math designations together with the text font family designation. Suppose we combine MathTime (`mt`) with Baskerville, whose family designation is `mbv` in the Berry scheme. The new fonts will be described in macro files `zmtmbv.tex` and `zmtmbv.sty` (for users of `plain` or LATEX). Baskerville plus Euler or Baskerville plus Lucida would form the *z*-names `zeumbv` and `zlumbv`. The *z*-name `zmambv` describes a marriage between Baskerville and the *Mathematica* math fonts.

The fonts themselves follow the Berry scheme, but you don't need to keep track of this, since the *MathInst* style files load the fonts for you automatically and establish their correspondence either with familiar nicknames (`\it`, `\bf`) or with the NFSS. But if you find yourself poking about your font directories, here's a quick key to the many new fonts you'll see.

Additional font variants `m`, `e`, `l`, or `a` indicate a math font, while variants `m`, `y`, or `v` following the encoding digit denote the math italic, symbol, or extension fonts. For Monotype Baskerville, the four math fonts at text size have names

- `mbvrm7t`, the math Roman font;
- `mbvrm7m`, the math italic font;
- `mbvrm7y`, the symbol font; and
- `mbvrm7v`, the extension font.

(The presence of the "7" in the font name reminds us we are using the original OT1 TEX encoding.) A math Roman font connecting Baskerville and Euler or Baskerville and Lucida would be called `mbvre7t` or `mbvrl7t`. A similar math Roman font for Baskerville plus Mathematica would be `mbvra7t`.

MathInst does its best to "fake out" new fonts at script and scriptscript sizes. You will see as many as three fonts for each of the above varieties listed above. In addition to `mbvrm7t`, the math Roman font for text sizes, you'll see `mbvrm7t7` and `mbvrm7t5`, the same fonts fine-tuned for use in script and scriptscript contexts. (The suffixes "7" and "5" recall the sizes of seven and five points that Computer Modern typesetting uses for these designations.) There is only a single math extension font for any family.

Using the new fonts Whether you use LATEX or plain TEX, you'll only need to remember the *z*-name for your new fonts. One purpose of the *MathInst* test files is to provide living examples showing just how the new fonts are invoked.

In a LaTeX document, all the work is done by the package file whose first name is precisely the *z*-name for the math fonts. As with all package files, its extension is `sty`. If you add a line like

```
\usepackage{zmtmbv}
```

following `\documentclass{...}` then all the usual LaTeX font-switching mechanisms will now apply to the `zmtmbv` fonts rather than to the default Computer Modern fonts.

To typeset mathematics in a `plain` document, simply place a statement like

```
\input zmtmbv
```

very near the beginning of your file. Thereafter, all the usual plain font commands like `\it`, `\rm`, `\tt`, `$`, `$$`, and so on, refer to the new math fonts. The `pdcfsel` package itself provides more flexibility than plain users are used to, and it would be well worth any (plain) reader's time to gain familiarity with this package.

The MathTime fonts Michael Spivak developed the MathTime math fonts to be used with the Times family of text fonts in a TeX document;Y&Y is the vendor. Many authors will find these fonts the most useful for math typesetting. Their "Times Romany" look goes well with many other Roman fonts.

The package consists of three fonts—an extension font, a math italic font, and a symbol font. The extension font `mtex` is directly analogous to `cmex10` (the same characters are in the same positions), but the remaining fonts have slightly different layouts from their Computer Modern counterparts. These differences are largely due to the elimination of the oldstyle digits and the calligraphic alphabet from the italic and symbol fonts. The slots opened up by these omissions have been filled with uppercase Greek letters and redesigned operator symbols. (The documentation that accompanies the fonts discusses the differences in greater detail.)

Users can also create MathTime math fonts in a second way—by following the instructions that accompany these fonts. This approach is not so heavily dependent on virtual fonts as is the *MathInst* way and relies on a well-written macro file accompanying this package.

The Euler fonts Euler fonts consist of math literals (neither Roman nor italic, but a unique upright font which is a compromise between the two forms), symbols (with a compatible uppercase calligraphic alphabet), Fraktur, and extension fonts. Because they predate virtual fonts, and because the font tables themselves follow slightly quirky layouts, they have not been as useful heretofore as they might have been. The extension font is quite sparse, but we can add virtual flesh using Computer Modern glyphs to fill in the blanks of the font table.

Lucida New Math The Lucida math fonts for TeX were designed by to follow normal TeX typesetting conventions and yet be compatible with the extensive Lucida and Lucida Bright font families, and are available for purchase from Y&Y.

This Lucida New Math family consists of five fonts. Because each contains the full complement of 256 characters, these fonts are crammed with all kinds of additional glyphs. These additions include all the special symbols that occur in the additional symbol fonts commissioned and made available by the American Mathematical Society and include a Blackboard Bold font. The three fonts useful for standard TeXnical typesetting are the symbol, math oblique, and math extension fonts. A math italic font doesn't follow the original math italic font quite as closely as the oblique font. Finally, a math arrow font contains many, many new symbols plus the Blackboard Bold alphabet.

Lucida math extension differs from other extension fonts in that it contains many new glyphs in its upper half. With this font properly in place, you can use a new extensible set of open brackets, additional wide accents, newly sized integral symbols, and a fully extensible integral. The font also contains the uppercase Greek alphabet, which we already use to construct the math Roman font. *MathInst* style files contain commands (where necessary) to use these new features.

The wide accent symbols are automatically in place; simply continue to use `\widetilde` and `\widehat` as before.

There are new kinds of square brackets that grow to enclose filler material. These brackets, are amenable to the usual "growth" mechanisms that govern `\left`, `\right`, `\bigl`, and so on.

The several new integral signs include new surface integrals, a new size for the regular and contour integrals, and pieces for a generally extensible integral. The new command `\surfint` and the existing integral commands `\int` and `\oint` work as expected. In addition, there are large variants, summoned into play by `\lint`, `\loint`, and `\lsurfint`. These control sequences ensure that the various integral signs change their size depending on a text or display context.

You might like to have TeX select the right size integral for you. For that reason, there are three variant integral commands, `\varint`, `\varoint`, and

`\varsint` (for regular, contour, and surface integrals) that try to do that for you. Each of these takes as argument the contents of the integrand. Figure 1 shows that this mechanism works poorly for the surface and contour integrals when the total height of the integrand is taller than the largest of the available integrals.

MathInst automatically places the TEX-hackery necessary in the Lucida style files it writes. You could type

```
$$
  \overbrace{\vphantom{\lint}
    \hbox{$\int\lint\oint\loint\surfint
    \lsurfint$\ }}%
    ^{\hbox{text}}
  \overbrace{\int\lint\oint\loint\surfint
    \lsurfint}%
    ^{\hbox{display}}
$$
$$\varint_{-\infty}^{+\infty}{\setlimits
  \left\Lbrack
\vcenter{\halign{\strut\hfil${#}$\hfil\cr
  \widehat 1\,\widehat{23}\,\widehat{456}\,
  \widehat{7890}\,\widehat{12345}
\cr
  \widetilde{67890}\,\widetilde{1234}\,
  \widetilde{567}\,
  \widetilde{89}\,\widetilde{0}
\cr
  \varoint{\short}\,\varoint{\med}\,
  \varoint{\tall}\,
  \varoint{\Tall}\,\varoint{\Talll}\,
  \varoint{\VTall}
\cr
  \varsint{\VTall}\,\varsint{\Talll}\,
  \varsint{\Tall}\,
  \varsint{\tall}\,\varsint{\med}\,
  \varsint_{\scriptscriptstyle\partial C}%
  {\setlimits\omega}
\cr
  \left\Lbrack x\right\Rbrack\
  \left\Lbrack\med\right\Rbrack
  \left\Lbrack \tall \right\Rbrack\
  \left\Lbrack \Tall \right\Rbrack\
  \left\Lbrack \Talll\right\Rbrack\
  \left\Lbrack \ontop{42pt}\right\Rbrack
\cr
  \varint_0^9{\Talll}\
    \varint_{-1}^{+1}{\Tall}\
  \varint{\tall}\
  \varint^{+\infty}_{-\infty}{\med}\
    \varint{x}\cr
}}\right\Rbrack\,dx
}$$
```

to get figure 1, which combines the Lucida math and Lucida Bright Roman fonts. Here, `\short`, `\med`, `\tall` and so on are simply temporary control sequences to generate arguments of various relative heights.

Mathematica math fonts The Mathematica math fonts were in development as this book was written, but Wolfram Research graciously made their interim fonts available to me. The fonts in their eventual release may have different names, different characters, and a different ordering.

These fonts consist of five font series comprising all of the characters that TEX normally expects, a calligraphic and a Blackboard Bold alphabet, and many more additional characters. Each series consists of four variants, normal, bold, typewriter, and typewriter bold. As far as TEX is concerned, the characters in these fonts are scrambled in a funny order, so we first create raw fonts, each of which appears more meaningfully ordered to TEX. You can do this with the script `makemma.tex`, part of *MathInst*. Running this script, and then creating virtual fonts in the usual way, creates three fonts `mmami`, `mmasy`, and `mmaex` (math italic, symbol, and extension), which are themselves suitable components for virtual font shenanigans. Although these three are in fact virtual fonts, we will treat them as raw fonts in the creation of additional virtual fonts.

Fine tuning the new math fonts

Adding special-purpose fonts Authors may want to add special fonts to their math style. Here's what *MathInst* allows you to add:

- a sans serif font *family*,
- a typewriter font,
- a blackboard bold font,
- a Fraktur font,
- a calligraphic font, and
- a bold Greek font (suitable for setting bold math).

You may add any, all, or none of these. If any of these fonts are present on your system, *MathInst* adds high-level font-switching commands to the style and macro files it creates that recognize the presence of these fonts.

Where do these fonts come from? Many of them are proprietary, but a large number of them reside in the public domain, albeit in unlikely or unsuspected places.

As far as typewriter type is concerned, I strongly recommend the freely available Computer Modern typewriter font `cmtt10`, which blends well with almost every other digital face. There are alternatives. The Pandora typewriter font `pntt10` is also free from CTAN, and of course the printer-resident Courier font is widely available. There are several other variants of `cmtt10` in the TEX suite that some users may prefer, and proprietary typewriter fonts

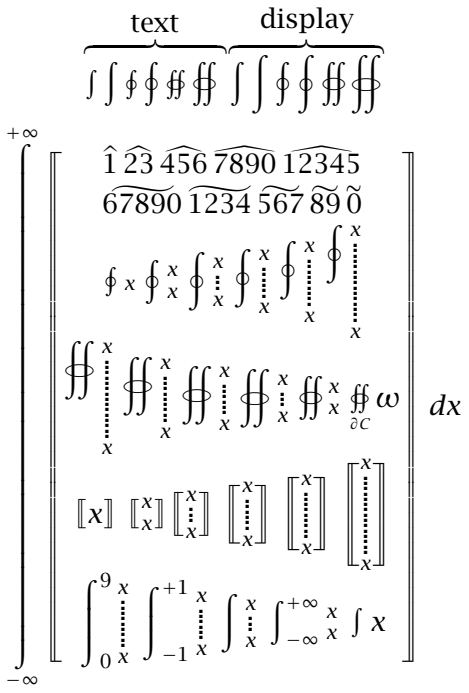


Figure 1: New Lucida math extension characters in action.

include offerings in the Lucida Bright families and ITC American Typewriter. Other authors may use a monowidth sans serif font (such as Letter Gothic) or some other contrasting face entirely.

A wide choice exists for sans serif families. Common choices will be Computer Modern sans serif, and the Helvetica fonts resident in all PostScript printers. I am personally partial to Gill Sans (from Monotype) and the Lucida Sans fonts (Bigelow & Holmes), but both of these are commercial fonts.

A calligraphic uppercase alphabet is necessary to make the `\mathcal` or `\cal` commands work properly. *MathInst* can add this alphabet (in a virtual way) to the math symbol font. Among the widely available candidates are alphabets from the Computer Modern symbol and Euler symbol fonts, and the printer-resident Zapf Chancery font. Several bitmap script fonts in the `fonts` area of CTAN (such as Calligra, the RSFS fonts, script fonts, and `twcal`) may be appropriate. Many commercial fonts are suitable, but authors should refrain from choosing too fancy a script.

There is less choice for a Blackboard Bold and Fraktur font. In CTAN, we find the `bbold` fonts (by Alan Jeffrey) and the `msbm` fonts developed and provided by the American Mathematical Society; both of these are in the `fonts` area of CTAN, the latter being in its `ams` subdirectory. Commercial sources include the Lucida New Math family (the "arrows" font contains the Blackboard Bold glyphs) and Adobe's Math Pi fonts (the sixth of these contains the Blackboard Bold). Choices on CTAN for fraktur include Euler fraktur `eufm` in `fonts/ams` (a scalable version is part of the BaKoMa collection, also on CTAN) and the `yfrak` fonts in `fonts/gothic/yfrak`. Commercial choices include the Math Pi package from Adobe (check out the second font in the series for Fraktur) and a font called Fraktur from Bitstream.

Mathematicians often want formulas in bold type. *MathInst* will create bold math fonts for you, but the sticking point might be bold variants of the uppercase Greek letters. Computer Modern, Euler, and Mathematica fonts contain bold Greek alphabets, but neither Lucida nor MathTime do. If a bold version of the Greek letters is available, *MathInst* would like to know about it. There seems to be nothing available that exactly matches the Lucida Greek types, but bold Greek Times fonts can be purchased.

To make these fonts visible to *MathInst*, you'll need to enter the names of the font files to the right

of the equal signs in the statements making assignments to `$tt_`, `$sansserif_`, and so on. Don't forget to remove any comment characters from the beginning of the line! Thus, to use `pmp6` as the source for Blackboard Bold, we need the line

```
$bbold_ = "pmp6";
```

in the parameter `par` file. Note that font names in these statements need double quotes fore and aft. Note too that these changes need to be part of *all* the `par` files (or at least all the ones you'll be using).

MathInst produces test files `testmath.tex` for LaTeX and `testmatp.tex` for `plain`. These files show how to implement the fonts you've just created and exercises these fonts in some reasonably complete manner. The files themselves are closely modeled after a similar test file originally designed by Alan Jeffrey. (The original of this file appears on CTAN in the `fonts/utilities/fontinst` area.) It is a good idea to compile these tests and print one out each time you create a new math font family.

New math fonts via Metafont

Think of the reasons that Computer Modern math fonts clash with other text fonts—they are somehow too skinny, the wrong height and depth, and their shapes may not harmonize well with text fonts. Being that they are meta-fonts, can we not alter the parameters to generate math fonts that more closely approximate text fonts we may be using? This strategy lies behind the *MathKit* scripts I have developed. *MathKit* aids in the creation of math fonts that may be compatible with a text font family. It consists of a Perl script and some auxiliary files to help an author—even one ignorant of virtual fonts or of METAFONT—to perform these tasks. This material can be found in the `fonts/utilities/mathkit` area of CTAN.

MathKit takes METAFONT parameters that are appropriate to an outline font family and uses these to create math fonts. The symbols and other special characters look pretty good—and are compatible with your outline fonts—but the italics and numerals look ghastly. Using TeX's virtual font mechanism, we create math fonts that combine the new special symbols with letters and numerals from the outline fonts. *MathKit* does some of this work for you, and provides scripts for the remaining steps (described in the accompanying documentation). It also provides style files for plain TeX and for the NFSS of LaTeX for you to use these fonts in your documents.

The current version of *MathKit* comes with three sets of font templates. Since Palatino and Times-Roman are so common, I prepared templates for these fonts. For fun, I prepared a template for Monotype Baskerville. Times comes in regular and bold series, and Baskerville in regular and semibold; Palatino is regular only.

MathKit itself produces a number of scripts and batch files. Once these are properly executed, you get the following:

1. Virtual fonts for math and text typesetting. You will also get fonts for bold math if you have supplied a template containing bold parameters.
2. Style files for plain TeX and LaTeX (NFSS) . These files support bold math if bold parameter templates were present.

The main *MathKit* script requires three parameters. These are:

1. The name of the parameter template. '`tm`' refers to Times-like parameters, '`pl`' to Palatino-like, and '`bv`' to Baskerville-like.
2. The name under which text fonts are installed. This is apt to be something like `ptm` or `mnt` for Adobe Times or Monotype Times New Roman, `ppl` for Palatino, and `mbv` for Monotype Baskerville (which is *quite* different from ITC New Baskerville).
3. The encoding your fonts follow. Only `OT1` or `ot1` (original TeX encoding) are allowed.

For example, I type

```
perl ../mathkit tm ptm OT1
```

in my work directory to create Times-like fonts following the original TeX encoding. (If your system supports the `#!` syntax for specifying the name of an interpreter, then put the proper path at the very top of `mathkit`, make sure the execute bit is set, and type the simpler injunction `../mathkit tm ptm OT1` from the work directory.)

I've had success matching `bv` (Baskerville-like) parameters to other Roman fonts. For example, I typed

```
../mathkit bv mjn OT1
```

to generate a nice-looking set of fonts combining Monotype Janson text with Baskerville-like math fonts.

The following steps complete the font creation. Perform them all within the *MathKit* work directory.

1. Use the `mkdirs` script to create any missing directories.
2. Execute the file `makegf.bat` to have METAFONT create the pixel fonts for your fonts. This step will take some time.

3. You'll need to pack all the pixel files. The file called `makepk.bat` that may be helpful in this regard. *Caution:* before executing this script, it may be necessary to edit it.
4. Execute the script `makepl.bat` to create some property list files needed by the next step.
5. Run the file `makevp.tex` through TEX. That is, execute the command `tex makevp` or something appropriate for your system. This step will take some time. Along with lots of superfluous files, this creates many "virtual property list" files with extension `vpl`.
6. Create the actual virtual files by running every `vpl` file through the program `vptovf`; execute the file `makevf.bat` which *MathKit* creates for you.
7. Execute the file `putfonts.bat` to place the font and other files where they belong.

This sequence is summarized for you again on the computer screen when you execute *MathKit*.

Using the new fonts *MathKit* produces two style files, one for LATEX and one for `plain`. Their file names are formed according the naming scheme

`z`⟨*mock-family*⟩⟨*font-family*⟩

Here, ⟨*mock-family*⟩ is the two-character designation for one of the font parameter templates (such as `tm`, `pl`, or `bv`); the word "mock" refers to the fact that these fonts imitate but don't equal the actual fonts in this family. ⟨*font-family*⟩ is the Berry family designation. Thus, if I create a Times-like set of fonts for use with font family `ptm`, I would find files `ztmptm.sty` (LATEX) and `ztmptm.tex` (plain). In the same way, the style files for mock-Palatino and mock-Baskerville fonts are named `zplppl` and `zbvmbv` (with the appropriate extensions).

At the top of a `plain` file, include the statement

```
\input ztmmnt
```

(or whatever the style file name is). Then, standard font nicknames like `\bf` and `\it` and math toggles like `$` and `\(` will thenceforward refer to these new fonts.

If bold fonts have been generated, a command `\boldface` typesets everything in its way in boldface—prose, mathematics, whatever. Bold math may be appropriate for bold captions, sections heads, and the like. Like any other font changing command, this command should be placed within grouping symbols.

In LATEX documents, you simply need to include the style name as part of the list of packages that you use in the document. Thus, a typical document would have a statement like

```
\usepackage{ztmptm,epsf,pstricks,...}
```

at the outset.

If *MathKit* has created bold math fonts, a `boldface` environment will typeset everything in that environment as bold, including all mathematics.

If your outline fonts have been installed using expert fonts, you may need to alter the `\rmdefault` command. It might be necessary, say, to type

```
\rmdefault{ptmx}
```

instead of `\rmdefault{ptm}`.

Preparing parameter files It was surprisingly easy to prepare these parameter files. I prepared a test document in which individual characters are printed on a baseline at a size of 750 pt. It's (relatively) easy to measure the dimensions of such large characters, and METAFONT can be asked to divide by 75 to compute the proper dimension for 10-pt fonts. It was particularly easy for me to make these measurements, as I use Tom Rokicki's superior implementation of TEX for NextStep. This package contains on-screen calipers, which take all the work out of this chore.

If you plan to create your own parameter files for other font families, please use the supplied files as models (those files with extensions `mkr`, `mks`, or `mkb`). Make sure all measurements are given in terms of "`pt#`"; *MathKit* looks for this string. And please consider placing this information in CTAN.

Rogues' gallery

The following displays show the results of mixing and matching various math families to many text fonts. *vfinst* installed all the text fonts, and *MathInst* or *MathKit* generated all the math + text fonts.

These displays should be regarded as experiments only. I showed these pages to several people, and all concluded that some of the experiments are successful and others are failures. However, no one agreed which were the successes and which were the failures!

Computer Modern math + Monotype Modern

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$b \simeq r\sin(\varphi - \varphi_\infty) \simeq r(\varphi - \varphi_\infty)$$
$$-V \simeq \tfrac{d}{dt}\left(r\cos(\varphi - \varphi_\infty)\right) \simeq \tfrac{dr}{dt}$$

where b is the "impact parameter" and φ_∞ is the incident directions. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$
$$E = 1 - V^2. \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\varphi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\varphi(r) = \varphi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r_0)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in φ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\varphi(r_0) - \varphi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\varphi = 2|\varphi(r_0) - \varphi_\infty| - \pi.$$

If this is positive, then the angle φ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\varphi$ is negative then the trajectory is bent away from the sun.

MathTime math + Times New Roman (Monotype)

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$b \simeq r\sin(\varphi - \varphi_\infty) \simeq r(\varphi - \varphi_\infty)$$
$$-V \simeq \tfrac{d}{dt}\,(r\cos(\varphi - \varphi_\infty)) \simeq \tfrac{dr}{dt}$$

where b is the "impact parameter" and φ_∞ is the incident directions. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$
$$E = 1 - V^2. \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\varphi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\varphi(r) = \varphi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r_0)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in φ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\varphi(r_0) - \varphi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\varphi = 2|\varphi(r_0) - \varphi_\infty| - \pi.$$

If this is positive, then the angle φ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\varphi$ is negative then the trajectory is bent away from the sun.

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$b \simeq r\sin(\varphi - \varphi_\infty) \simeq r(\varphi - \varphi_\infty)$$
$$-V \simeq \tfrac{d}{dt}\,(r\cos(\varphi - \varphi_\infty)) \simeq \tfrac{dr}{dt}$$

where b is the "impact parameter" and φ_∞ is the incident directions. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$
$$E = 1 - V^2. \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\varphi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\varphi(r) = \varphi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r_0)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in φ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\varphi(r_0) - \varphi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\varphi = 2|\varphi(r_0) - \varphi_\infty| - \pi.$$

If this is positive, then the angle φ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\varphi$ is negative then the trajectory is bent away from the sun.

Unbound Orbits: Deflection of Light by the Sun

Consider a particle or photon approaching the sun from very great distances. At infinity the metric is Minkowskian, that is, $A(\infty) = B(\infty) = 1$, and we expect motion on a straight line at constant velocity V

$$b \simeq r\sin(\varphi - \varphi_\infty) \simeq r(\varphi - \varphi_\infty)$$
$$-V \simeq \tfrac{d}{dt}\,(r\cos(\varphi - \varphi_\infty)) \simeq \tfrac{dr}{dt}$$

where b is the "impact parameter" and φ_∞ is the incident directions. We see that they do satisfy the equations of motion at infinity, where $A = B = 1$, and that the constants of motion are

$$J = bV^2 \tag{1}$$
$$E = 1 - V^2. \tag{2}$$

(Of course a photon has $V = 1$, and as we have already seen, this gives $E = 0$.) It is often more convenient to express J in terms of the distance r_0 of closest approach to the sun, rather than the impact parameter b. At r_0, $dr/d\varphi$ vanishes, so our earlier equations give

$$J = r_0\left(\frac{1}{B(r_0)} - 1 + V^2\right)^{1/2}$$

The orbit is then described by

$$\varphi(r) = \varphi_\infty + \int_r^\infty \left\{ \frac{A^{\frac{1}{2}}(r)\,dr}{r^2\left(\frac{1}{r_0^2}\left[\frac{1}{B(r)-1+V^2}\right]\left[\frac{1}{B(r_0)-1+V^2}\right]^{-1} - \frac{1}{r^2}\right)^{\frac{1}{2}}} \right\}.$$

The total change in φ as r decreases from infinity to its minimum value r_0 and then increases again to infinity is just twice its change from ∞ to r_0, that is, $2|\varphi(r_0) - \varphi'_\infty|$. If the trajectory were a straight line, this would equal just π;

$$\Delta\varphi = 2|\varphi(r_0) - \varphi_\infty| - \pi.$$

If this is positive, then the angle φ changes by more than 180°, that is, the trajectory is bent *toward* the sun; if $\Delta\varphi$ is negative then the trajectory is bent away from the sun.

Developing Database Publishing Systems Using TEX

Jeffrey McArthur
ATLIS Publishing Services
jmcarth@atlis.com

Abstract

Many directories and publications are created and maintained using "off the shelf" desktop publishing systems. Producing a publication may take months of hard work. Some publications can be converted to a database publishing system which is capable of producing a finished book in a matter of hours. This paper looks at some of the issues involved in developing a database publishing system that uses TEX as the typesetting system.

The Key Features of a Content Management System

Content management systems store information in a database. The list below enumerates some of the benefits of developing a system to maintain information using TEX as the typesetting engine:

1. Consistency: The typesetting is regular and predictable. Books that are typeset using desktop publishing are often broken into sections. Each section is done by a different person. Each person has control over the layout of their section. This can result in subtle differences between sections of a publication. A content management system using TEX removes the possibility of undesirable differences between sections.
2. Timeliness: Document preparation no longer takes days or months. Pushing a key can generate an up-to-date publication reflecting the state of the data in the system. A complete publication can be finished in a matter of hours using TEX. Information constantly changes and last minute changes will automatically be incorporated in the final output with no additional effort. The predictable nature of publication generation allows for tighter scheduling of production.
3. Indexing: Automatic generation and extraction of index information are part of the typesetting process. A content management system using TEX as the typesetting engine provides the capability to create indexes that would be impossible to do using desktop publishing in a timely manner.
4. Repeatability: The ability to generate a book over and over again with different data is a key feature of content management systems. Multiple books can be quickly created using different subsets of the information contained in the system.

This paper focuses on using TEX as the back end or typesetting engine in a content management system. The choice of TEX as the typesetting system provides many benefits but also affects the development and implementation.

Limitations of Desktop Publishing Systems

Desktop publishing systems are designed around the WYSIWYG paradigm. It is easy to create great looking pages but there are no integrity checks on the data. Without validation on the input data it is possible to have dates like "February 31", or to have a two letter state abbreviation like "23". Errors of this type are amazingly common. Spelling and grammar checking will not detect errors of this type. In a desktop publishing system the information is just textual data. If a person changes his email address, all the documents in the system must be searched and a replacement done to each and every instance. In content management systems each person is an entity. All calls to the person are by reference. If the email address of the person changes, then all references to the person are automatically updated.

One of our clients took three months with up to fifteen people to generate the index to one of their books. That index can be generated in a matter of hours with a content management system developed for them.

Desktop publishing systems are not designed for publishing large quantities of data quickly. TEX works hand in hand with the content management system to quickly and accurately generate printed or electronic documentation.

Most desktop publishing systems provide tools to import database information. The WYSIWYG paradigm means that the user is prompted to "flow" the data into the system. Style sheets provide only limited capabilities over club and widow control. Import capabilities are usually limited to importing a single table, query, or view of the data. Content management systems do not have this limitation.

Database Development Must Work Hand-in-hand with TEX

All successful content management systems must keep the goal of producing the output documents in mind during all phases of development. TEX as a typesetting engine places demands on the database design.

Sorting. It is possible to sort in TEX, but the macros to do this are complex and difficult to use. Database systems are designed for sorting and each system should be used for its strengths. Thus, all sorting should be done by the database system and not in TEX.

Accents. Many database systems in the United States are not able to accurately store accented data. Some database systems translate accented characters to some other form internally. This can cause problems when the data is output for TEX to typeset. The database system must be configured so that it is easy for the user to enter accented data, and be able to get that data back out into a file that can be typeset by TEX.

Publication order. Normalized databases have an implicit order. This seldom matches the order that is desired in the printed or electronic output. TEX can rearrange data. It is easy to exceed the capacity of TEX by trying to store large quantities of data internally so that they can be rearranged on output. The database should be designed to allow quick generation of the data in an order which closely resembles the final output. This makes the job of typesetting the data with TEX easier.

Line lengths. One limitation of TEX is that the input file must be broken into lines. Most implementations of TEX only allow one to two thousand characters on a line. This is a serious limitation with database publishing. The database system must separate the data into lines that are less than the input-line limit of TEX.

Special characters. Publishing data from a database places some requirements on the macros developed in TEX. A common requirement is that all printable characters must be usable. If the user can enter a character, they want that character to output in the finished document. If the user enters a backslash, the resulting output needs to print the backslash. Although this causes difficulties with TEX, there are several methods for solving the problem. One solution is to have the database filter the data for input into TEX. This is quite slow since it requires the database system to check every character to see if it needs to be "escaped". Another approach is to change the way TEX reads the data by using verbatim macros such as are commonly used for listings. The composition file generated by the database system does not need to be editable by a human since it is only a temporary file used to transfer data from the content management system into TEX.

This approach allows many of the control characters normally used by TEX to have their `\catcode` changed to that of a letter. Macro packages used to typeset databases often end with the following sequence:

```
\catcode`\$=\other
\catcode`\%=\other
\catcode`\&=\other
\catcode`\#=\other
\catcode`\_=\other
\catcode`\^=\other
\catcode`\{=\other
\catcode`\}=\other
```

The lines above allow most characters to be printed. The tilde character, — —, however, is a special case. Internet URLs often contain a — —. The problem with typesetting internet URLs is that the — — character should be typeset as ∼ and not as — —. Using a —∼— instead of a — — is more consistent with the way the information looks when keyed or when viewed in a browser. The simplest solution to changing the typesetting of — — is to do the following:

```
\def\Tilde{\hbox{$\sim$}}
\catcode`\~=\active
\let~=\Tilde
```

Changing the `\catcode` of —— and —— means that grouping cannot be done using them. This is usually not a problem. The parameter text rules of TEX provide a mechanism to specify the boundaries of the input parameters.

Typesetting `\` is a bit more challenging. One solution is to change the escape character to one that is non-printable. One choice is character 255. Very few typefaces define a glyph for character 255. On the PC, character 255 prints as a space. The character is non-printable, so it is difficult to give an

example how to use it. Below is a set of macros that use character 255 as the escape character. In the example below, character 255 has been replaced by the sequence `M-^?` so that the macros are readable.

```
\chardef\other=12
\def\QuoteBS{
    \begingroup
    \catcode`\\=\other
    \EndQuoteBS}
\begingroup
\catcode`\M-^?=0
\catcode`\\=\other
M-^?gdefM-^?EndQuoteBS#1\End{
    M-^?line{M-^?hfil***#1***M-^?hfil}
    M-^?endgroup}
M-^?endgroup
```

There is a problem with macros like the one above: they are difficult to read and maintain. Few texts, other than ones on TEX, use a lot of \ characters. A more rational approach is to restrict the input of data into the content management system to disallow entering a \ except in those few fields that actually require it. This simplifies writing the macros, makes the macros much more readable, and more maintainable.

Index generation. TEX has the ability to create auxiliary output files that list the page number where the `write` occurred. TEX can extract information that can be used to form an index. We have used two different methods of creating extracted indexes. One method is for TEX to be responsible for outputting a file suitable for composition with TEX. That is, TEX creates a file that can be run through TEX. The file is sorted prior to running through TEX, using a simple sort program. One major disadvantage to this method is that the resulting code is very difficult to read because many characters have their `\catcode` changed. Another difficulty is in sorting the resulting index. Care must be taken to allow the file to be sorted properly. It is usually easier for the database program to export a "sort key" into the data stream for TEX to pass on to the extracted index file than to try and create the "sort key" in TEX. This is particularly true if the index contains entries like 3M which need to sort under M (Minnesota Mining and Manufacturing). Accented characters are a problem. Care must be taken to allow accented characters to pass through TEX without any change.

A second approach is for TEX to only output the "record id" and page number. The auxiliary file is read into a database program and a new composition file is generated with the page number data. This method has a couple of advantages:

1. The database provides the tools to sort the information.
2. Indexes where an entry may occur under many headings are much easier to handle.

Extraction from the Database

The information in the database or content management system must be exported for use by TEX. For simple database projects it is possible to use standard database export utilities. Typesetting directly from the quote-delimited format can be done by making the double quote character active to start the input, and then using a form of tail-recursion to call macros for the next field.

For each field there are three macros. One to start the input, one to store or typeset the data, and one to finish the field and call the macro to start the next field. If the database contained only three fields: first name, last name, and phone extension, then an exported quote-delimited file would look like this:

```
"Jeffrey","McArthur","4253"
"Jeannine","McArthur","1234"
"Bilbo","Baggins","5678"
```

The following macro would typeset the data, moving the first name after the last name and setting the name left-justified on the line and the extension right-justified on the line.

```
\chardef\other=12

% These assume " is active
\def\BegFirstName{
    \begingroup
    \catcode`\"=\other
    \MidFirstName}

\def\BegLastName{
    \begingroup
    \catcode`\"=\other
    \MidLastName}

\def\BegExtension{
    \begingroup
    \catcode`\"=\other
    \MidExtension}

% These assume " is other.
% They pull in the parameter
% and store or set it
\def\MidFirstName#1","{
    \gdef\ValFN{#1}
```

```
    \EndFirstName}

\def\MidLastName#1","{
    \gdef\ValLN{#1}
    \EndLastName}

\def\MidExtension#1"{
    \line{
        \ValLN,
        \ValFN
        \hfil
        #1
    }
    \EndExtension}

% these assume " is other
% they reset the catcode and
% call the next field
\def\EndFirstName{\endgroup\BegLastName}
\def\EndLastName{\endgroup\BegExtension}
\def\EndExtension{\endgroup}

\catcode`\"=\active
\let"=\BegFirstName
```

There are several problems with typesetting quote delimited files. TEX can only work with files that do not exceed its input buffer line length, usually one to two thousand characters. Databases with large record structures can generate lines longer than this. If the database adds a new field to the table, then the output file would be a different structure and TEX would no longer typeset the file properly.

Generation of composition files. The approach we have taken to preparing the data for use with TEX is to write a program that generates a composition file. The program processes each record in the database and tags and outputs each field. This gives full control over the order of the data and can provide any special processing required.

We have used a descriptive tagging scheme which allows the same extracted data to be used for more than one purpose. The data file can be composed with different sets of macros producing different output. Using the same data file for multiple outputs has various advantages. For instance, the generation of a composition file can be a time-consuming process. TEX can process the data two to ten times faster than the data file can be generated. For example it can take up to four hours to extract a sixty megabyte composition file; TEX can compose that file into about twelve-hundred pages in less than half an hour.[1]

SGML. Content management systems that use SGML encoding can be typeset using TEX. Valid SGML document instances are stored as memo fields in relational databases. The document instances are extracted into composition order and "stitched together" using TEX macros. With proper care TEX can directly typeset the SGML document instances. This requires that all the tags follow consistent casing scheme, or preferably a change to the SGML declaration to make the tags case-sensitive. If no output processing or filtering on the SGML is required, the composition file can be quickly generated. This is one of the few cases we have encountered where the composition file can be generated faster than TEX can compose it.

SGML tables. SGML tables using the SoftQuad table model are computationally expensive to do in TEX. Some of the SGML tables we have typeset extend beyond four pages. Our implementation of macros in TEX to typeset SoftQuad SGML tables uses `\halign`. This has proven to be very memory intensive. Large tables require a version of TEX that can use more than 20 Meg of RAM. TEX normally processes pages quickly, but on slower hardware[2] TEX can take up to ten minutes to process a single table. During that time no pages are output. Once the table processing has finished, TEX resumes quickly outputting pages.

Entity and table validation. During our implementation of SGML content management systems we ran into some problems with tables. A valid SGML document instance can have an invalid table. That is, the table can define three columns and actually have four or more columns of data. The SGML parser is not designed to catch this type of problem. Typesetting SGML document instances that have malformed tables causes TEX to generate an error message. The content management system hides most of the details from the end users. Error messages generated by TEX are helpful to those well versed in TEX, but to an average user they are total gibberish. To avoid this problem we developed a small program using a variant of Lex[3] that compares the number of column definitions with the number of actual columns. We further enhanced the program to process the ampersand character `&` and make

[1] Using a Pentium 166 running on a Novell 4.11 network with Paradox tables.

[2] 486DX33 with 32 Meg of RAM.

[3] TPLexYacc 3.01, this version of Lex emits Pascal code instead of C.

sure that it was only used as the start of an entity and that all entities were followed by semicolons. Making & active and using \csname and \endcsname allowed TEX to process the SGML entities. The macros below show how to do this:

```
\catcode`\&=\active
\def&#1;{\csname ENTITY#1\endcsname}
```

Push-button Book Generation

Any time the user wants to generate a book, or see what a book will look like, we can push a button and have the finished book in a few hours. We have used TEX to generate completely turn-key database publishing systems in which the user does not need to know anything about TEX.

The content management systems we have developed provide the user with the ability to select what sections or documents to print. We have also implemented systems where the user has some limited capabilities to change how the book is typeset. For instance, we have developed systems where the user can select the number of columns, the font sets to use, the point sizes to use, and the overall page size. The user cannot enter arbitrary combinations since the selection is limited to combinations that would work.

Test and Fit

Push-button generation means that the user does not need to make any decision on the fly about how the program is to typeset the book.

However, the measuring capabilities of TEX do allow for runtime calculations. For example, in a common directory style the phone numbers should fit into a right-hand column, with the text on the left. If the phone numbers follow a regular pattern, as they do in the United States and Canada, then the width of the phone number column can be calculated at run time. This is done by setting a test phone number in an hbox and taking its width. The advantage of this method is that the fonts can be changed and TEX will recalculate the column widths.

Test and fit can also be used in another phone number situation. Although phone numbers in the United States and Canada follow a 3-3-4 pattern, e.g. 301-578-4200, other countries do not follow that pattern. TEX can measure a foreign phone number to see if it will fit. This is accomplished by placing the phone number into an hbox. The width of the hbox can then be compared to the space allowed on the line for phone numbers. If the phone number is larger than the allowed space, the data can be re-typeset using a smaller or condensed font. This capability should be used judiciously or the pages will be aesthetically displeasing.

Widow and Club Control

TEX provides two penalties \clubpenalty and \widowpenalty that control how paragraphs break. The paragraph-breaking capability of TEX can be used for more than just simple lines. One common rule is to "leave and carry two" entities. That is, if the book is a list of people, then the requirement is that there are always at least two people prior to and following a column break. The information for a person may actually take several output lines. If each person is eventually placed in a \line and the list of people is typeset as a paragraph, then setting \clubpenalty=10000 and \widowpenalty=10000 means that TEX will always "leave and carry two" people.

One of the limitations of TEX is that its paragraph-breaking algorithm does not provide mechanisms for doing things like "leave and carry three" or higher. In cases like this the database extraction program must provide the information to TEX on where to allow it to break. One way to do this is to output each entity as a vbox. If TEX is in vertical mode then a series of vboxes cannot break if there is no glue between them. Let the database extraction program count the number of records and insert the glue and penalty commands at the allowed breakpoints. Another option is for the database program to only output the number of records and let TEX count the records as it processes them add glue at the appropriate points. So even though TEX does not provide the capability to "leave and carry three" this can easily be done in a content management system by having the database extraction program work with TEX.

Suppression and Selection

The macro capabilities of TEX provide powerful selection capabilities allowing TEX to act as a filter. For example, one application we developed typeset a large directory of phone and fax numbers. The sorting was complicated and the extraction from the database took several hours. We were able to produce two different books from the same composition file. The first book listed all the entries. The second book listed only those entities that had fax numbers. Because the books were generated from the same data file it was guaranteed that the order of the records in the books could not change.

TEX can also filter out visually redundant information. Normalization is the process of removing redundant information from a database. Extracting the information from a database will denormalize the data. Consider the following simple example. The database contains three tables: a department table containing the id number of the department and the name of the department, a person table containing the name and phone number of the person and a link table that connects the department and person tables. The SQL statement to select all the records from the tables for composition could look something like this:

```
SELECT DISTINCT
    D0.DeptId,
    D0.Department,
    D2.PersId,
    D2.First,
    D2.Last,
    D2.PersPhone
FROM
    "Department.DB" D0,
    "Link.db" D1,
    "Person.DB" D2
WHERE
    (D1.DeptId = D0.DeptId) AND
    (D2.PersId = D1.PersId)
ORDER BY
    D0.Department,
    D2.Last,
    D2.First
```

When the resulting answer table is output using descriptive tags the result would be a file looking like this:

```
\StartRecord
\DeptId{2}
\Department{Medlars}
\PersId{3}
\FirstName{Bilbo}
\LastName{Baggins}
\PersPhone{5678}
\EndRecord
\StartRecord
\DeptId{2}
\Department{Medlars}
\PersId{2}
\FirstName{Jeannine}
\LastName{McArthur}
\PersPhone{1234}
\EndRecord
\StartRecord
\DeptId{1}
\Department{Programming}
\PersId{1}
\FirstName{Jeffrey}
\LastName{McArthur}
\PersPhone{4200}
\EndRecord
```

All the fields are output into the composition file with a start and end tag for each record. It is usually undesirable to reprint duplicate information. Using combinations of `\def`, `\let`, and `\ifx`, TEX can filter out the duplicate information. Doing this type of filtering, or deduping, in TEX simplifies the generation of the composition file.

TEX can also do simple rearrangement of the data. Below is a set of simple macros that demonstrate these capabilities.

```
\let\StartRecord=\empty
\let\LastDept=\empty

\def\DeptId#1{\def\ValDeptId{#1}}
\def\Department#1{\def\ValDepartment{#1}}
\def\PersId#1{\def\ValPersId{#1}}
\def\FirstName#1{\def\ValFirstName{#1}}
\def\LastName#1{\def\ValLastName{#1}}
\def\PersPhone#1{\def\ValPersPhone{#1}}

\def\EndRecord{
    \ifx\LastDept\ValDeptId\else
        \let\LastDept=\ValDeptId
        \medbreak
        \line{\bf\ValDepartment\hfil}
    \fi
    \line{\ValLastName,
        \ValFirstName
        \hfil
        \ValPersPhone}
}
```

TEX could also be used to select which entries to typeset. The same data file could be used to generate a complete directory, and a separate directory for each department. For the complete directory TEX would set each record. For the department records TEX could test the department id.

Various Output Options

TEX generates DVI files. None of our end users were interested in DVI files. Using tools like DVIPS, and recently pdfTEX we have created PostScript and PDF files for our clients, as well as finished camera ready pages and film. We have used TEX to generate questionnaires to be programmatically faxed.

TEX generated faxes. The process of using TEX to fax questionnaires is outlined below:

1. The application generates a composition file.
2. The application executes TEX which composes the file and generates a DVI file. TEX generates an auxiliary file listing the account number, starting page
   and ending page of each questionnaire.
3. The application reads in the auxiliary file.
4. The application executes DVIPS to extract each questionnaire using the starting and ending page number from the auxiliary file. The name of the resulting PostScript file is based on the account number.
5. Acrobat Distiller converts the PostScript File into a PDF file and embeds all the needed graphics and fonts.
6. A second application watches the output directory from Acrobat Distiller. The file name, which is based on an account number, is used to look up the fax number. Using OLE, Acrobat Exchange is executed and the PDF file is opened. Using DDE and WinFax, the fax number is set. Using OLE, Acrobat Exchange is told to print the PDF to the Fax.

Caveats

Developing macros for database publishing requires a full understanding of all of the capabilities of TEX. Macro packages like LATEX are designed for authoring and they were not designed for database publishing. The macro writer must understand how to write output routines, use `\mark`, and even the list capabilities of TEX as described in Appendix D of *The TEXbook*. Compared to desktop publishing systems this is a very high threshold. Finding and training people to write macros of this complexity is a difficult task.

All the logic to typeset the pages must be programmed into TEX. The rules for when to break, when to kern, and so on must be incorporated into the macros. This means that the output pages are very regular. One often-heard complaint is "can't you change the typesetting in just this one case?" TEX is not WYSIWYG. This means that the user of a turn-key system that uses TEX as its typesetting engine may not allow any visual fine-tuning of the pages.

Although TEX is very robust, many DVI translation programs have problems with complex pages. We have seen numerous implementations fail trying to process large complex pages. Many DVI translation programs are "fussy" about the types of graphics they will use. Integration of "off-the-shelf" versions of TEX is relatively easy. Integration of "off-the-shelf" DVI translation programs has been problematic.

Most implementations of TEX and the related DVIware for the PC use environment variables. The tools assume that there is only one implementation of TEX being used. Configuration management is much more difficult than it should be.

The most serious disadvantage to using TEX as the back end system is the difficulty finding people with the aptitude to learn and understand how to write complex macros. It is relatively easy to find someone to use a desktop publishing system. Finding someone who can write the macros for a content management system is a lot harder. We are always looking for qualified people. The lack of qualified people is part of the overall shortage in the computer industry.

Conclusion

TEX has proven to be a valuable tool in developing content management systems, but it cannot be quickly adopted. It takes time and commitment to find, hire, and train the people needed to develop the macros. There are disadvantages to using TEX as the typesetting engine, but the disadvantages are compensated for by the quality and speed of the resulting system.

Presenting Mathematics and Languages in Web-pages, using LaTeX2HTML

Ross Moore
Mathematics Department
Macquarie University
Sydney, Australia 2109
ross@mpce.mq.edu.au
http://www-math.mpce.mq.edu.au/~ross/

Introduction

LaTeX2HTML is a very flexible tool for creating Web pages to display the information contained in a manuscript prepared using LaTeX. As of July 1998, the current version is LaTeX2HTML v98.2. It runs under Unix, Linux, Windows NT, Windows'95, OS/2 and DOS. The latest released version, with online manual for browsing, can be obtained from its distribution site[1], in the USA or the European mirror[2]. On CTAN, look under `support/latex2html`. There is a developers repository[3] for minor updates and (α- or β-) development versions.

In the following sections we first discuss some general considerations for Web pages using HTML, including some pragmatic tips for authors wishing to use the LaTeX2HTML translator. This is followed by a study of the different ways that are available for the presentation of mathematics using LaTeX2HTML, discussing the available options and when a particular approach may be most appropriate. Further examples are presented on how to use LaTeX2HTML to produce multi-lingual documents.

General considerations: Why HTML?

For distribution of text-like data on the Internet the HTML formats, in their various versions, are very efficient and widely supported in Web browser software on all computing platforms. Thus a converter that produces documents using HTML can guarantee that the information to be presented is accessible to the widest possible audience. Furthermore, there is no requirement for 'plug-in' modules or other special software, beyond what is normally available with a Web browser.

Since an `.html` file contains just editable text, it is easily modified in any editor. This property alone adds a significant level of flexibility to any translation tool. If the result of the automatic translation is not quite what is desired, it is a simple matter to find the place where a correction is necessary and do it 'by hand'.

Even if the translation is flawless, at some time in the future there may be a change desired in the information being presented. For example a name or address may change, or a different graphic image may be desired, or a hyperlink to some external site may become invalid so needing to be replaced by another. Such minor alterations and updates can be done *without the need to reprocess the whole document* from the original LaTeX source (which may no longer even be available).

Another flexible aspect of HTML is that the reader has control over the browser window's characteristics. This includes size, style and colour of the text-font in which most of the information is to be presented, as well as the location and shape of the viewing window. The reader can customise these properties to suit personal requirements and preferences. This is a feature not available with other data formats, such as `.dvi`, `.pdf` or PostScript.

Mathematics with LaTeX2HTML

Figure 1 shows how pieces of mathematics may be presented, using LaTeX2HTML's default settings for the versions released during 1997 and 1998. This is a 'screen-shot' of a portion of a Web page generated using LaTeX2HTML. Fuzziness in the image is due to the lower resolution for on-screen display than is typically used with a printed version. Furthermore "anti-aliasing" is used with the font characters, to avoid a jagged appearance.

The LaTeX code for this example is given at the end of this article. It displays many common features of typeset mathematics:

- Greek letters and calligraphic (script) symbols;
- superscripts, subscripts, fractions and derivatives;
- large operators, such as $\int$ and $\sum$ with limits;

[1] `http://www-dsed.llnl.gov/files/programs/unix/latex2html/`
[2] ftp://ftp.rzg.mpg.de/pub/software/latex2html/
[3] http://cdc-server.cdc.informatik.tu-darmstadt.de/ latex2html/

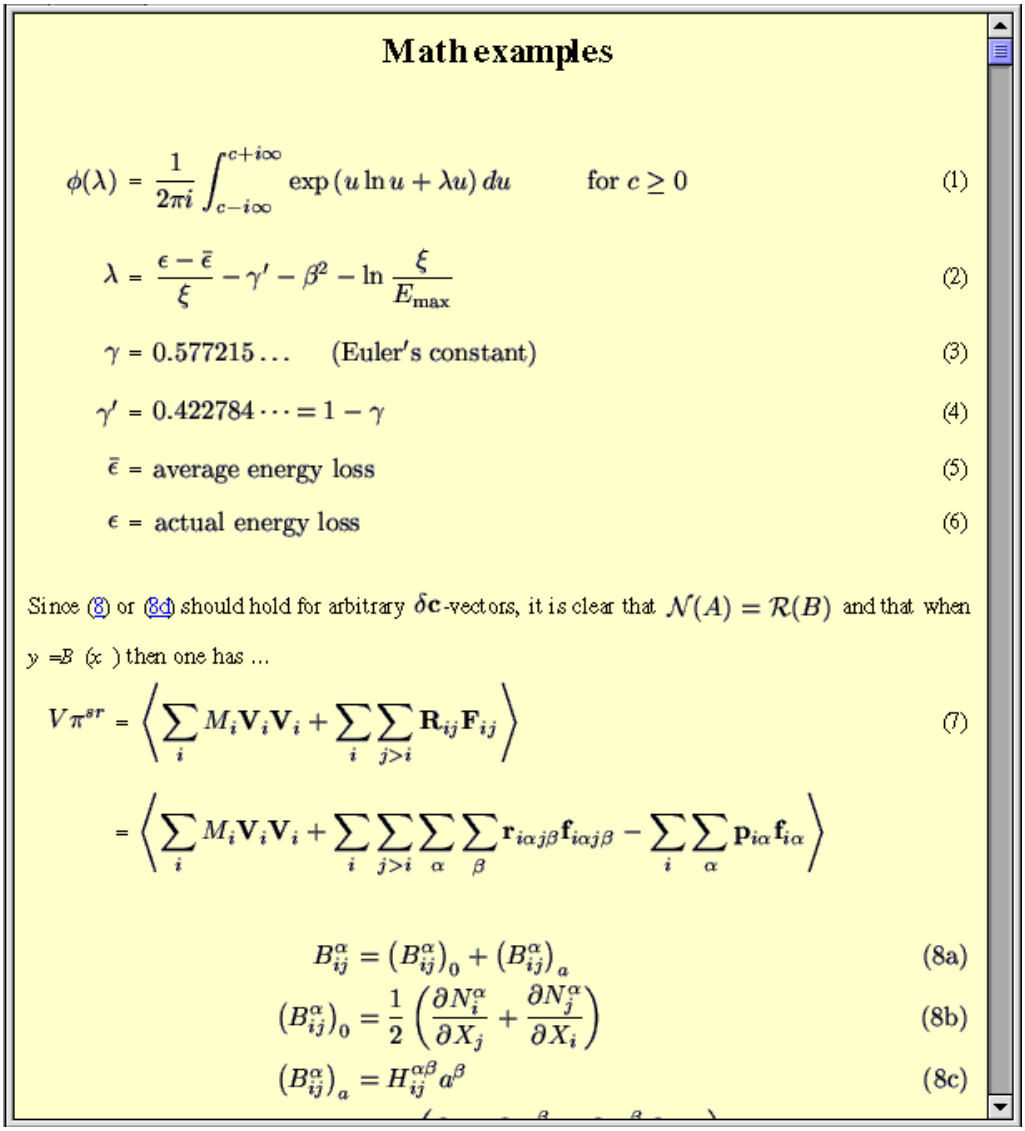


Figure 1: Some mathematics in a Web document produced by LaTeX2HTML, using the default settings.

- aligned arrays of equations (in particular the `subequations` environment from the `amsmath` package);
- extended brackets and parentheses.

Notice how most of the mathematics looks just like it has been typeset in TeX, because that is precisely what has happened. These expressions are actually images, in GIF format.[4] This is achieved by typesetting each mathematical expression on a single page, processing the `.dvi`-file using *dvips*, then rendering the resulting PostScript[5] files using *Ghostscript*.[6] For a better quality on-screen appearance in these low-resolution images, the 'anti-aliasing' technique is employed to soften the edges of otherwise 'blocky' font characters. For readability on-screen, images of mathematical expressions are normally made to correspond to a 14 pt font-size. Like most choices in LaTeX2HTML, this can be altered.[7]

[4] Alternatively images can be generated using the PNG graphics format.

[5] PostScript is a registered trademark of Adobe Systems Inc.

[6] *Ghostscript* is a product of Aladdin Enterprises, Menlo Park, CA. Version 4.02 or later is required for 'anti-aliasing' effects.

[7] ... by adjusting the value of the `$MATH_SCALE_FACTOR` configuration variable.

'Simple math'. Some of the mathematical expressions in figure 1 do *not* use an image; e.g., the inline expression $y = B(x)$ and most of the '=' signs in the first alignment. For these the whole expression can be represented using ordinary font characters, so this is what is done—with names set in italics, of course.

If any special symbol, indeed any macro (apart from those listed below), occurs within a mathematical expression then an image is made of the whole expression. To LaTeX2HTML users this is known as the "simple math" strategy. Superscripts, subscripts and some simple type-face macros are handled appropriately. Allowable macros include `\mathbf`, `\mathrm`, `\mathtt`, `\mathit` and `\boldsymbol`, as well as the recent `\mb` addition to LaTeX. Furthermore `\textbf`, `\textrm`, `\texttt`, `\textit` are allowable, but not recommended.

Although the appearance of expressions presented using 'simple math' are generally not as attractive as with an image, the benefit is that less information needs to be transferred across the network. For example, the expression $\mathcal{N}(A) = \mathcal{R}(B)$ results in HTML code:

```
<!-- MATH: $\mathcal{N}(A) = \mathcal{R}(B)$ -->
<IMG
 WIDTH="104" HEIGHT="31" ALIGN="MIDDLE" BORDER="0"
 SRC="img14.gif"
 ALT="$\mathcal{N}(A) = \mathcal{R}(B)$">
```

Notice how the TeX source is included as a comment. This ensures that the information is available in the `.html` file, in case the image fails to load successfully. When sufficiently short, the source is also included within the `ALT` attribute of the `<IMG>` tag. This allows a textual representation to be shown by browsers which do not support images (e.g., *lynx*) or when image support has been deliberately disabled.

Compare this with the amount of code generated for $y = B(x)$, using 'simple math':

```
when <I>y</I>=<I>B</I>(<I>x</I>) then one has
```

It is not difficult to appreciate the advantages to this 'simple math' approach. Furthermore, in the previous paragraph there was no mention of the actual size of the file `img14.gif` that needs to be transferred to show the image, and the extra server-connection required to request it be sent. In practice these are more significant than the extra text required within the `.html` file. Furthermore images do not rescale automatically when the font-size is changed within the browser.

It is clear that 'simple math' is a good strategy when a Web document contains only simple mathematical expressions, for then the overhead to request and transfer images is minimal. However when a lot of quite complicated mathematics is to be presented, this approach is not ideal. We later discuss alternative strategies available with LaTeX2HTML.

Alignment environments. Equation alignments are achieved using HTML's `<TABLE>` tag. Such tags became available as a standard part of HTML with the version 3.2 recommendation in early 1997. Some browsers provided support earlier than this.

Each cell in the table is treated as a separate expression, for deciding whether to use "simple math" or to make an image. Compare the different size of the '=' signs in the equation beginning $\gamma' = \ldots$ in figure 1. The first uses the browser's font whereas the second one is part of an image.

Notice also that equation numbers are placed in a separate column of cells within the `<TABLE>`. The `leqno` document-class option causes numbering to be put on the left-hand side, as with LaTeX.

Overriding 'simple math'. The default 'simple math' strategy can be turned-off using the `-no_math` command-line switch. That is, run LaTeX2HTML on the LaTeX source file using the command:

```
latex2html -no_math  ...  myfile.tex
```

where the '...' indicate the possible presence of other command-line switches. This will provide a consistent style for the mathematical expressions in all parts of the environment, as in an on-paper typeset version. This can be seen in figure 2, showing the inline mathematics portion of figure 1.

Since (8) or (8d) should hold for arbitrary $\delta\mathbf{c}$-vectors, it is clear that $\mathcal{N}(A) = \mathcal{R}(B)$ and that when $y = B(x)$ then one has ...

Figure 2: Inline mathematics, without using the 'simple math' strategy. An opaque background shows the size of images; extra space is included to allow correct alignment. (Normally image backgrounds are transparent.)

There will be more images than when 'simple math' is used. An appropriate situation for this strategy might be when the complete HTML document is available on the local machine or network (LAN), so that expensive file-transfers are not an issue.

Normally images are created with transparent backgrounds.[8] In figure 2, an opaque background has been used to show the size and alignment of the images, with respect to the surrounding text. Notice

[8] This is overridden by the `-no_transparent` command-line switch.

that when there is a 'descender' the image contains extra white space below the baseline. This allows the `<IMG ... ALIGN="MIDDLE" ...>` attribute to position the image correctly. With no descender `<IMG ... ALIGN="BOTTOM" ...>` is appropriate.

The extra height causes wide line-spacing in older browsers. This anomaly can be fixed for more recent browsers, by using the `.css` stylesheet[7] that LaTeX2HTML produces automatically. One needs to set the `line-height` property to a fixed amount; e.g.,

```
P.INLINE        { line-height : 20pt }
```

Now within the `.html` pages, change the `<P>` tag to `<P CLASS="INLINE">` for paragraphs containing over-sized images. The technique was used with figure 8. Future versions of LaTeX2HTML will handle this automatically, at least when preparing code according to HTML 4.0 specifications.

Images of aligned environments. In earlier versions of LaTeX2HTML an image was made of whole `eqnarray` and `equation` and other environments. Before `<TABLE>` tags were recommended within the HTML 3.2 standard, this was necessary and equation numbering was included as part of the image. Now this effect can be achieved, when desired, in several different ways.

Easiest is to request that LaTeX2HTML produce HTML code conforming to the version 2.0 standard, using the command-line option:

```
latextohtml -html_version 2.0  ...  myfile.tex
```

However this will disallow other constructions; e.g. forcing images also of `tabular` environments. Using also `-no_math` ensures images of all inline formulae as well.

Alternatively, images can be forced selectively by including an `\htmlimage` command within the environment. This command takes an argument which allows extra graphic effects to be specified for the image; see the User Manual[1] for the available effects:

```
\begin{eqnarray}
\htmlimage{}
...
...
\end{eqnarray}
```

Finally, the `makeimage` environment creates an image of whatever LaTeX code it contains. Both this and the `\htmlimage` command require the `html` package be loaded within the document preamble.

```
\begin{makeimage}
\begin{eqnarray}
...
...
\end{eqnarray}
\end{makeimage}
```

Image Reuse and Reduction Strategies

A document such as a research paper, thesis or class notes, can require a lot of mathematics. This can lead to many images. LaTeX2HTML automatically detects when LaTeX code is essentially identical to that used for an image already occurring within the document. A single image serves all such instances. However, even with this 'image reuse' the total number of images can still be large, giving significant loading delays.

math extension. One way to reduce these effects is to create *more* images, but of smaller pieces of mathematics. The idea is to extend the 'simple math' idea to use the text-font whenever possible. Only when a symbol or sub-expression cannot be represented adequately using the text-font is an image made. Any given HTML page can be expected to contain more images this way, however the same image may occur in many places on that page. The total size (in bytes) needed for images is reduced significantly, compared to when images are made of complete expressions.

Typically the first page is slow to load, as the images are downloaded across the network. Later pages in the same document require less download time as most of the required images will have been cached locally by the browser, from being present within earlier pages.

To activate the extra processing required for this strategy one must load LaTeX2HTML's special `math` extension, as follows:

```
latex2html -no_math -html_version 3.2,math  ...
```

UNICODE fonts. Further reduction in the number of images is obtained by presuming that the browser will provide at least limited support for the UNICODE font encoding[9]. In particular there should be support for Greek letters, both upper and lower-case, and some extra mathematical symbols.

To activate this, append the `unicode` extension to the `-html_version` command-line switch (don't leave any spaces):

```
... -no_math -html_version 3.2,math,unicode ...
```

Compare figure 4 with figure 3 to see the effect. This strategy is not yet ideal; notice the different styles of ϵ with and without the overline accent in the lower equations. Use of `\varepsilon` within the LaTeX source alleviates this discrepancy; alternatively it may become possible for browsers to render accented UNICODE characters.

[9] This is the case with the most recent versions of the *Netscape Navigator* and Microsoft's *Internet Explorer*

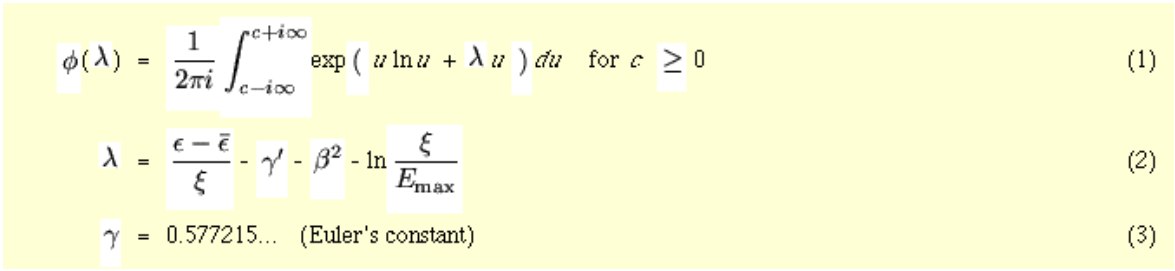


Figure 3: With the math extension loaded extra parsing of mathematics produces a mix of font-characters and smaller images. Opaque image backgrounds are used here only to show clearly which parts are images. In normal use these backgrounds are transparent.

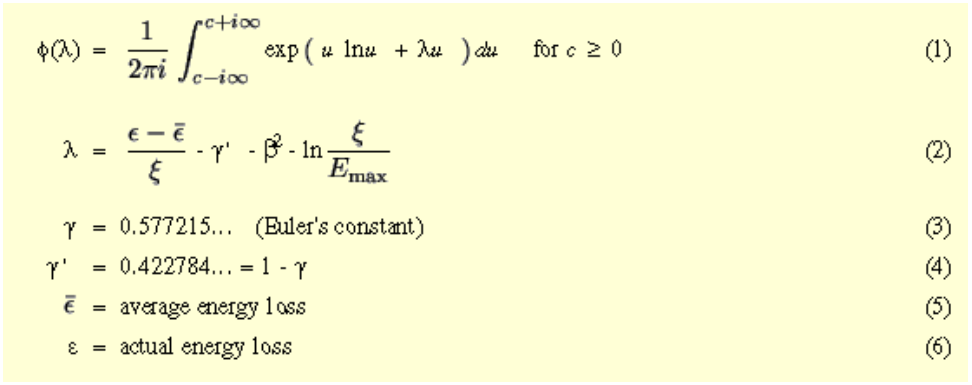


Figure 4: When the unicode extension is also loaded, Greek letters and other symbols can use font-characters also. This requires the browser to have some support for UNICODE.

Browser inadequacies. The commonly available Web browsers are continually improving, as more of the HTML 4.0 recommendations[7] are implemented. However some aspects of less advanced effects still create difficulties.

Look at the placement of superscripts and subscripts within figure 6. In mathematics these should be positioned above one another, as in B_{ij}^{α}. Furthermore the browser places extra space after italiced text. This is clearly evident in figure 5.

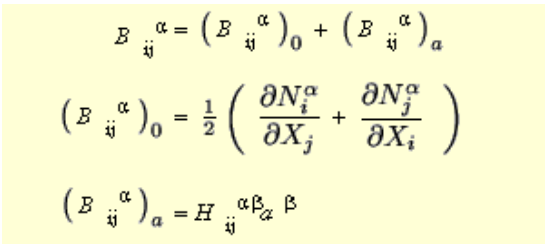


Figure 5: A browser's placement of multiple superscripts and subscripts is not always ideal for mathematical usage.

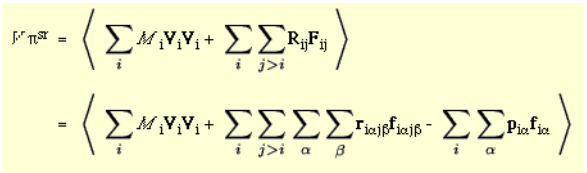


Figure 6: Some browsers place extra space after italiced text, over-compensating for the slope. This is particularly awkward for placing subscripts.

Such details should be fixed in future releases of browser software. Alternatively it may become possible to overcome these deficiencies within the HTML code, by specifying 'box-like' placement properties with a CSS style-sheet[7]. This requires browsers to support these advanced features.

Future support for MathML. As support for the new XML[7] (similar to HTML but more versatile) is incorporated into Web-browsers, it will become appropriate to extend the translation capabilities of LaTeX2HTML. In particular, an ability to prepare

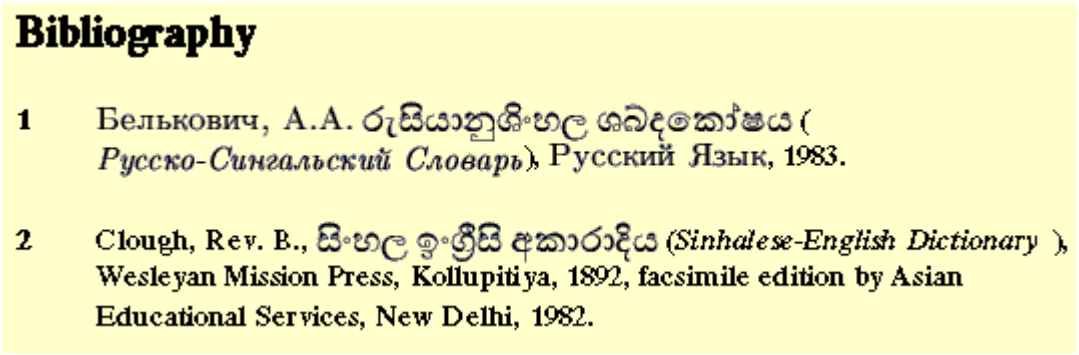


Figure 7: Bibliography entries using images for text of non-Latin based alphabets.

mathematics according to the MathML[7] markup scheme is a goal for future development.

Multi-lingual documents

Representing different languages within the same Web document presents problems similar to those with mathematics. There is no real difficulty when the languages are all based on the latin alphabet, provided any required accented letters are all available within a single font encoding.

The ISO-8859 encodings contain complete character-sets for various languages. Modern browsers provide support for Web pages having some of these as the designated character-set. LaTeX2HTML has specific support to produce pages using Latin-1, ... , Latin-6 (i.e. ISO-8859-1, 2, 3, 4, 9 and 10). A particular character set is specified using an extension to the `\html_version` command-line switch. This is fully compatible with other extensions; e.g.

```
... -html_version 3.2,latin2,math,unicode ...
```

Images of special fonts. A single encoding rarely suffices when non-Latin languages are also required. Using images is a convenient strategy. Figure 7 shows some bibliographic entries[10] using characters from the cyrillic alphabet and sinhalese script.

LaTeX2HTML recognises TeX's `\font` command as declaring a macro that will require an image to be made of enclosing environment. For example the cyrillic text was produced using:

```
\font\wncyr = wncyr at 10pt
...
...
{\wncyr Bel\char126koviq, A.A.} ...
```

Pre-processing for exotic scripts. The sinhalese script in figure 7 was generated in a similar way to the cyrillic, but only *after* the source is filtered through Haralambous' *Indica* preprocessor, part of 'Sinhala–TeX'[3]. After pre-processing, the LaTeX source contains parts like:

```
{\SHb\char29a\char8}{\SHb\-\char69i}{\SHb...
```

in which each grouping generates an image for the appropriate letter or syllable. This is acceptable for small pieces of text in the exotic script. However many images are needed when there are whole paragraphs and pages of the script.

Automatic pre-processing. In figure 8 we see a portion in which each paragraph is presented as a separate image. One way is to use the `makeimage` environment, as was done with mathematics.

A better way is to use LaTeX2HTML on the manuscript, *before* pre-processing with *Indica*. Since the alphabets do not map one-for-one with the latin alphabet, a transliteration or transcription scheme is employed. Multi-letter combinations correspond to single letters or syllables in the exotic language. Portions of the manuscript using such schemes are included with the other parts to be rendered as images, just as with pieces of mathematics. The difference is that these portions need not be valid TeX code, requiring pre-processing first. This is done as an extra step prior to image-generation.

Systems have been devised for the typesetting of various languages using TeX, after first using such a pre-processing step. A suite of packages for LaTeX and appropriate implementations for LaTeX2HTML, known as IndicTeX/HTML[4], automate this process with some of the pre-processors available for Indic languages and traditional scripts. This includes support for Avinash Chopde's 'ITRANS' preprocessor [5] which handles many different languages and transliteration schemes.

The pre-processor was used this way for the page from which figure 8 was extracted. Some of

[10] These are taken from a LaTeX2HTML conversion of the 'Sinhala-TeX' documentation[3], available at: `http://www-texdev.mpce.mq.edu.au/l2h/indic/Sinhala/lreport/`

the HTML coding is shown in figure 9. Notice how the original transliteration is included as a comment. Just as with mathematics, this ensures the information is available even when the image fails to render.

UNICODE fonts, Ω and Λ. As UNICODE becomes more widely used, it should become possible to use its extensive range of characters, instead of images. Furthermore, it should become possible to employ Ω[2][6], via its LaTeX variant Λ, in conjunction with LaTeX2HTML. It could be used for several tasks:

- as the pre-processing engine;
- replacing LaTeX for the typesetting necessary when producing images;
- to generate UNICODE font-entities.

References

[1] Nikos Drakos & Ross Moore, "The LaTeX2HTML Translator". Documentation and User Guide accompanying the software; online version at `http://www-dsed.llnl.gov/files/programs/unix/latex2html/manual`.

[2] Yannis Haralambous & John Plaice, "ΩTimes and ΩHelvetica Fonts Under Development: Step One", *TUGboat*, The Communications of the TeX Users Group, Volume 17, No. 2 (1996) pp. 126–146.

[3] Yannis Haralambous & Dominik Wujastyk, "A Sinhalese TeX System", documentation for 'Sinhala–TeX' and the *Indica* preprocessor, 1994; available at `http://ctan.tug.org/ctan/tex-archive/languages/sinhala/`.

[4] Ross Moore, "IndicTeX/HTML, Traditional Scripts within Web-pages", to appear in: TUG–India, volume 1, 1998; online version available at `http://www-texdev.mpce.mq.edu.au/indic/IndicHTML/`.

[5] Avinash Chopde, *ITRANS* "Indian Language Transliteration Package", A package for printing text in Indian Language Scripts, available from `http://www.aczone.com/itrans/`.

[6] John Plaice & Yannis Haralambous, "The Latest Developments in Ω", *TUGboat*, The Communications of the TeX Users Group, Volume 17, No. 2 (1996) pp. 181–183.

[7] World Wide Web Consortium, online site at `http://www.w3c.org/Consortium/`;
HTML 4.0: `http://www.w3c.org/Markup/`
Stylesheets: `http://www.w3c.org/Style/`
MathML: `http://www.w3c.org/Math/`
XML: `http://www.w3c.org/XML/`

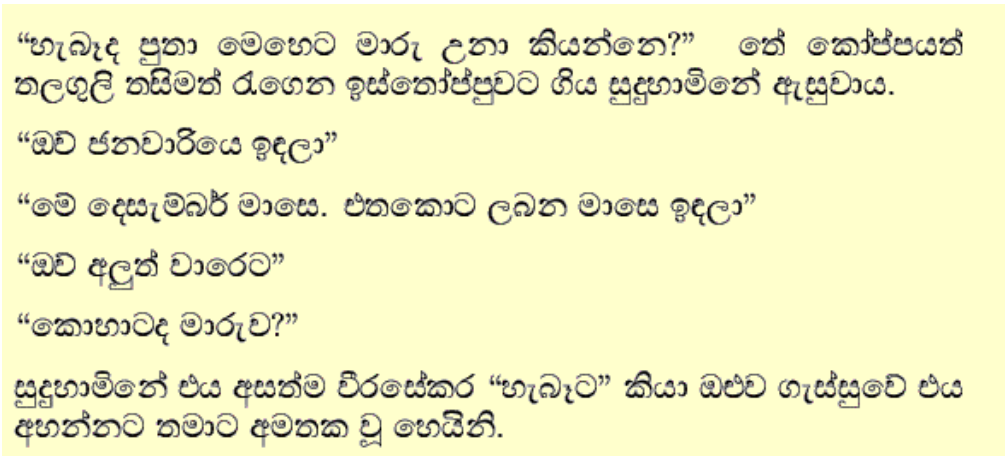


Figure 8: Single images are made of whole paragraphs, when pre-processing is delayed until the image-generation phase.

```
<!-- INDICA S
 ‘‘e˜ka nambuyine putha˜’’ vi˜rase˜kara katha˜va patan gaththe˜ nodhannekuta
yamak kiya˜ dhena paridhdheni.
 -->
<P><IMG
 WIDTH="554" HEIGHT="45" ALIGN="BOTTOM" BORDER="0"
 SRC="img4.gif"
 ALT="\lq\lq e˜ka nambuyine putha˜’’ vi˜rase˜kara katha˜va patan gaththe˜ nodhannekuta
yamak kiya˜ dhena paridhdheni."></P>

<!-- INDICA S
 ‘‘e˜ka nambuyi. koLa"mba ugannanava kiyandath puLuvan. i˜tath koLa"mba loku
isko˜lavalata enne loku lokkange Lamayi. e˜ Lamayi thama˜ issarahata ho"ndha
tha^nakata enne. i˜tath e˜ Lamayinge ma˜rgayen puLuvan e˜ Lamayinge
tha˜ththalagen o˜na^˜ va^dak karava ganna’’
 -->
<P><IMG
 WIDTH="558" HEIGHT="108" ALIGN="BOTTOM" BORDER="0"
 SRC="img5.gif"
 ALT="\lq\lq e˜ka nambuyi. koLa’’mba ugannanava kiyandath puLuvan. i˜tath koLa’’mba lo
ku...
...ma˜rgayen puLuvan e˜ Lamayinge tha˜ththalagen o˜na^˜ va^dak karava ganna’’"></P>

<!-- INDICA S
 ‘‘ballata dha˜mu. ballata. ballata’’ baladhe˜va sina˜suNe˜ya.
 -->
<P><IMG
 WIDTH="433" HEIGHT="22" ALIGN="BOTTOM" BORDER="0"
 SRC="img6.gif"
 ALT="\lq\lq ballata dha˜mu. ballata. ballata’’ baladhe˜va sina˜suNe˜ya."></P>
```

Figure 9: HTML code produced for some of the paragraphs of Sinhalese shown in figure 8, using a standard transliteration and preprocessed by *Indica*.

LaTeX code for figure 1

The following LaTeX code is adapted from pieces of coding provided by Michael Hall[11] and Michel Goossens,[12] for testing during the development of certain aspects of the mathematics support within LaTeX2HTML.

```
\documentclass[a4paper]{article}
\usepackage{html, amsmath, array, alltt}
\usepackage[dvips]{color}
% ensure \bm is defined if not latest LaTeX
%begin{latexonly}
\providecommand{\bm}[1]{\mathbf{#1}}
%end{latexonly}
\begin{imagesonly}
\providecommand{\bm}[1]{\mathbf{#1}}
\end{imagesonly}
\newcommand{\Range}{\mathcal{R}}
\newcommand{\Ker}{\mathcal{N}}
\newcommand{\Quat}{\vec{\mathbf{Q}}}
\renewcommand{\d}{\partial}

\begin{document}
\htmlhead[center]{section}{Math examples}

\begin{eqnarray}
\phi(\lambda )&=& \frac{1}{2 \pi i}\int^{c+i\infty}_{c-i\infty}
\exp \left( u\ln u+\lambda u \right)du \hspace{1cm}\mbox{for } c\geq 0\\
\lambda & = & \frac{\epsilon -\bar{\epsilon}}{\xi}-\gamma' -\beta^2 -\ln\frac{\xi}{E_{\rm max}} \\
\gamma  & = & 0.577215\dots \mathrm{\hspace{5mm}(Euler's\ constant)}\\
\gamma' & = & 0.422784\dots = 1 - \gamma \\
\bar{\epsilon}& = & \mbox{average energy loss} \\
\epsilon      & = & \mbox{actual energy loss}
\end{eqnarray}

Since~\eqref{bgdefs} or~\eqref{gdef} should hold for arbitrary
$\delta\bm{c} $-vectors, it is clear that $\Ker(A) = \Range(B)$ and
that when $y=B(x)$ then one has ...

\begin{eqnarray}\label{eqn:stress-sr}
 V \bm{\pi}^{sr} & = & \left<
  \sum_i M_i \bm{V}_i \bm{V}_i + \sum_i \sum_{j>i} \bm{R}_{ij} \bm{F}_{ij} \right> \\
\nonumber & = & \left< \sum_i M_i \bm{V}_i \bm{V}_i
  + \sum_{i} \sum_{j>i} \sum_\alpha \sum_\beta \bm{r}_{i\alpha j\beta} \bm{f}_{i\alpha j\beta}
  - \sum_i \sum_\alpha \bm{p}_{i\alpha} \bm{f}_{i\alpha} \right>
\end{eqnarray}

\begin{subequations}\label{bgdefs}
\begin{align}
 B_{ij}^\alpha & = \left(B_{ij}^\alpha\right)_0 + \left(B_{ij}^\alpha\right)_a \label{bdef} \\
 \left(B_{ij}^\alpha\right)_0 & = \frac{1}{2}\left(
   \frac{\d N_i^\alpha}{\d X_j} + \frac{\d N_j^\alpha} {\d X_i} \right)\label{b0def} \\
 \left(B_{ij}^\alpha\right)_a & = H_{ij}^{\alpha \beta} a^\beta \label{budef} \\
 H_{ij}^{\alpha \beta} & = \frac{1}{2}\left(
   \frac{\d N_k^\alpha}{\d X_i} \frac{\d N_k^\beta}{\d X_j}
   + \frac{\d N_k^\beta}{\d X_i} \frac{\d N_k^\alpha}{\d X_j} \right)\label{gdef}
\end{align}
\end{subequations}

\end{document}
```

[11] Dr. Michael L. Hall, Los Alamos National Laboratory.
[12] Dr. Michel Goossens, IT Division, CERN, Geneva

BibTeX 101

Oren Patashnik
10388 Rue Riviere Verte
San Diego, CA 92131
opbibtex@cs.stanford.edu

Abstract

This paper introduces BibTeX to those having little or no previous BibTeX experience but having at least some familiarity with TeX or LaTeX. It also answers some frequently asked BibTeX questions, from complete novices as well as from experienced users.

Introduction

BibTeX is the bibliography program designed originally to accompany Leslie Lamport's LaTeX; it now works with other incarnations of TeX, too. BibTeX removes the tedium, and adds some flexibility, in producing a reference list.[1] When BibTeX creates your reference list, it's BibTeX, not you, minding the minutiae like ensuring that your reference-list entries are in the correct order, that every comma is in place, and that the information is formatted consistently across entries. Furthermore, a single, simple, change of bibliography-style name lets you convert your reference list from style A (which might order the entries alphabetically, spell out journal names in full, and list all authors as first-name then last-name), to a completely different style B (which might order the entries according to their order of mention in the text, abbreviate journal names), and invert just the first author's first and last names).

The next section of the paper explains how to use BibTeX. The final section answers some frequently asked BibTeX questions.

Getting Started with BibTeX

To use BibTeX, you first put your bibliographic information into a bibliography database file. For example, your file `mybib.bib` (all database file names end with `.bib`) might contain an entry like:

```
@BOOK{knuth:tex,
  author = "Donald E. Knuth",
  title = "The {{\TeX}}book",
  publisher = "Addison-Wesley",
  year = 1984,
}
```

The `@BOOK` tells BibTeX that this is a book entry type. The `knuth:tex` is the database key, which is a sequence of characters to be used as the name for this entry. And the rest of the entry comprises four ⟨field⟩ `=` ⟨field-value⟩ pairs appropriate for a `BOOK` entry type. In general you will have many such entries in a database file; you might also have multiple database files.

Once you've entered the bibliographic information into the database file(s), the hard part is done. For the easy part, you put into your (La)TeX[2] source file citations like

```
... in the \TeX{}book~\cite{knuth:tex} ...
```

The `\cite` command's argument here, `knuth:tex`, is called a cite-key, and must match the corresponding database-key. (La)TeX might typeset this `\cite` command as

... in the TeXbook [23] ... or
... in the TeXbook[23] ... or
... in the TeXbook (Knuth, 1984) ...

depending on the citation style. (La)TeX's default citation style uses a number in brackets, and for that citation style, together with an appropriate bibliography style, the corresponding reference-list entry might look like:

23. Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984.

Besides the citation commands, you also put into your (La)TeX source file two BibTeX-related commands:

```
\bibliography{mybib}
\bibliographystyle{plain}
```

The `\bibliography` command does two things; it tells (La)TeX to put the reference list at that spot in your document, and it tells BibTeX which file(s) to

[1] Throughout this paper, the term 'reference list' is used generally to refer to what might also be called a 'bibliography' or a 'list or sources' or anything similar.

[2] The term '(La)TeX' is used to mean either LaTeX or plain (or other variations of) TeX.

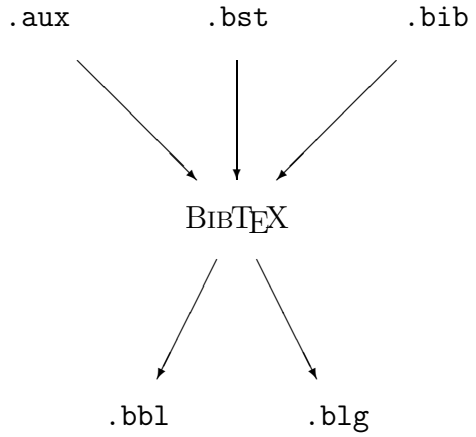


Figure 1: BibTeX's input and output files.

use for the bibliographic database, here just the single file `mybib.bib`. The `\bibliographystyle` command tells (LA)TeX nothing, but tells BibTeX which bibliography style to use, here the standard style `plain`; bibliography style file names end with `.bst`, thus the relevant file is `plain.bst` in this case.

So with your database file(s) and your (LA)TeX source file structured appropriately, your citations are formatted according to the citation style, and your reference list is formatted according to the bibliography style.

To actually produce the typeset document, you run (LA)TeX, BibTeX, (LA)TeX, (LA)TeX. The first (LA)TeX run writes, to an `.aux` file, information for use by BibTeX — which bibliography style to use, which database file(s) to use, and which database entries to include. The BibTeX run reads all that information from the `.aux` file, reads the specified database (`.bib`) file(s), formats the reference list according to the instructions in bibliography style (`.bst`) file, and writes its output onto a `.bbl` file. The next (LA)TeX run reads the `.bbl` file and incorporates the reference list into the document. The final (LA)TeX run fixes the references into the reference list. Figure 1 shows the files that BibTeX uses. The `.blg` file is BibTeX's log file, in which BibTeX records any warning or error messages.

To try using BibTeX with LaTeX, put the six-line `BOOK` entry shown on the previous page into a file called `mybib.bib`, and then, into a file called `mypaper1.tex`, put these six lines of LaTeX:

```
\documentclass{article}
\begin{document}
The \TeX{}book~\cite{knuth:tex} is good.
\bibliography{mybib}
\bibliographystyle{plain}
\end{document}
```

Exactly how you run LaTeX and BibTeX is system-dependent, but on my system I type four commands:

```
latex mypaper1
bibtex mypaper1
latex mypaper1
latex mypaper1
```

To try using BibTeX with plain TeX, create the file `mybib.bib` as above, and then put into a file called `mypaper2.tex` these seven lines of plain TeX:

```
\input btxmac
The \TeX{}book~\cite{knuth:tex} is good.
\medskip
\leftline{\bf References}
\bibliography{mybib}
\bibliographystyle{plain}
\bye
```

To run `mypaper2` through TeX and BibTeX on my system I simply type

```
tex mypaper2
bibtex mypaper2
tex mypaper2
tex mypaper2
```

But `mypaper2` `\input`s the file `btxmac.tex`, which contains the macros that make BibTeX work with plain TeX. Those macros are a standard part of most TeX distributions, but if they're not a part of yours, you'll have to go fetch a copy from CTAN in `tex-archive/macros/plain/contrib/`.

That's a brief introduction to BibTeX. The following sources provide further details. Leslie Lamport's LaTeX manual [3] explains how to use BibTeX with LaTeX. In particular, section B.1 describes the `.bib`-file format in detail. The file `btxmac.tex` [1] documents its own use, with or without Karl Berry's `eplain.tex` package (for which the `btxmac` macros were originally written). The "BibTeXing" document [4], which is distributed along with BibTeX itself, contains further hints for BibTeX users. The "Designing BibTeX Styles" document [5], also distributed with BibTeX, explains the postfix stack-based language used to write BibTeX bibliography styles (`.bst`) files. *The LaTeX Companion* [2], by Michel Goossens, Frank Mittelbach, and Alexander Samarin, summarizes much of the information contained in the sources above, and it describes some of the tools available for helping with BibTeX bibliographies. Norman Walsh's *Making TeX Work* [7] also describes such tools. (Many users find the tools for managing bibliographic database files to be particularly useful.) BibTeX's standard bibliography styles, like `plain`, are based on Mary-Claire van Leunen's *A Handbook for Scholars* [6]. That book is

worthwhile reading for anyone wanting to design a bibliography style.

Frequently Asked BibTeX Questions (FABQs)

The questions in this section are ordered, roughly, by user sophistication, with the earlier questions coming from the least experienced users.

FABQ: Can I include an entry in the reference list without having to give an in-text citation for it?

Answer: Yes. If there's a `\nocite{my-ref}` in your (LA)TeX source file, the entry whose database-key is `my-ref` will appear in the reference list but without a corresponding in-text citation.

FABQ: Can I include all the entries in my database in the reference list without my having to `\cite` or `\nocite` all of them explicitly?

Answer: Yes. Putting a `\nocite{*}` command in your (LA)TeX source file has the effect of putting in that spot of your source file a `\nocite` command for each entry in your database.

FABQ: If I can't find a bibliography style to my liking, how can I make my own bibliography style (`.bst`) file?

Answer: The `.bst` language is fairly flexible, but it's meant to be programmed, except for simple changes, by reasonably experienced programmers. Patrick Daly's `custom-bib/makebst` package, on the other hand, allows nonprogrammers, too, to create their own bibliography styles.

FABQ: How can I have two different database files use the same set of abbreviations without duplicating the abbreviations?

Answer: If you put all your abbreviations, like

```
@STRING{A-W = "Addison-Wesley"}
```

into a database file, say `abrvs.bib`, containing just abbreviations, and if you list that file first in the `\bibliography` command, then all other `.bib` files listed in that command may use the abbreviations in `abrvs.bib`. For example, two files `cs-books.bib` and `math-books.bib` may have entries that use the field

```
publisher = A-W,
```

if the `\bibliography` command looks like

```
\bibliography{abrvs,cs-books,math-books}
```

FABQ: How can I keep BibTeX from converting all my journal-article titles to lower case?

Answer: Technically, it's the bibliography style file, not BibTeX itself, that's doing the case conversion. Many bibliography styles (*The Chicago Manual of Style*, for example) say that a reference-list entry for a journal article should have the article title converted to lower case, because it is a smaller thing inside a bigger thing, but should have the title of the bigger thing — the journal title itself — left in uppers-and-lowers form (in which you capitalize the first word, and, in most styles, the first word after a colon — which indicates a subtitle — and all other words except articles and unstressed conjunctions and prepositions). But if you don't like that style, it's a simple change to the `.bst` file to eliminate the case conversion. For example, many `.bst` files will have something like:

```
FUNCTION {format.title}
{ title empty$
    { "" }
    { title "t" change.case$ }
  if$
}
```

That's the function that converts the titles of, for example, journal articles, from uppers-and-lowers form to lowercase. Changing that function to

```
FUNCTION {format.title}
{ title field.or.null
}
```

will eliminate the case conversion.

FABQ: How can I change the citations from using brackets to using parentheses or superscripts.

Answer: Certain bibliography style (`.bst`) files have accompanying (LA)TeX style files; make sure you are using the accompanying (LA)TeX style file if it's required. For example, if you are using the `apalike` bibliography and citation style, which uses parentheses rather than brackets in its citations, you need, in addition to `apalike.bst`, either `apalike.sty` (under LaTeX) or `apalike.tex` (under plain TeX). You invoke those files with a

```
\usepackage{apalike}
```

command under LaTeX, or a

```
\input apalike
```

command under plain TeX. If there is no such accompanying (LA)TeX style file for your `.bst` file, you must redefine `\cite` and any other relevant citation command yourself.

FABQ: Sometimes I enter an author in my database file as

```
author = "D.E. Knuth",
```

but in my reference list the author appears without the middle initial, as just 'D. Knuth' — what's going on?

Answer: Probably you are using a bibliography style that automatically abbreviates first names to just initials. In this case, BibTeX thinks that 'D.E.' is a single name, rather than two initials, because there is no space between the initials, and the style abbreviates this to 'D.' The solution is to, in the database file, insert a space between the initials:

```
author = "D. E. Knuth",
```

If you really want to close up the space between initials in the output, it's a simple matter to change the bibliography style file to do that.

FABQ: How can I have other BibTeX questions answered?

Answer: Post them to the `comp.text.tex` newsgroup; I've been known to send private email replies to questions that seem to receive inadequate answers in that newsgroup.

References

[1] Karl Berry and Oren Patashnik. `btxmac.tex`. Macros to make BibTeX work with plain TeX; current version 0.99k, 13 November 1995.

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[3] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, second edition, 1994.

[4] Oren Patashnik. BibTeXing. General documentation for BibTeX users, contained in the file `btxdoc.tex`, 8 February 1988.

[5] Oren Patashnik. Designing BibTeX styles. Documentation for BibTeX style designers, contained in the file `btxhak.tex`, 8 February 1988.

[6] Mary-Claire van Leunen. *A Handbook for Scholars*. Oxford University Press, revised edition, 1992.

[7] Norman Walsh. *Making TeX Work*. O'Reilly & Associates, 103 Morris Street, Suite A, Sebastopol, CA 95472, 1994.

One-Document Scientific Publishing for Print and Web/CD

Peter Signell
Physics and Astronomy Department
Michigan State University
East Lansing, MI, 48824
signell@physnet.pa.msu.edu

One-content document, several auxilliaries

When a document must be published in more than one version and must also undergo periodic revision, the use of a different stored manuscript for each version may easily result in the versions getting out of step with each other, so it is considered good practice to make arrangements to have just one stored master version. This is especially important if the different versions vary greatly in the order in which the content elements are presented, as they are likely to do when one version is for the print medium and another for the Web or for CD-ROM. With all versions sharing the same master content document, each version must have its own auxiliary non-content documents specifying the version's unique architecture and formatting. Then any content revisions are made only to the master content document and any format revisions for a particular version are made only to the document which specifies the formatting of that particular version. This process is quite familiar to LaTeX document managers who often separate formatting from content in order to maintain uniform formatting throughout a document, throughout a product line, or across revisions. In LaTeX, the formatting information is usually placed in one or more auxiliary "style files", some of which are shared among different printed versions of the material or between different documents, a kind of "inheritance." In this paper we discuss some experiences with extending that publication model, of one master content document and auxiliary architecture and content documents, to the case of simultaneous print and Web publication. Along the way, we discuss the differences between the screen and print media and the implications for the auxiliary files, new linking opportunities in the Web version, and ways to move the content document toward compatibility with the new Extensible Markup Language, XML.[1]

[1] Commented links to documents on XML can be found at www.sil.org/sgml/xml.html and there are answers to frequently asked questions at www.ucc.ie/xml.

Overview of media differences

When a document is to be published both in print and on the Web, the formatted print and Web versions are likely to take rather different forms. This is because of the differing characteristics of the two media. For our present purposes, there are four main ways in which print presentation differs from computer-screen presentation: (1) the print medium has much higher resolution then the computer screen so scientific text can be packed much more densely in a print version; (2) the effective dimensions of the computer user's browser window can vary over a wide range, even at the whim of the user, in contrast to the totally controlled dimensions of book paper; (3) the computer is able to instantly present hidden material whenever the user asks to see it; and (4) the computer can have virtually unlimited amounts of material available for instant presentation.

Resolution-related differences

The limited resolution of the computer screen requires an increase in font size, particularly for the display of equations and math symbols, and this makes screen real estate particularly valuable. For example, one does not want to take up valuable space with task bars, sticky pads, and icons. In the computer-screen version, these functions can be provided through menus of choices that pop up when the right mouse button is pressed on a PC or when the shift key accompanies the mouse click on a Mac.

The fact that a very limited amount of material can be on the screen at any one time means that a figure should not float to the top or bottom of the page, as in LaTeX, but should instead be displayed next to the first reference to it in the text. The figure must also be available to be displayed at any point where it is referenced, since the small amount of material on the screen means that the figure is unlikely to still be on the screen at the time the user reads the reference. Also, because of the limited resolution, the user must be able to click on any figure to see an enlarged view that shows details with clarity sufficient to satisfy the user. A similar

kind of availability is necessary for equations and definitions: they must be actually on display at the first reference to each, and they must be available for display by the user at all further references.

Dimensional differences

We know quite well the size of paper on which a textbook will be printed, but we do not know the size of a computer user's browser window. Even if we know the dimensions of a particular user's screen, the user may shrink or expand the browser window at will in one or both dimensions. The user may increase the font size because of poor eyesight or limited screen capabilities. Any change in width or font size will produce a change in the number of characters allowed per line and so will require that the material on the screen be instantly and transparently reformatted. Another effect of a user narrowing the effective width of the browser window is that it will cause a figure caption to the right of a fixed-width narrow figure to be partially "off the screen to the right" unless the caption alone is instantly and transparently reformatted into lines of a narrower width alongside the figure. If the window is made too narrow, the caption must be seamlessly moved to a position underneath the figure and reformatted for that position. For equations, a good line-breaking algorithm must be used to allow equation formatting and reformatting to make the equation fit the screen size of the moment.[2]

Information-hiding differences

The computer has the unique ability to pop up information at the user's discretion, and this affects the placement of material in the flow of the document. For example, in printed textbooks the answer to a homework problem is never printed at the end of the problem because it would then be too easily seen by a user working the problem. Instead, in textbooks the printed answers to problems and exercises are almost always collected at the ends of the books. Other "optional" materials are collected away from the points at which they will be needed by some users but not needed by others. In the computer version, each of these elements can be made to pop up at the relevant point if the user so desires. In our case, these optional pop-up elements consist of specifically targeted help sequences and additional skill-based instructional elements and practice problems as well as the usual problem and exercise answers. Thus the computer-screen and print versions are very different in the flow and user-activated flow of document elements. In addition, there are proposals for "information that knows about me (my needs and preferences)" and this would require a multitude of possible paths through the kinds of information that may be available to construct a custom document. Finally, we note that a print version is limited in the amount of material that can be included because more information results in a higher price and a heavier weight, and sufficient amounts of different kinds of optional material can make the user navigate what seems to be a gigantic maze. No such problem occurs in the Web version.

Next year's solution: XML

Both print and Web versions of books have recently been produced from content-only documents, plus version-specific non-content documents, using the World Wide Web Consortium's "Extensible Markup Language," universally called XML.[3] However, very few of XML's eventual capabilities have been used because parts of the XML specification suite are still under development by working groups of the World Wide Web Consortium (hereafter "the W3C"). The basic specification for XML was "recommended" by the W3C in February and full approval is expected in the fall. The math markup language is in the "recommended" stage and may also be approved this fall by the members of the W3C. The specification for the XML formatting ("style") language, XSL, may emerge from the XSL working group this summer. As for XML browsers, Microsoft's *Internet Explorer* 4.0 already includes some XML tools and Netscape Navigator is scheduled for significant XML compliance in version 5.0. IBM has produced XML tools and Sun has put its extensive Solaris documentation into XML.

The power and relative simplicity of XML have led to its endorsement by IBM, Netscape, Microsoft, Sun, Adobe, and a host of other institutions and individuals prominent in the information industry. Developers are creating XML tools and XML workshops are being held around the country. It is expected that XML will be used instead of HTML for many Web pages and will be used for many printed

[2] See Michael Downes, *Breaking Equations*, *TUGboat* 18, 3, September 1997, pages 182-194. A new release of the software is expected in early August, 1998 (private communication from M. Downes). We hope that this work, so important for the Web, can eventually be made available for use in XML.

[3] A publishing house use of XML to produce both HTML and RTF versions, the former for a Web version and the latter for the commercial printed-book version, can be seen in some detail at `www.mcp.com/info/1-57521/1-57521-334-6`. That example is also interesting because it includes use of TEI, the Text Encoding Initiative, and because it treats XML as a special case of SGML, the Standard Generalized Markup Language.

publications. LaTeX may turn out to be an application of choice for printing XML documents, especially those involving math. The feeling of some XML working groups and developers seems to be that true XML Web browser and print applications, including math, formatting, linking, data, pointer, and document architecture, will gradually become usable starting next spring.

Math in XML

The XML math markup language, called MathML, has already been incorporated into several tools.[4] Although MathML makes sense in terms of the ambitious goals of the MathML working group, it is rather laborious to write and difficult to proof-read. In an example from IBM,[5] markup for the quadratic root formula, LaTeX takes one line while the Presentation form of MathML takes 35 lines:

```
LaTeX: $$x=-b\pm\sqrt(b^2-4ac)/(2a)$$
MathML:
       <mrow>
         <mi>x</mi>
         <mo>=</mo>
         <mfrac>
           <mrow>
             <mrow>
               <mo>-</mo>
               <mo>b</mo>
             </mrow>
             <mo>&PlusMinus;</mo>
             <msqrt>
         ...(22 lines)...
         </frac>
       </mrow>
```

As a result of this complexity, it has been proposed that LaTeX or another math markup language might be used in XML documents with "helper applications" converting it "on the fly" to MathML for processing by the user's XML browser. It is assumed that LaTeX is exactly equivalent to Presentation MathML.[6]

Meanwhile: LaTeX and techexplorer

While waiting for XML to become usable for documents that contain math, we are using LaTeX and IBM's **techexplorer**[7] to produce Web and print versions of a physics textbook. The LaTeX compiler combines its own style files and the one master file to produce the `.dvi` file for the printed version. **techexplorer** is a plug-in for current browsers that combines its own "macro" style file with the one master file, on the user's machine and in real time, to produce the on-screen version. The LaTeX compiler and the **techexplorer** interpreter can work from the same master file because **techexplorer** uses LaTeX's command structure and also because it recognizes many LaTeX commands. Thus many formatting macros in the LaTeX style file can be taken over directly to **techexplorer**'s macros file. **techexplorer** simply ignores the LaTeX commands in the master file that are not in its repertoire. In addition to the many LaTeX commands that it understands, **techexplorer** has commands that are useful for Web browser display and which provide some of the capability expected in XML. While we are using **techexplorer** and LaTeX, we are also using a specific LaTeX markup scheme that is designed to capture the information needed for a future conversion to XML. It is fortunate that one of XML's strongest requirements is also a requirement of LaTeX; namely, that scopes be nested (which makes possible the description of elements as distinct objects).

techexplorer's new "user-embed" link

We make considerable use of **techexplorer**'s implementation of XML's new "user-embed" link.[8] The **techexplorer** command is "`\altLink`" and it allows us to specify two hot elements (elements that are visually identifiable as clickable links) which alternate as the user clicks on them. For example, the default hot element can be the word "help" and the alternate element can be a long sequence of help

[4] A list of tools incorporating MathML is available at `http://www.w3.org/Math/`.

[5] Download **techexplorer** from `www.software.ibm.com/techexplorer` Install it as a plug-in to Netscape Navigator, then display, in the Navigator: `Netscape/Communicator/Program/Plugins/techexplorer/Examples/MathML/mml002.html`.

[6] Another set of MathML markup, without any formatting, is called Content MathML. In contrast to Presentation MathML, Content MathML is strictly generic (formatless) markup. To see an example, display the file `mml002.html` referenced in Footnote 5.

[7] See the **techexplorer** reference in Footnote 5.

[8] XML specifies a suite of six pre-defined links and allows for custom-designed links. The built-in types are the six combinations produced by combining the "show" attributes "auto" and "user" with the "actuate" attributes "replace," "new," and "embed." Here "auto" and "user" indicate who controls activation of a link, while "replace," "new," and "embed" indicate the action to be taken when a link is activated. Whereas "replace" and "new" switch to a different flow of information, one in the current browser window and the other in a new window, "embed" causes the link-targeted object to be seamlessly incorporated into the current flow of information at some designated spot just as though the targeted element had always been there. Another part of the XML specification says that the element to be embedded need only be an identifiable element, not a complete file.

that includes text, graphics, and interactive computer programs. When the user clicks on the hot word "help," that word is instantly replaced by the actual help sequence which is sometimes quite long. The insertion is downward from the point of the default element, with the elements above the point of insertion remaining fixed in position on the screen. The actual help sequence, no matter how long, is also visually identified as a hot element so the user can click on it and cause it to disappear and be replaced by the first alternative, the single hot word "help."

Our use of user-embed links

We use **techexplorer**'s version of the user-embed link to let the user bring in objects that in print would only be referred to, not displayed, after their first occurrence. For example, the first time Figure 6 is referred to in a print version, it is displayed. Thereafter, however, the print version will merely show the words "...Figure 6 ..." and it is up to the reader to turn back and find the appropriate page to see the figure. Using the user-embed link, however, the screen version has all references beyond the first as hot elements that can bring in the actual figure, complete with caption, and then take it out again, all at the user's discretion. Similarly, the displayed figure can be clicked on to be exchanged with an enlarged version for detailed examination. References to previously-encountered equations and definitions are also shown as user-embed links that will alternate the reference to the object, usually hot words, with the actual object. Finally, we use user-embed links for objects that are not displayed at all unless or until the user wants to see them: answers to problems, helpful hints at specific points in the discussion or in homework problems, additional problems to practice specific skills, tutorials that provide additional instruction for students that need it, answers to problems in the tutorials, and items in the chapter summaries.

Separating form from content

The feedback we have received over the years from students and instructors, along with insights from research, have led to a never-ending stream of alterations of the contents of the book we have been converting for print and Web. These continuous alterations have led us to the removal of all formatting commands from the content files and the placing of them in a separate style file, a procedure long advocated by experts and which is advocated by virtually all XML developers. One justification for this separation becomes evident when even a small revision upsets the formatting for the entire remainder of an unseparated document. It is best to save time and frustration by letting the the LaTeX compiler handle the reformatting using a style file. To make the decisions involved, the LaTeX compiler must be informed of the type of each element in the content file. This is accomplished by making each element be the argument of a LaTeX command whose name labels the type of the element. Thus, for example, the title of a book could be the argument of a "`\BookTitle`" command in the content file and this might be converted to a "`\textit`" command in the style file. In general, the style file should give LaTeX all the information it needs to make an appropriate formatting decision for each type of element that occurs in the book and for each type of formatting situation in which LaTeX might have to format that type of element. The complete separation of the content from the format instructions has the added benefit of enforcing 100% conformity with the publisher's and author's desired format for each of the various types of elements in the book. This enables the user to immediately and reliably recognize the intent of an element just from its appearance. It also allows the author or publisher to easily change the format of all members of a particular class of element.

Problems in separating the content

It is well known that one cannot completely separate content from style within the confines of the current LaTeX compiler used for print versions of books, but our experience is that such separation can easily be made complete for the screen version within the confines of the current **techexplorer**. The reason for this difference is mainly that the screen version has no page ends (**techexplorer** ignores page-end commands) whereas a number of page-end formatting "tweaks" must be put into the content file for the print version. Even experts have this problem. In *The LaTeX Companion*, Goosens, Mittlebach, and Samarin remark that they inserted 237 commands in the book's content files to over-rule formatting decisions that were made by the LaTeX compiler as it followed the instructions the authors had placed in the book's style file.[9] We hope that the table of tweaks shown in that book can sometime be used by a LaTeX expert to give us some commands which will cover the situations the authors (and we) have encountered.

[9] See *The LaTeX Companion*, M. Goosens, F. Mittlebach, and A. Samarin, Addison-Wesley, Reading, MA, 1994, second page after page 528.

Moving the markup toward XML

Eventually we will be able to encode our content files in XML to produce both the print and screen versions, and we are moving toward that capability by capturing some of the necessary information in our master content documents. To move our files in that direction while retaining our LaTeX and **techexplorer** capabilities, we followed these procedures: (1) We removed *all* formatting instructions from the content (".tex") files. (2) We made each content element's type identifier be a "backslash" command with braces around its argument. Here are some examples using names that seemed reasonable to us:

$...$ ⇒ \m{...}
%... ⇒ \rem{...}
each paragraph ⇒ \p{...}.

(3) We put, near the head of the style file, each content type identifier in a single line with either a simple format definition or the name of a more complex formatting macro (the third case below):

```
\newcommand{\m}[1]{$#1$}
\newcommand{\BookTitle}[1]{\textit{#1}}
\newcommand{\Def}[2]{\DefF{#1}{#2}}.
```

(4) We put, near the head of the content file, definitions of elements that may be used more than once such as figures, definitions, and equations. Here is an example of a definition which appears in a box that is labeled "C-1" in the right margin of both the print and screen versions:

```
\newcommand{\DefWrdC1}{mass}
\newcommand{\DefDefC1}{Mass is...};
```

Each figure contains a graphic and a caption. The graphic part is an `eps` file for LaTeX and a `gif` file for **techexplorer**. These graphics files are called "external entities" in XML and they require special markup in both LaTeX and **techexplorer**.

Markup of figures, without \ifthenelse

At the present time, **techexplorer** does not have the `\ifthenelse` and `\equal` commands that come with the LaTeX `IfThen` package. This forces us to write out figure references in messy detail.

Here is a fragment of the list of figure captions and figure graphics files that we put at the head of the content document (with `\nc` indicating `\newcommand`):

```
...
\nc{\figEbGrap...
\nc{\figEcCapt}{Fig. E-3. This fig...}
\nc{\figEcGrap}{m407gr19}
\nc{\figEdCapt...
...
```

This shows data for parts or all of figures 2, 3, and 4 in the document's Section E. Numbers are not allowed in LaTeX command names so lower case letters have been used instead: `b` in place of `2`, etc.

Here is an XML equivalence for the figure graphics command:

```
<!ENTITY figEcGrap SYSTEM "m407gr19.gif"
                                NDATA GIF>
```

where the first pair is the object data (type and name), the second pair is entity-retrieval data (attribute and value), and the third pair is application data (type and application). Here "NDATA" indicates "notation data."

Here is the markup at the place the figure is first mentioned in the document, the place where the figure will naturally appear:

```
\Fg{\figEcCapt}{\figEcGrap}
```

Next we have the markup to be placed at succeeding references to the figure. During LaTeX processing for print, the third argument, the reference to the figure, will simply be printed. During **techexplorer** processing for the Web, reference to the figure will be a hot word whose selection will cause its replacement by the actual figure as a hot object (click on the figure and it instantly goes back to being the third-argument hot word):

```
\FgRef{\figEcCapt}{\figEcGrap}{Fig. E-3b.}
```

To finish the markup, here are the LaTeX style file definitions for the print version of the document, with `\nc` again indicating `\newcommand` and with a period on each side of the figure caption indicating code that is unrelated to the issues being discussed:

```
\nc{\Fg}[2]{.#1.\epsfig{file=#2.eps}}
\nc{\FgRef}[3]{#3}
```

Finally, here are the **techexplorer** style file definitions for the Web/CD version of the document, with more code being shown because it may be less familiar:

```
\nc{\Fg}[2]{
  \fcolorbox{black}{green}{
    \begin{tabular}{l p{0cm}}
    \fbox{\includegraphics{#2.gif}} & #1\\
    \end{tabular}
  }
}
\nc{\FgRef}[3]{\altLink{\Fg{#1}{#2}}{#3}}
```

Note the tabular attribute `p{0cm}` which tells **techexplorer** to format the figure caption using all of the remaining horizontal space in the browser window at the moment. Also note the `\altLink` command that displays the third `\FgRef` argument, the figure reference, as a hot word. Its selection by the user

will cause the reference to be replaced with the first and second `\FgRef` arguments, the actual figure, as a hot object. Subsequent selection of the figure will cause it to change back to being just the reference.

Markup of figures, with `\ifthenelse`

If and when `\ifthenelse` and `\equal` are implemented in **techexplorer**, the figure references can be made simpler in two ways: (1) we can use the usual LaTeX simulation of associative arrays to identify a figure by a simple ID; and (2) we can write the first- and consecutive-figure references as one command, branching inside the associated macro on whether the hot-word argument is empty or not. Here is a fragment of the set of figure data at the head of the document, simulating an associative array:

```
\nc{\fig}[2]{
  ...
  \ifthenelse...{E2}...
  \ifthenelse
    {\equal{#1}{E3}}
    {\Fg{Fig. E-3...}{m407gr19}{#2}}{}
  \ifthenelse...{E4}...
  ...
}
```

Here is the first text reference to the figure, where the empty second argument indicates that the figure is to appear here and there is to be no user choice:

```
\fig{E3}{}
```

Finally, here is the subsequent reference which will appear to the user as the hot word contained in the second argument and whose activation by the user will instantly replace the hot word with the actual figure as a hot object (click on the hot figure and it instantly goes back to being the second-argument hot word):

```
\figRef{E3}{Fig. E-3b.}
```

Dealing with our upgrade-process errors

During the rather lengthy upgrading toward XML, our LaTeXfiles were also undergoing continual content revision and had to be continuously available for the usual LaTeX printing. We found this to be workable providing: (1) we first made any markup change to one element and then checked that the change had occurred properly before applying it to all occurrences of the same type of element; (2) after each markup change to all elements of the same type, we checked the changes in somewhat random places through visual checking of appropriate `.dvi` files; (3) we kept a log of the markup changes made each day, recording them in a lab notebook; and (4) we had our office server make backup copies of all files in the middle of each night. The main use of the "markup changes log" was in handling cases where the markup changes we made were irreversible and turned out to be erroneous. When that happened, and it did happen, we could bring back the previous day's backup files and then repeat the good changes noted in the log (we saved the code used for each change). However, we did learn the hard way to check that the correct backup tape was in the DAT drive before we went home each night.

The software we used

For search and examination through the file system we used the programmer's editor called TextPad,[10] and for making changes to all items having a common pattern of characters we used Perl.[11] Our Perl script used macros that find elements delineated by braces that may themselves contain arbitrary numbers and levels of nested elements. We intend to use Perl to convert from LaTeX braces to XML angle brackets when the proper time arrives.

[10] See `www.textpad.com`.
[11] See `www.ActiveState.com`.

TEX to HTML Translation via Tagged DVI Files *

Michael D. Sofka
Computing Information Services
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
sofkam@rpi.edu
http://www.rpi.edu/~sofkam/

Abstract

This paper describes `dvihtml`, a program under development for translating a tagged DVI file into HTML. A common problem when translating TEX into another format is handling unexpected macros. Fortunately, TEX's macro language is flexible enough to pass markup information to the DVI file in the form of `\special`'s, fonts and small horizontal or vertical movements. Translating the resulting DVI file thus allows TEX itself to serve as the macro parser for translation. This technique can be extended for writing smarter DVI viewing programs, including viewers that can perform common layout editing.

A common typesetting request is the ability to place copies of books and articles on the Web in HTML, or to provide files in SGML or common word processor format. To aid in this task, many translators have been written that read TEX or LATEX files and write the appropriate output. Translators that read TEX files directly, however have the common limitation of not understanding TEX's macro language, or even being fooled by macros that simply redefine a common command already known to the translator. Add to this the inconsistency with which some authors (and typographers under the pressure of a deadline) code TEX files, and a uniform and universal translator seems a hopeless task.

TEX authors commonly write new macros that generate content or important layout not understood by the translator. In order to handle arbitrary macros the translator must be updated, or new translation tables supplied. Even then, a macro writer could fool the best translators by redefining the input syntax to better suit idiosyncratic work habits. For this reason most TEX translators have targeted specific input languages, usually LATEX. This is the method used by LATEX2html and Scientific Word, which are both discussed elsewhere in these proceedings (Deland, 1998, Moore, 1998). It is also the method used by IBM Techexplorer, which understands LATEX and a wide range of TEX's math primitives and plain TEX macros (Sutor and Dooley, 1998).

Alternatively, one could write a translator that understood TEX's primitives and macro language. This, however, is a daunting task given the many special cases embodied in TEX's expansion rules. Fortunately, a readily available TEX translator exists which is guaranteed to understand and correctly interpret any TEX file. The program is, of course, `initex`, the TEX executable itself. The only problem is that the output of TEX is a low-level DVI file in which most of the high-level document structure is lost.

Using the `\special` command and some other macro tricks, however, TEX can translate a document into a "tagged" DVI file. A tagged DVI file is a DVI file which encodes information about the higher-order coding which produced the lower-level DVI output. This tagging, along with the hierarchical structure of the DVI file, can be used to create HTML or other output according the user specifications. Depending on the specific restrictions required by the target language, the tagging need not even be complete. For example, HTML encodes headers as:

```
<H1>LaTeX and Postmodern Typesetting:
Hermeneutics and the Tyranny
of Documentclass Structure.</H1>
```

with no regard to specific font, size or line breaks. Indeed, this information should be left to the display program when standard HTML is the desired outcome. The only information required in the DVI

* I would like to thank Sebastian Rahtz and Eitan Gurari for their helpful comments on an early draft of this paper. I would also like to thank my managers at RPI, Gary Schwartz and Katherine Bursese, for the quiet time at work to finish this article, and for allowing me attend the Northeast TUG Conference.

file is a "tag" identifying which characters are in the header.[2]

Such is the flexibility of TEX's macro language, that the original author coding may not need to be modified. A LATEX package file, for example, could redefine common commands to produce tags. The same package could further redefine primitives and definition commands so that all new macros will either be tagged, or will at the least not interfere with the translation process. Problem commands which do not generate content important for HTML display (such as running heads, page breaks, etc) can be disabled or tagged and ignored during translation.

The DVI File

While most TEX users are aware that the output of TEX is something called a "DVI" file, fewer have ever had the opportunity to study this file in detail. Indeed, this task is difficult since the file is binary and displays poorly in most editors. I suspect this is one reason various flavors of TEX input files have been the source language of choice for translation (the other being the lack of high-level information within the DVI file).

DVI files, however, are really simple. As described in Knuth (1986b) they consist of a series of 1-byte commands and parameters which compactly describe how characters and rules should be placed on a page. There are 250 DVI commands in all, but most are for setting characters and changing fonts. In addition, many commands come in four flavors depending on if the parameter is 1, 2, 3 or 4-bytes long. Full details on the DVI file format, along with sample code for reading DVI files, can be found in `dvitype.web` (Stanford University, 1995).

Depending on how you group the commands there are about 11 categories of DVIoperation codes (or op-codes, as they are called). The entire set of op-codes is shown in Table 1.

There are a few items to note from Table 1. First, fully 136 of 250 DVI op-codes are used to print characters, and another 68 are used to select a font. Likewise, there are 14 horizontal and 14 vertical movement commands. Font definitions, which provide a mapping between an external font name and a DVI file font number, take another 4 bytes. This profligate consumption of op-codes for setting characters is done for efficiency. The letter 'G' in Computer Modern, for example, can be typeset with the DVI op-codes

[2] There are additional issues such as handling simple math in a header, and finding correct word boundaries. These are addressed below.

Category	*op-codes*
Print Character	*set_char0*...*set_char127*, *set1*, *set2*, *set3*, *set4*, *put1*, *put2*, *put3*, *put4*
Select Font	*fnt_num0*...*fnt_num63*, *fnt1*, *fnt2*, *fnt3*, *fnt4*
Define Font	*fnt_def1*...*fnt_def4*
Print rule	*set_rule*, *put_rule*
Horizontal Movement	*right1*...*right4*, *w0*, *w1*...*w4*, *x0*, *x1*...*x4*
Vertical Movement	*down1*...*down4*, *y0*, *y1*...*y4*, *z0*, *z1*...*z4*
Header	*pre*, *post*, *post-post*
Page	*bop*, *eop*
Stack	*push*, *pop*
Special	*xxx1*, *xxx4*
Undefined/nop	*nop*, *250–255*

Table 1: DVI op-codes by category. Note that 136 commands are used to print characters, another 68 for fonts and 28 for moving within the DVI file.

fnt_num0 set_char71

which is only two bytes in the file. The word "Gentle" can be typeset using a total of 7 bytes, plus three bytes for a *right2* command (one for the command, and two for the parameter) which kerns between 'n' and 't'.

Second, there are a number of commands of the form *op*⟨n⟩ where ⟨*n*⟩ is the value 1, 2, 3 or 4. For example, *right1*, or *fnt2*. These are variations of a single op-codes which take a 1, 2, 3 or 4 byte parameter. TEX tends towards using the more efficient op-code to represent a value.

Third, the movement parameters *w1–4*, *x1–4*, *y1–4* and *z1–4* are register commands. They move the given distance and set the value of the corresponding *w*, *x*, *y* or *z* register. These register values can then be recalled using the one byte *w0*, *z0*, *y0* or *z0* commands. TEX tends to use the horizontal registers for word spaces and kerns, and the vertical registers for movement between lines and paragraphs.

Fourth, the *push* and *pop* commands store and retrieve the current values of the *w*, *x*, *y* or *z* registers and the current horizontal and vertical position on the DVI page. TEX uses these to slightly optimize parameter setting. More important for translating tagged DVI files, TEX outputs *push*/*pop* pairs which correspond to boxes in the original TEX file. This

correspondence is not 100%. Particularly, TEX optimizes the output of lines from paragraphs so that most boxes are removed from common baselines. But in math-mode and tables most of the boxes remain.

Finally, the *xxx1* and *xxx4* are how TEX outputs `\special`'s to the DVI file. The literal (macro expanded) text of the `\special` is placed in the file. The single parameter of the *xxx* command is the length of the special. It is entirely left to the DVI translator program to interpret what a `\special` means, and the macro writer to be sure the contents of a `\special` are correct, and correctly located within the DVI file.

Tagging a DVI File

How can information in the DVI file be used to recover high-level coding? The trick is to use TEX's superlative macro language to send markup information, embedded in the DVI file, to the translator. The markup information can be indicated in at least three ways: distance, fonts and `\special`'s. Further, much of the marking can be accomplished by redefining existing TEX macros and primitives, reducing intervention into the authors coding.

Tagging using distance. One source of tagging information in a DVI file is the size of horizontal and vertical movements. TEX use the *w* and *x* registers for movement between words, but the amount of a move will vary from line to line. Likewise, movement between lines and paragraphs is accomplished with the *y* and *z* register commands. A typical DVI sequence (simplified) representing two lines in a paragraph is:

push
    right3⟨*n1*⟩*fnt_num0*
    set_char71 set_char101 set_char110 right2⟨*n2*⟩
    set_char116 set_char108 set_char101 w3⟨*n3*⟩
    set_char114 set_char101 set_char97 set_char100
    set_char101 set_char114 set_char115 w0
...
pop
y3⟨*m*⟩
push
    set_charn...
pop

That is, each line is nested in a *push*/*pop* pair. Within this pair the *w* register is used for interword spacing, while a *right* or the *x* register is used for kerns. Each line is separated by a *y* register command. In addition, paragraphs are usually separated by a *z* register command if `\parskip` is nonzero.

The problem is that while words are *typically* separated by *w* register commands, not all *w* commands are the result of word spaces. When generating the DVI file, TEX will optimize horizontal and vertical movements within boxes by using the *w*, *x*, *y* and *z* registers. A kern might be a *right*, or it could be a *x* command if a kern of the same amount appears later in the same line (a frequent occurrence). The details of this optimization are in Knuth, 1986b.

Fortunately, TEX's macro language can help us out. Consider the following TEX code.

```
\spaceskip=1sp
\xspaceskip=1sp
\hsize=\maxdimen

\baselineskip=1sp
\lineskip=0pt
\lineskiplimit=-16383pt

\parskip=0pt
```

The first two lines set the value of word spaces to one scaled point (sp). A scaled point is 1/65536th of a point, and is the smallest unit that TEX can move. Under normal circumstances there are no distances of 1 sp in a DVI file. Typical distances actually found are measured in at least $^1/_{10}$ of a point units.

The third line sets the width of a paragraph to the value of `\maxdimen`, which is 16383.9999 pt or about 18.9 ft—longer than a typical paragraph. The combined effect is to turn off line breaks making each paragraph a single line, and move exactly one scaled point between each word.

The next three lines adjusts TEX's vertical list building so that one scaled point is placed between each line (each paragraph) of text. This is accomplished by first setting `\baselineskip` to 1 sp then turning off other interline glue by forcing TEX to never use `\lineskip` glue.

Finally, `\parskip` is set to 0 pt so that no additional glue is added between paragraphs. The same overall effect could be accomplished by setting:

```
\cs{baselineskip=0pt}
\cs{parskip=1sp}
```

The sum effect is that one can be reasonably sure that all 1 sp horizontal movement in the DVI file represent word spaces, and all 1 sp vertical movement represents paragraphs. All other movement can be ignored, unless it to is being used for tagging.[3]

[3] Variations on the above allow for normal hyphenation and justification, but mark lines and paragraphs with one and two scaled point vertical movements. Recovering exact

A potential problem remains in that a later macro might be expected to set the `\baselineskip`, `\parskip` or other values, or even restore `\hsize` something under 8 inches. Fortunately this can be prevented with the following commands.

```
\newskip\junkskip
\let\spaceskip=\junkskip
\let\xspaceskip=\junkskip
\let\baselineskip=\junkskip
\let\lineskip=\junkskip
\let\parskip=\junkskip

\newdimen\junkdim
\let\lineskiplimit=\junkdim
\let\hsize=\junkdim
```

To be thorough we should also disable vertical and horizontal movement commands such as `\vskip` and `\hskip`. Care must be taken, however, to ensure the semantics of such commands otherwise remains the same.

Tagging using fonts. A second method of sending tagging information to the DVI file is by fonts. There are two ways a font can be used to indicate output format: name and size. For example, in a particular document the font Palatino at 16 point might only be used in one-heads. This is a clear indication that during translation all 16 point Palatino and intervening rules should be set within `<H1>`/`<H2>`.

What if the design includes a three head in 10 point Optima, but 10 point Optima is also used for figure captions. How can the two be distinguished based only on fonts? One method would be to increase the font size by one scaled point. The difference between Palatino at 655360 sp and Palatino at 655361 sp is will have no discernable affect on appearance, but they will be two different fonts in the DVI file.

There are two drawbacks to using fonts to tag markup information. First, it uses more fonts. TEX has a limit of 256 fonts per DVI file, so any method that makes extensive use of fonts will need to carefully select which fonts are actually loaded and used. Second, each font can only carry one tag. Setting, for example, `\it\bf` will result in only the boldfaced font being used.

Tagging using specials. Nearly any tagging information can be included in a DVI file by using TEX's `\special` command. The `\special` command causes TEX to out insert the literal, macro expanded argument, into the DVI file as an *xxx1* or *xxx4* command, depending on the length of the string. This is among the more heavily used and abused features of TEX since specials are used for all rotation, color, figure inclusion and PostScript commands.

The major disadvantage of specials is that they require DVI interpreters which understand the specific specials used—interpretation of specials is outside the purview of TEX. As a result, there appeared a number of drivers which understood only specific sets of specials. Some of these drivers were commercial or were used internally by typography companies, and made use of `\special`'s which were not in general use. Others were freely available, but as a result lagged behind in the special sets accepted.

In 1997 Tom Rokicki (Rokicki, 1994) proposed a set of specials to be supported by his `dvips` program. This was recomended with modification by the TUG Technical Working Group on DVI Driver Implementation and Standardization Issues(Rokicki, 1995). While the proposed standard has inherent flexibility, it cannot be used for all `\special` needs. Specifically, it doesn't cover markup tagging, and its stack scheme doesn't allow for DVI file re-writing (as described below). It does, however, propose a standard method of writing non-standard macros which will be followed in `dvihtml`. See Sofka (1995) for more details of the standardization process.

Delimited tags. In principle, markup via the `\special` primitive is easy. To mark a section, for example, would require:[4]

```
\catcode`\@=11

\let\t@gsection=\section
\def\section#1{%
   \special{::tag begin(section)}%
   \t@gsection{#1}%
   \special{::tag end(section)}}

\catcode`\@=12
```

assuming the macro `\section` had previously been defined.

The `\let` primitive is used to preserve the true definition of `\section`. The new definition is same as the old, except `\special` places tags around it.

The `::` identifies the special as being experimental according to the draft standard. The type of special is a "tag", which means it is providing high-level information for an interpreter. The `begin` and `end` indicate that the high-level element is delimited by two specials. `section` is the name of the tag.

word boundaries would be more difficult, but the resulting paragraphs would be legible and formatted by TEX.

[4] My examples are in plain TEX to keep them simple. The same can be done in LATEX by suitably redefining basic generator macros such as `\@startsection`, `\new@command`, etc.

Block scoped tags. Not all tag-able elements can be delineated using begin and end markers. Sometimes the the range of an element is implicit in the coding, but not explicitly marked. For example, when processing:

```
$$ABCE\over DEFG$$
```

"ABCD" is in the numerator, while "DEFG" is in the denominator. It would be awkward to require plain TEX users type

```
$$\special{::tag begin(numerator)}
     ABCE
  \special{::tag end(denominator)}
\over
  \special{::tag begin(denominator)}
     DEFG
  \special{::tag end(denominator)}$$
```

when inputing math—even if suitable shorthand tagging macros were defined. However, a tag can be inserted into the scope of the numerator and denominator by redefining the `\over` primitive as:

```
\def\tag#1{\special{::tag block(#1)}}

\catcode`\@=11
\let\t@gover=\over
\def\over{\tag{num}\t@gover\tag{den}}
\catcode`\@=12

$${ABCE \over EFGH}$$
```

This is output in the DVI file roughly as:

push
    set_char65...
    xxx1⟨*16*⟩::tag block(num)
pop
right4⟨*n*⟩
down3⟨*m*⟩
putrule⟨*a*⟩⟨*b*⟩
down3⟨*m*⟩
push
    xxx1⟨*16*⟩::tag block(den)
    set_char69...
pop

Note that the contents of the numerator and denominator are each contained within a *push*/*pop* pair. The `block` type of `::tag` affects the entire block within which it is contained.[5]

Nested tags. The `::tag` specials can be nested. For example, a tag for italic text (assuming this were not indicated using a font) might be nested within the tag for a section. There is an ambiguity, however, when a delimited tag and a block tag interact. How, for example, should the following be interpreted?

push
    xxx1⟨*17*⟩::tag begin(list)
    set_char71 set_char101...
    xxx1⟨*18*⟩::tag block(quote)
    set_char108 set_char111...
    xxx1⟨*15*⟩::tag end(list)
pop

Is the quote contained within the list, or the list within the quote? When the order of application matters, the resulting output will be different for each interpretation.

By default this will be resolved by assuming that block tags are delimited by begin/end tags, as well as *push*/*pop* pairs. That is, internally, `dvihtml` or other `::tag` aware translator should convert the above into:

push
    xxx1⟨*17*⟩::tag begin(list)
    xxx1⟨*18*⟩::tag begin(quote)
    set_char71 set_char101...
    set_char108 set_char111...
    xxx1⟨*16*⟩::tag end(quote)
    xxx1⟨*15*⟩::tag end(list)
pop

Why does the `end()` tag specify the element being ended? Wouldn't a simple `end` with no argument be enough to end the current tag? Unfortunately no. The problem is TEX's asynchronous output routine. This means that in the middle of a paragraph of quoted material you may suddenly find yourself in the middle of page layout. The result is the following sequence in the DVI file:

push
    xxx1⟨*18*⟩::tag begin(quote)
    set_char$n_1 \ldots n_x$
pop
xxx1⟨*15*⟩::tag end(page)
pop
eop
bop⟨$c_0, \ldots, c_9, p$⟩
right3⟨*4736286*⟩
push
xxx1⟨*17*⟩::tag begin(page)
push
    set_char$n_{x+1} \ldots n_z$
    xxx1⟨*16*⟩::tag end(quote)
pop

[5] There is an annoying rule between the two tags in this example. If rules are being translated, this one can be removed by redefining `\over` using the `\atop`. If the rules used in `\over` need special treatment they can be set with a 1 sp width using `\above`.

If the output routine were also tagging elements (e.g., top of columns, crop-marks, running head, and so on), they would all appear between and interlaced with the `quote`. Explicite `end` statements with matching parameters helps the above be rewritten as:

push
    xxx1⟨*18*⟩::tag begin(quote)
    set_char$_1 \ldots n_x$
    xxx1⟨*16*⟩::tag end(quote)
pop
xxx1⟨*15*⟩::tag end(page)
pop
eop
bop$\langle c_0, \ldots, c_9, p\rangle$
right3⟨*4736286*⟩
push
xxx1⟨*17*⟩::tag begin(page)
push
    xxx1⟨*18*⟩::tag begin(quote)
    set_char$_{x+1} \ldots n_z$
    xxx1⟨*16*⟩::tag end(quote)
pop

The problem is knowing exactly where in the DVI file to insert matching `begin` and `end` tags. There are at least three ways to resolve this. The first is the method shown above, which uses explicit tags in the output routine to delimit pages and columns. All that is necessary for correct rewrite is inserting
textend tags at the same nesting level as the matching `begin`, but before the end of page is marked. Likewise for `begin` tags at the top of the page. The assumption is that all `begin`/`end` pairs should perfectly nest in the rewritten DVI file.

A second method of resolving this problem, applicable only to a translator, is to redefine macros so that page breaks do not occur at inopportune times. For example, setting spacing and paragraph parameters as given above guarantees that page breaks will not occur in the middle of a paragraph. By further defining `\output` to be simply `\` other interrupted tags can be reconstructed. Alternatively, the techniques discussed in Appendix D of Knuth (1986a) can be used to signal the output routine about bad break points.

Finally, it is possible to reconstruct the original nesting of the `begin`/`end` pair by merging all intervening *push*/*pop* pairs nested at the same level as the interrupted tags. This method works, however, only if it is assumed that *push*/*pop* pairs and `begin`/`end` perfectly nest—a condition that requires careful macro writing since TEX has no way of enforcing the rule.

All three of the methods are used in `dvihtml`. Macro and simplification will be used when possible, tag nesting will be encouraged and nesting rewrites will be used whenever it can simplify the coding. The goal is a minimal re-write of author macros, so the translator must make use of all the information available in the DVI file.

Overriding scope. There are times when it may be necessary to override the default scope of a `::tag` special (for example, if a `block` tag should be moved outside of a delimited tag. This can be done using the `scope()` option, which takes a single parameter indicating what the scope for the current tag should be. There are special cases for `global` scope and `page` scope, to affect the entire DVI file or the page on which the tag appears. `stack` specifies the current *push*/*pop* pair. Otherwise, the parameter should be label of a delimited tag which encloses the new tag at any level.

What about alignments? The alignments commands used by TEX present a mixed bag of difficulties. Redefining & and `\cr` to provide block-level tagging is trivial, but this breaks the `\halign` alignment template. While scanning the alignment template TEX is expecting category 4 characters to indicate tabs, and a real `\cr` (or `\endline`, which is defined in `virtex`) to end the template. So, while pre-defined math alignments such as `\eqalign` can be handled via:

```
\def\tag#1{\special{::tag block #1}}

\catcode`\&=\active
{\catcode`|=4\gdef&{\tag{AMP}|}}

\catcode`\==\active
\def={\tag{EQ}\char`\=}

\def\cr{\tag{CR}\endline}

$$\eqalign{A&=B\cr
           B&=D}$$
```

This same code breaks any future `\halign` attempts. Tagging alignment entries requires something slightly more convoluted. An example of how to do this is in figure 1, which redefines `\halign` so that & is a tab character while the template is being scanned, but is an active character while the body of the alignment is being read. The active character inserts tag specials.

Dvihtml and Tagged DVI Files

An outline of the proposed tagging `\specials` is in figure 2.

```
\catcode`\@=11
\def\tag#1{\special{::tag block #1}}

\def\makebraceother{\catcode`\{=12 }
\def\makebracenormal{\catcode`\{=1 }

\def\maketabactive{\catcode`\&=\active}
\def\maketabtab{\catcode`\&=4 }
{\maketabactive \catcode`|=4\gdef&{\tag{lamp}|\tag{ramp}}}

\let\t@ghalign=\halign

% Remove the { from \halign
{\makebraceother \catcode`[=1 \catcode`]=2
   \gdef\@halign{[\makebracenormal\@@halign]}

% Collect alignment template and call halign primitive
\def\@@halign#1\cr{\t@ghalign\bgroup#1\cr\global\maketabactive}

% set catcodes and start halign
\def\halign{\makebraceother\maketabtab\@halign}
\catcode`\@=12
```

Figure 1: Redefining `\halign` so that & is category code 4 (tab) while the alignment template is being read, but active characters while the body of the `\halign` is read. The above introduces a potential problem in that & remains active between `\halign`'s. This is okay for most macros built using `\halign` because the alignment template was read when the macro was defined. This macro also breaks plain TEX's tabbing macros.

The `dvihtml` translator understands these specials, and uses them to re-write the DVI file so that hierarchical information is preserved, and tagging applied to the appropriate elements. It will optionally write out a new DVI file, or a translated tagged output file (HTML by default). Translation is guided by a configuration file specifying conversions for horizontal and vertical movements, fonts and `::tag` specials. By default, tags labels will be converted verbatim so that in the absence of additional information the ASCII output file will have intelligible markup. A sample `dvihtml` configuration file is in figure 3.

In the case of LATEX files, a package can be written which redefines the standard commands to produce tagged output. Plain TEX is, of course, trickier since there is no way of knowing in advance what an author will call a macro. Adding a couple `\special` calls, however, is relatively easy and by default the translation will pick up changes in font size, paragraphs, simple math, etc, without needing to know the individual macros which produced the DVI file.

Smart DVI Viewers

The approach of translating a tagged DVI file was been used in at least two private translators (Rahtz, 1995, Sofka, 1993). It is also the approach used by TEX4ht (Gurari, 1997b, Gurari, 1997a), which is used to author hypertext documents. The method is robust, and it is hoped that a pseudo-standard set of tagging `\special`'s will encourage macro writers to voluntarily pre-tag their code.

Once a DVI file is tagged, however, a number of additional translation possibilities arise. For example, complex page layout is notoriously difficult using TEX. Usually, by the time a book is printed, the source code is filled with hard-coded page-breaks, `\vskip`'s to balance columns, and so on. For some designs, all glue stretch is removed to prevent TEX from "fixing" layout attempts. This is tedium at it's worst.

On the other hand, the actual task—moving a block of text a couple points up or down, or cutting and pasting a figure—are trivial in WYSIWYG environment. The typographer knows exactly what he or she wants to do, the difficulty is conveying that information to TeX. What if the DVI viewer knew how to edit TEX files? What if there were a way to go from the image on the screen to the source file that generated the image?

::tag := ⟨*tag*⟩ [scope(⟨*scope*⟩)]
| line: ⟨*file:lineno*⟩
⟨*tag*⟩ := begin(⟨*label*⟩) ⟨*op-codes*⟩ end(⟨*label*⟩)
| block(⟨*label*⟩)
⟨*label*⟩ := [_,a-z,A-Z,0-9] | ⟨*quoted-string*⟩
⟨*scope*⟩ := global | page | stack | ⟨*label*⟩
⟨*quoted-string*⟩ := "⟨*printable ASCII*⟩"
⟨*op-codes*⟩ := ⟨*any DVI op-codes*⟩

Figure 2: Specials recognized by `dvihtml`for tagging a document.

```
# Translate fonts to bold, italic, etc.
font cmit10:  scope(<I>, </I>);
font cmb10:   scope(<B>, </B>);

font cmr17 at 28pt:
   insert(header,
          scope(<TITLE>, </TITLE>)),
   scope(<H1>, </H2>);
...

hdimen 1sp:   translate(" ");
vdimen 1sp:   translate(<P>);
...

tag  section:      scope(<H1>, </H1>);
tag  subsection:   scope(<H1>, </H1>);

tag  enumerate:    begin(<OL>);
tag  enumerate:    end(</OL>);

tag  list_item:   translate(<LI>);
...
```

Figure 3: Sample `dvihtml` configuration file. The elements are choose to display the range of translation possibilities.

```
\nopagenumbers
\def\sb#1{\special{before #1}}
\def\sa#1{\special{after #1}}

\gdef\numberlines{\special{line: \jobname:\number\inputlineno}%
        \immediate\write-1{line: \jobname:\number\inputlineno} }

{\catcode`\^^M=\active%
  \gdef\startnumbering{\catcode`\^^M\active \let^^M=\numberlines}%
  \global\let^^M=\numberlines} % this is in case ^^M appears in a \write

\startnumbering

Misc paragraph: This is a normal line ending, while
this line ends with the macro \TeX
and this one ends with a  hyphenated-
word broken across lines.  This last line%
ends with a \%.
```

Figure 4: Macro to number input lines in the DVI file. Note that this macro modifies TEX's end-of-line semantics slightly.

This style of editing has been dubbed "two-view" by Kenneth Brooks (Brooks, 1988). In a two-view editor both the source language and the WYSIWYG image can be modified with changes being reflected in both views. This approach is used in Lilac (Brooks, 1991), which uses a non-TEX boxes-n-glue language to typeseting (short) documents. Brooks' choice of language was to avoid TEX global scoping, lack of key-words, and modifiable syntax. Contrast Lilac with Blue-Sky's Lightning TEXtures(Hampson and Smith, 1992), which repeatedly reads the entire TEX file from the beginning while the user types. Inbetween these two extremes, Chen, Harrison, and Minakata (1988) and Harrison (1989) have discussed some of the problems associated with incremental formatting in the VorTEX project.

A tagged DVI file offers another intermediate approach. Tags can be inserted into the DVI file to aid two-view editing. For an extreme example, consider the macro in figure 4, which inserts a `\special` into the DVI file at the end of each **input** line. A two-view editor could count input lines to find the TEX code that produced the DVI output. An example of the viability of this approach can be seen in Asher (1992), who used specials to mark pagination points within a DVI file, and the *push/pop* structure of the DVI file to find good breakpoints within paragraphs. The resulting file was processed, paged and printed automatically.

The problem of efficiently parsing TEX's input, however, will require the cooperation of macro writers and users. It would be nice, for example, if in LATEX3 all the relevant state information could be inferred by the environment nesting, and commands which altered expansion or redefined control-sequences were unavailable to the user. This would greatly reduce the amount of processing required by a two-view TEX editor. The goal for `dvihtml`, beyond document conversion, is to serve as a testbed for using tagged DVI files in smarter, if not true two-view, TEX editing systems.

References

Asher, Graham. "Inside Type & Set". *TUGboat* **13**(1), 13–22, 1992.

Brooks, Kenneth P. *A Two-View Document Editor with User-Definable Document Structure*. Ph.D. dissertation, Stanford University, 1988.

Brooks, Kenneth P. "A Two-View Document Editor". *Computer* **24**(6), 7–19, 1991.

Chen, Pehong, M. A. Harrison, and I. Minakata. "Incremental Document Formatting". In *Proceedings of the ACM Conference on Document Processing*, page 93–100. ACM, NY, 1988.

Deland, Donald. "WYSIWYG LATEX" 1998. workshop presented at TEXNorthEast conference, March 1998.

Gurari, Eitan M. "A Demonstration of TeX4ht". 1997a. URL: `http://www.cis.ohio-state.edu/~gurari/tug97/tug97-h.html`.

Gurari, Eitan M. "TeX4ht: TeX and LaTeX for Hypertext". 1997b. URL: `http://www.cis.ohio-state.edu/~gurari/TeX4ht/mn.html`.

Hampson, Steve and B. Smith. "A High Performance TEX for the Motorola 68000 Processor Family". *TUGboat* **13**(3), 269–271, 1992.

Harrison, Michael A. "News from the VorTEX Project". *TUGboat* **10**(1), 11–15, 1989.

Knuth, Donald E. *The TEX Book*. Addison-Wesley, Reading, MA, 1986a. TEX version 3.0, 1994, 14th printing.

Knuth, Donald E. *TEX: The Program*. Addison-Wesley, Reading, MA, 1986b. Reprinted with corrections May, 1988.

Moore, Ross. "Making Web Sites using LATEX2HTML" 1998. Workshop presented at TEXNorthEast conference, March 1998.

Rahtz, Sebastian. "Another Look at LATEX to SGML Conversion". *TUGboat* **16**(3), 315–324, 1995.

Rokicki, Tomas G. "Driver Support for Color in TEX: Proposal and Implementation". *TUGboat* **15**(3), 205–212, 1994.

Rokicki, Tomas G. "A Proposed Standard for Specials". *TUGboat* **16**(5), 395–401, 1995.

Sofka, Michael D. "`dvitag` Users Guide". 1993. Internal document, Publication Services, Inc.

Sofka, Michael D. "DVI Driver Implementation and Standardization Issues.". 1995. URL: `http://www.rpi.edu/~sofkam/dvi.html`.

Stanford University. *The DVIype processor*. Stanford University, 1995.

Sutor, Robert S. and S. S. Dooley. "TEX and LATEX on the Web Via IBM Techexplorer". *TUGboat* **19**(2), 157–161, 1998.

Abstracts

Les Cahiers GUTenberg
Contents of Double Issue 28–29
Proceedings of the tenth European TEX conference

Numéros 28–29 — mars 1998

MICHEL GOOSSENS, Éditorial : dix ans de collaboration [Editorial: Ten years of collaboration]; pp. vi–vii

These proceedings contain most of the presentations made at the EuroTEX '98 Conference, which took place from March 29th to April 1st in Saint Malo (France) in the framework of the "Second Week on Electronic Publishing and Typography" (*WEPT'98*).

EuroTEX'98 was the tenth in a series of European conferences dedicated to the latest developments around TEX. ... It is noteworthy that several of [the other European TEX] organizations, just like GUTenberg, also celebrate their tenth anniversary in 1998.

... I would like to stress how the enthusiasm of the participants at the Conference has transformed EuroTEX '98 into a real TEX fiesta, proving once more that the Lion and Friends are well-prepared and ready to enter the next millenium with confidence and limitless energy!

[Excerpts from the English editorial]

BRUNO BACHIMONT and JEAN CHARLET, PolyTEX : un environnement pour l'édition structurée de polycopiés électroniques multisupports [PolyTEX: an environment for structured editing of multi-purpose electronic documents]; pp. 1–16

PolyTEX is a prototype editorial working environment to facilitate production of materials *from a single source* for multimedia: specifically, course notes, Web pages, and transparencies for distribution via electronic means (computer screens) or hardcopy. Initially for the Mac and UNIX platforms, it uses programs currently available for free or at low cost. The article presents the project from initial course design (the conceptual stage) to final implementation (the teaching and materials distribution).

A. BERDNIKOV, O. LAPKO, M. KOLODIN, A. JANISHEVSKY and A. BURYKIN, [The encoding paradigm in LATEX 2_ε and the projected X2 encoding for Cyrillic texts]; pp. 17–31

This paper describes the X2 encoding which is designed to support Cyrillic writing systems for the multilanguage mode of LATEX 2_ε. The restrictions of the LATEX 2_ε kernel, the specific features of Cyrillic writing systems and the basic principles used to create X2 are considered. This projected X2 encoding supports all the Cyrillic writing systems known to us, although the majority of the accented letters need to be constructed from pieces. The general scheme of the X2 encodingh was approved at CyrTUG-97 (the annual conference of Russian-speaking TEX users) and its final form was agreed on the `cyrtex-t2` mailing list.

[authors' abstract]

A. BERDNIKOV, O. LAPKO, M. KOLODIN, A. JANISHEVSKY and A. BURYKIN, [Alphabets necessary for various Cyrillic writing systems (Towards X2 and T2 encodings)]; pp. 32–43

Characters, accents, modifiers, punctuation and stress symbols, etc., necessary to support modern Cyrillic texts are considered. The list of glyphs that we present supports all [Cyrillic] writing systems we know of. The paper also describes the peculiarities of several writing systems which are essential for TEX.

[authors' abstract]

A. BERDNIKOV and O.A. GRINEVA, Some problems with accents in TEX: Letters with multiple accents and accents varying for uppercase/lowercase letters ; pp. 44–55

The problems of using the internal command `\accent` as a tool for support of some Cyrillic writing systems is investigated. It is shown that the internal features of `\accent` prevent construction of some Cyrillic letters which require several accents simultaneously. A special macro which emulates the work of `\accent` by some other commands is suggested.

The accents for I/i and J/j, which are different for uppercase and lowercase letters, are also considered. *If-then-else* structures by use of which correct accents can be placed, depending on the letter case, are proposed. A similar technique can be used for case change in the Cyrillic "capital form" ligatures ЈЬ and НЬ.

[authors' abstract]

MARCIA J. BOSSY, WWW-TED : thesaurus évolutif et dynamique pour bases de liens HTML

[WWW-TED: dynamic thesaurus for database management of HTML links]; pp. 56–71

We consider the need for a database management tool in Web-based scientific research. We then propose an approach using WWW-TED, a dynamic thesaurus for use with medium-sized (300 to 3,000 links) HTML pages. The audience for such a tool includes researchers and research groups which require precise management of their database collections.

[from author's résumé and introduction]

Šarūnas Burdulis and Vytas Statulevičius, [Real-life application of TEX and Adobe Acrobat for electronic publishing: A handbook for algebra and a journal archive]; pp. 72–81

A classical way of using TEX in printed typesetting was enhanced for use of the same TEX source to publish electronically. A handbook of algebra and a 4-year journal archive (280 articles) were electronically published using the same TEX source files to produce both the PDF in a form for reading on-screen and a version for printing a hard copy. A package written in plain TEX provided the mark-up of the logical structure, cross-references, bibliographical references, author names, keywords and symbols. The hypertext contents, index pages and a complete navigation system are also made in PDF and were pre-programmed at the TEX level. Being completely a PDF product the same publications are thus usable on any computer system for which a PDF viewer exists.

[from authors' abstract]

Janka Chlebíková, [The Euromath system — The structured editor for mathematicians]; pp. 82–93

The Euromath system is the result of a project funded through the SCIENCE programme of the European Commission and administered through the European Mathematical Trust. Its aim is to create a homogeneous computer working environment for mathematicians, based on a uniform data model, and to stimulate interchange among them based on modern information technolotg.

The core of the system is a powerful SGML structured editor, Grif, combining the advantages of a WYSIWYG approach and structured editing. SGML is rapidly becoming the standard for publishing and for full-text databases. The Euromath system is at the forefront in exploiting the benefits of SGML for scientific documentation and also the typesetting qualities of the TEX system.

[from author's abstract]

Matthias Clasen and Ulrik Vieth, [Towards a new math font encoding for (LA)TEX]; pp. 94–121

This paper presents a snapshot of ongoing work towards a prototype implementation of new 8-bit math font encodings for (LA)TEX, based on the 'Aston' proposal, presented at the TUG '93 conference. The design goals and technical considerations that have led to the present font table layouts are summarized and the contents and organization of the individual encodings are presented in detail. Finally, some alternative approaches and some remaining open problems are discussed.

[authors' abstract]

Thomas Esser, [The teTEX system: Concepts of installation, configuration and maintenance]; pp. 122–130

teTEX is a complete TEX distribution for UNIX platforms that claims to be easy to install, to configure, to maintain and to use. This article describes the underlying basic concepts and design decisions that have been used to achieve this goal.

[author's abstract]

Jean-Daniel Fekete, Expérience de codage de document à intérêt graphique à l'aide de TEI [Encoding a graphics document using TEI]; pp. 131–142

While encoding text documents is now well in hand, documents with graphics still pose several problems. In this article, we describe the use of SGML in combination with the TEI DTD, to encode the encyclopedia, *La chose imprimée*. ... Normally, SGML documents are processed by DSSSL, which does not, however, currently have any mechanisms for documents with graphics components. We therefore used PERL to devise the necessary translation programs.

[from author's résumé]

Bernard Gaulle, Comment peut-on personnaliser l'extension `french` de LATEX ? [How to customize the `french` package for LATEX]; pp. 143–157

The `french` package for LATEX presents users with a large number of basic options which they can customise to suit their exact requirements. This customisation can be performed at various points in the document, and can be temporary or permanent. Some parameters affect the macro-typography of the document (such as page layout), whilst others are relevant to the micro-typography (such as spacing around punctuation). Possible actions are, for example, to add new functionality, to mix styles and even to define new languages or dialects.

This article describes the various ways of customising the `french` package, either for personal use or as part of a workgroup.

[author's abstract]

Denis Girou and Sebastian Rahtz, [Verbatim revisited — the '`fancyvrb`' package]; pp. 158–179

This talk introduces Timothy van Zandt's `fancyvrb` LaTeX package, which provides very sophisticated facilities for reading and writing verbatim TeX code. Users can perform common tasks like changing font family and size, numbering lines, framing code examples, colouring text and conditionally processing text. The main part of this paper is a set of tutorial examples of how to create customized verbatim environments, and it concludes with a description of how `fancyvrb` was used in the typesetting of the *LaTeX Graphics Companion*.

[authors' abstract]

Michel Goossens, XML et le futur du Web [XML and the future of the Web]; p. 180

Late in 1996, the W3C and several major software vendors decided to define a markup language specifically optimized for the Web: XML (eXtensible Markup Language) was born. It is a simple dialect of SGML, which does not use most of SGML's seldom used and complex functions, and does away with most limitations of HTML. After an introduction to the XML standard, we briefly describe XLL (eXtensible Linking Language) for hyperlinks and XSL (eXtensible Style Language) for style sheets. We also discuss some of the many applications based on XML.

[author's abstract]

[The author then notes that the complete text of the article will appear in an upcoming thematic issue of the *Cahiers GUTenberg*, to be devoted to XML.]

Michel Goossens and Jean-Yves Le Meur, Afficher les documents scientifiques sur le Web [Posting scientific documents to the Web]; pp. 181–196

Every day CERN handles a large number of research documents, mostly marked up in LaTeX and coming from many Internet servers. Our aim is to make them easily locatable on the Web with the help of the CERN Library's *Preprint Catalogue* in several formats (PostScript, PDF, GIF). We review the conversion procedures and give some details on some massive production trial runs to directly generate HTML from the TeX sources. We conclude with a discussion of recent developments in the framework of the XML (and MML) efforts which should ease the support of mathematics formulae in Web browsers.

[author's abstract]

Hàn Thế Thành, The pdfTeX Program ; pp. 197–210

pdfTeX is an extension to TeX which allows the user to generate either `DVI` or PDF as the primary output format The current feature set of pdfTeX is discussed, and further extensions which are currently under consideration for adoption are reviewed.

[author's abstract]

Hirotsugu Kakugawa, [VFlib — A general font library that supports multiple font formats]; pp. 211–222

VFlib is a font library written in C which provides several functions for obtaining bitmaps of characters (i.e. a rasterizer). VFlib hides the font format of font files and provides a unified API for all supported font formats. Thus, programmers of application software need not worry about font file formats. Instead, any software using VFlib can support various font file formats immediately. In addition to this, when a new font format is supported by VFlib, application software need not be modified to use such new fonts.

VFlib has been developed not only for Latin fonts but also Asian scripts such as Chinese, Japanese, and Korean. Since it is designed as a general font module, it can be used in DVI drivers for TeX and LaTeX. In this paper we explain the API of VFlib, a font database file called vflibcap, and the internal structure of VFlib.

[author's abstract]

Roger Kehr, [xǐndy — A flexible indexing system]; pp. 223–230

Whilst MakeIndex is an index processor which is suitable for the production of indexes in conjunction with many text formatters, its support for non-English langauges is weak and a new version called International MakeIndex was presented for processing international documents. The improvements concentrated on the internationalization of the sorting process for keywords in an index. Though it substantially improves the possibility of sorting new languages, there are still weaknesses in the processing model largely inherited from MakeIndex. Through the experience gained from the International MakeIndex project we have implemented a new index processor xǐndy that (a) improves the sorting of index entries at a finer granularity than International MakeIndex, (b) offers new mechanisms for processing structured location references besides

page numbers and roman numerals, and (c) allows for complex mark-up schemes.

[author's abstract]

Sergey Lesenko, [DVIPDF and Embedded PDF]; pp. 231–241

We explain how the current version of the DVIPDF program manages to integrate external multipage PDF files into its own PDF output.

[author's abstract]

Marie-Louise Munier and Ahmed Mahboub, Expérience de TEX (LATEX) dans la chaîne éditoriale [TEX (LATEX) experiences in the editorial process]; pp. 242–251

Our aim is not to address current topics in typography or the quality of electronic documents, but to describe our experience with LATEX and other public domain software in a publishing house. Following a brief historical overview of our experience with LATEX, the electronic submission of manuscripts, instructions for authors, stylesheets, LATEX 2_ε and $\mathcal{AMS}$-LATEX assets will be addressed. The last part of this report will be devoted to the EDP Sciences Web server.

[author's abstract]

Christophe Pythoud, *Français–GUTenberg* : un nouveau dictionnaire français pour ISPELL [*French–GUTenberg*: A new French dictionary for ISPELL]; pp. 252–275

This paper presents choices made in elaborating a new French dictionary for the ISPELL spell checker. How to augment the dictionary is also explained. The *ad hoc* tools to do this are demonstrated.

[author's abstract]

Petr Sojka, [An experience from a digitization project]; pp. 276–282

An experience from the process of adding logical markup to visually tagged scanned data is presented. The method of gradual markup enhancement is shown. Methods of navigation in a large hypertext document based on typesetting from logical markup are suggested — physical, logical and semantic user views. Their application on a 28,000-page project to create an electronic encyclopædia is described and problems faced when using Adobe's Acrobat technology for publishing are discusssed.

[author's abstract]

Richard Southall, [Prototyping telephone-directory pages with TEX]; pp. 283–294

The development of a prototype formatter for telephone-directory pages, written in TEX and using fonts made with Metafont, is described. The formatter was used to decide the detailed typography of directory entries. Issues connected with the markup language used in the directory data files are discussed.

[author's abstract]

Robert S. Sutor and Angel L. Díaz, [IBM techexplorer: Scientific publishing for the Internet]; pp. 295–308

The IBM techexplorer Hypermedia Browser is an application for the interactive publication of scientific and technical documents. The original project started as an experiment at IBM Research to see how a from-scratch implementation of a subset of TEX, LATEX, and $\mathcal{AMS}$-LATEX could be extended to support interactive viewing of documents for a computer algebra system. This interactivity is accomplished via support for hypertext, multimedia, user-defined pop-up windows and menus, and a modular architecture that allows connections with other applications and Java applets. The primary version of techexporer operates as a Netscape Navigator plug-in and is available for several platforms, including Windows 95 and NT, IBM AIX, and Sun Solaris. In addition to being able to display full documents using the supported TEX subset, techexplorer is being extended to support the new "Mathematical Markup Language" from the HTML Math Working Group of the World Wide Web Consortium. In this paper, we provide an overview of techexplorer and detail how it can be used to deliver mathematical articles, book, and course materials via the World Wide Web. We also discuss our intended use of the OpenMath standard to allow documents to contain reusable semantically attributed math objects.

[authors' abstract]

[Compiled by Christina Thiele]

Articles from *Cahiers* issues can be found in PostScript format at the following site:
`http://www.univ-rennes1.fr/pub/GUTenberg/publicationsPS`

Calendar

1998

Aug 17 – 20 **TUG'98** — The 19th annual meeting of the TEX Users Group, Torun, Poland: "Integrating TEX with the surrounding world". For information see the call for papers, *TUGboat* **18**(4), p.314, or visit `http://www.tug.org/tug-98/`.

Sep 4 First meeting of l'Association AsTEX, CNRS, Orleans, France. For information, contact Michel Lavaud (`Michel.Lavaud@univ-orleans.fr`) or retrieve the informational files posted at `ftp://ftp.univ-orleans.fr/pub/tex/PC/AsTeX/Readme/`.

Sep 21 UK TUG annual general meeting, University Centre, Cambridge: "TEX in its diversity". For information, visit `http://www.tex.ac.uk/UKTUG/`.

Oct 1 – 2 DANTE, 19th meeting, Katholische Universität Eichstätt, Germany. For information, contact `dante98@ku-eichstaett.de`.

Oct 7 – 12 Frankfurt Book Fair, Frankfurt, Germany. For information, contact `press@book-fair.com` or visit `http://www.frankfurt-book-fair.com/`.

Oct 25 NTG, Graphics and TEX course, Utrecht, Netherlands. For information, visit `http://www.ntg.nl/bijeenkomsten.html`.

Oct 30 – Nov 1 TypeCon '98, Society of Typographic Aficionados, Westborough, Massachusetts. Principal speaker: Matthew Carter. For information, contact Bob Colby (`sota@tjup.truman.edu`) or visit `http://tjup.truman.edu/sota`.

1999

Feb ?? DANTE'99, 20th meeting, "10 years of DANTE e.V.", Ruprecht-Karls-Universität Heidelberg, Germany.

Aug 8 – 13 SIGGRAPH, Los Angeles, California. For information, visit `http://www.siggraph.org/s99/`.

Aug 15 – 20 **TUG'99** — The 20th annual meeting of the TEX Users Group, Vancouver, Canada. Information will be posted to `http://www.tug.org/tug99/` as plans develop.

For additional information on TUG-sponsored events listed above, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: `tug@tug.org`). For events sponsored by other organizations, please use the contact address provided.

Status as of 30 June 1998

Late-Breaking News

Production Notes

Mimi Burbank

Well, I first must apologize for the missing page numbers in my last set of notes. Trying to standardize usage for production of files in both LaTeX and TeX is sometimes distracting, and I miss little things. I've been told that I'm suffering from "Halfzheimer's" (not full-blown Alzheimer's) so I'm going to use this as my excuse. For this issue, we had files written on multiple platforms, one of which (see Girou's article on page 101) could only be run on one computer because of 8-bit characters. They simply were *not* interpreted correctly on any of the other machines here at SCRI.

Shipping mixup. Some of our members received the wrong CDs with the last issue of *TUGboat*: some members received *no* CDs, and some members received duplicates of one CD. This set of circumstances occurred at the printer, who had received shipments from The Netherlands, Germany and England over a period of time, and somehow the mistake was made during the insertion of the CDs into the issue. We were told that, after 200 issues were stuffed, they noticed the error and tried to correct it, but evidently missed some.

Check your *TUGboat* and CD combinations against the following list and contact the TUG office (`office@tug.org`) in case of discrepancies:

TUGboat 19, no. 1	4AllTeX (2 CDs)
	TeX Live 3 (1 CD)
TUGboat 19, no. 2	CTAN (3 CDs)

Fonts, fonts and more fonts! An EPS file written by dvips, and then re-included in a TeX file, seems to call on dvips to reload the fonts it already embedded. This means that a) the font is loaded twice, and b) it must reside on the production system. Why this happens is still under investigation. But the interim procedure was to convert the EPS to PDF using Acrobat Distiller (after setting the page size to be that of the BoundingBox using the fitps technique), then use the ExportPS plug-in for Acrobat Exchange to create a new, clean, EPS file (specifying 'Embed all fonts' in the plug-in dialogue). This was necessary to process the articles by Bouche (see page 121) and Hoenig (see page 176).

Output The final camera copy was prepared at SCRI on the following UNIX platforms: IBM rs6000s running AIX v4.1.4.0, and v4.2 using the *TeX Live* setup (Version 3), which is based on the *Web2c* TeX implementation version 7.2 by Karl Berry and Olaf Weber. PostScript output, using outline fonts, was produced using Radical Eye Software's dvips(k) 5.78, using the `dvips -Pem` option, and printed on an HP LaserJet 4000 TN printer at 1200dpi.

Coming In Future Issues The next issue of *TUGboat* will be the TUG'98 Proceedings issue. For more on the topical information, please visit the TUG'98 Programme web site: `http://www.gust.org.pl/TUG98/progr.html`. For the December issue, we have a very nice article by Claudio Beccari on new Greek fonts and the `greek` option of the babel package. As well, we still hope to provide the listing of acronyms, promised in a previous issue, and an article entitled "METATeX" by Ramón Casares on METAFONT graphics in TeX.

Visit *TUGboat*'s Web pages *TUGboat* is represented on the TUG Web pages at the following locations; we invite you to visit our site:

```
http://www.tug.org/TUGboat/tugboat.html
http://www.tug.org/TUGboat/announce.html
http://www.tug.org/TUGboat/Contents.html
http://www.tug.org/TUGboat/errata.html
```

The "Contents by year" pages, beginning with 1995, have articles linked to the entries. Information regarding delays and shipment dates is posted on the "Announcements" page; this is the most current source of information for those of you wondering if your issue of *TUGboat* has been mailed yet. In the coming months, many more of our articles will become available in PDF format. As time permits (or volunteers help us), we would like to make older articles available as well.

If you would like to volunteer to help us with this project, send email to `TUGboat@tug.org`. Also, if you have not been contacted by us regarding permission to post your articles to the Web, send us email and let us know if we may post *your* article.

◇ Mimi Burbank
SCRI, Florida State University,
Tallahassee, FL 32306–4130
`mimi@scri.fsu.edu`

TUG'99

Vancouver, British Columbia

August 15 – 19, 1999

The TEX Users Group is proud to announce the **twentieth** annual meeting will be held at the University of British Columbia, Vancouver, British Columbia, August 15 – 19, 1999.

The theme for the meeting has not been decided. There will be a contest held for the best theme and the winner will be announced at TUG '98, Toruń, Poland.

The Program Committee is interested in focusing on "state-of-the-art" TEX/LATEX, providing practical information on using macro packages, installing and using existing software tools, announcing new macro packages, new software tools or new approaches using TEX/LATEX. We are committed to making this conference one in which each presentation or workshop adds value for the publishing professional — author, publisher, consultant, and developer. To better serve the TUG community, the Program Committee would like to provide parallel sessions. For example, a paper may be presented about "LATEX 2_ε: Improving Table/Figure Macros", which would go into technical detail about these macros. At the same time a workshop could be provided with step-by-step instructions for placing your tables and figures in the best location.

We encourage everyone to consider attending and presenting, especially publishers, commercial vendors of TEX, and consultants. We intend to provide a time for each to discuss or display their services and/or products.

We plan to provide full courses the week before the conference. Topics, dates, instructors to be announced later.

Deadlines

October 17, 1998:	(maximum 1 page each; 12pt fonts) Submit abstracts for paper presentations Submit workshop description, objectives, and prerequisites
December 18, 1998:	Notification of acceptance
March 12, 1999:	Preliminary papers due
July 16, 1999:	Preprint deadline
August 15 – 19, 1999:	TUG'99 Meeting

Please send all information regarding paper/workshop submissions to:
`tug99-pc@zebra.us.udel.edu`.

Institutional Members

Academic Press,
San Diego, *CA*

American Mathematical Society,
Providence, *Rhode Island*

CERN, *Geneva*, *Switzerland*

College of William & Mary,
Department of Computer Science,
Williamsburg, *Virginia*

CSTUG, *Praha*, *Czech Republic*

Elsevier Science Publishers B.V.,
Amsterdam, *The Netherlands*

Florida State University,
Supercomputer Computations
Research, *Tallahassee*, *Florida*

Hong Kong University of
Science and Technology,
Department of Computer Science,
Hong Kong, *China*

Institute for Advanced Study,
Princeton, *New Jersey*

Institute for Defense Analyses,
Center for Communications
Research, *Princeton*, *New Jersey*

Iowa State University,
Computation Center,
Ames, *Iowa*

Kluwer Academic Publishers,
The Netherlands

Los Alamos National Laboratory,
University of California,
Los Alamos, *New Mexico*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Masaryk University,
Faculty of Informatics,
Brno, *Czechoslovakia*

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, *Michigan*

New York University,
Academic Computing Facility,
New York, *New York*

Princeton University,
Department of Mathematics,
Princeton, *New Jersey*

Space Telescope Science Institute,
Baltimore, *Maryland*

Springer-Verlag,
Heidelberg, *Germany*

Stanford University,
Computer Science Department,
Stanford, *California*

University of California, Irvine,
Information & Computer Science,
Irvine, *California*

University of Canterbury,
Computer Services Centre,
Christchurch, *New Zealand*

University College,
Computer Centre,
Cork, *Ireland*

University of Delaware,
Computing and Network Services,
Newark, *Delaware*

Universität Koblenz–Landau,
Fachbereich Informatik,
Koblenz, *Germany*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

University of Stockholm,
Department of Mathematics,
Stockholm, *Sweden*

University of Texas at Austin,
Austin, *Texas*

Uppsala University,
Computing Science Department,
Uppsala, *Sweden*

TEX Consulting & Production Services

Information about these services can be obtained from:

TEX Users Group
1466 NW Front Avenue, Suite 3141
Portland, OR 97209-2820, U.S.A.
Phone: +1 503 223-9994
Fax: +1 503 223-3960

North America

Loew, Elizabeth
President, TEXniques, Inc.,
362 Commonwealth Avenue, Suite 5E, Boston, MA 02115;
(617) 670-1916; FAX: (617) 670-1916
elizabeth@texniques.com

Long-term experience with major publisher in preparing camera-ready copy or electronic disk for printer. Complete book and journal production in the areas of mathematics, physics, engineering, and biology. Services include copyediting, layout, art sizing, preparation of electronic figures; we keyboard from raw manuscript or tweak TEX files.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Email: Ogawa@teleport.com

Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX 2_ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and ßC++. Database and corporate publishing. Extensive references.

Outside North America

DocuTEXing: TEX Typesetting Facility
43 Ibn Kotaiba Street
Nasr City, Cairo 11471, Egypt
+20 2 4034178
Email: main-office@DocuTeXing.com

DocuTEXing provides high-quality TEX and LATEX typesetting services to authors, editors, and publishers. Our services extend from simple typesetting and technical illustrations to full production of electronic journals. For more information, samples, and references please visit our web site: http://www.DocuTeXing.com or contact us by e-mail.

Announcements

Volunteers needed for LaTeX2rtf coordination and development

Wilfried Hennings

In our daily tasks, we have to handle LaTeX documents as well as, e.g., MS Word or WordPerfect documents — like it or not, it's a fact. And there is a permanent need to convert between LaTeX and one of the PC wordprocessors.

There are already some converters available (see my FAQ list at `http://www.kfa-juelich.de/isr/1/texconv.html`), however none of them satisfies all needs. One of these converters is LaTeX2rtf.

Ralf Schlatterbeck, the author of LaTeX2rtf, cannot maintain it any longer, because he is now working somewhere else. I cannot either, because I am not familiar with C and have not enough time to make myself acquainted with it and work on programming. Really, that's not my job; I am just a Word and LaTeX user who desperately needs good converters, so I am collecting information about them which one day resulted in the FAQ list mentioned above.

Some weeks ago, Georg Lehner mailed me some enhancements he added, and now we are in touch with Ralf for joining Ralf's latest enhancements and Georg's, to get a new development version. But also Georg can not do everything alone, so we are searching for volunteers to join us for programming and testing.

Following a posting of this request in `comp.text.tex` and `de.comp.text.tex`, I have already received some responses, so at the moment there is already some development work going on.

Georg is willing to coordinate the work for *some* time, but it seems that in the long run we will need another volunteer willing to coordinate further developments. This does not mean she/he has to do all the programming her/himself, but collecting, selecting, coordinating and encouraging developments done by other people.

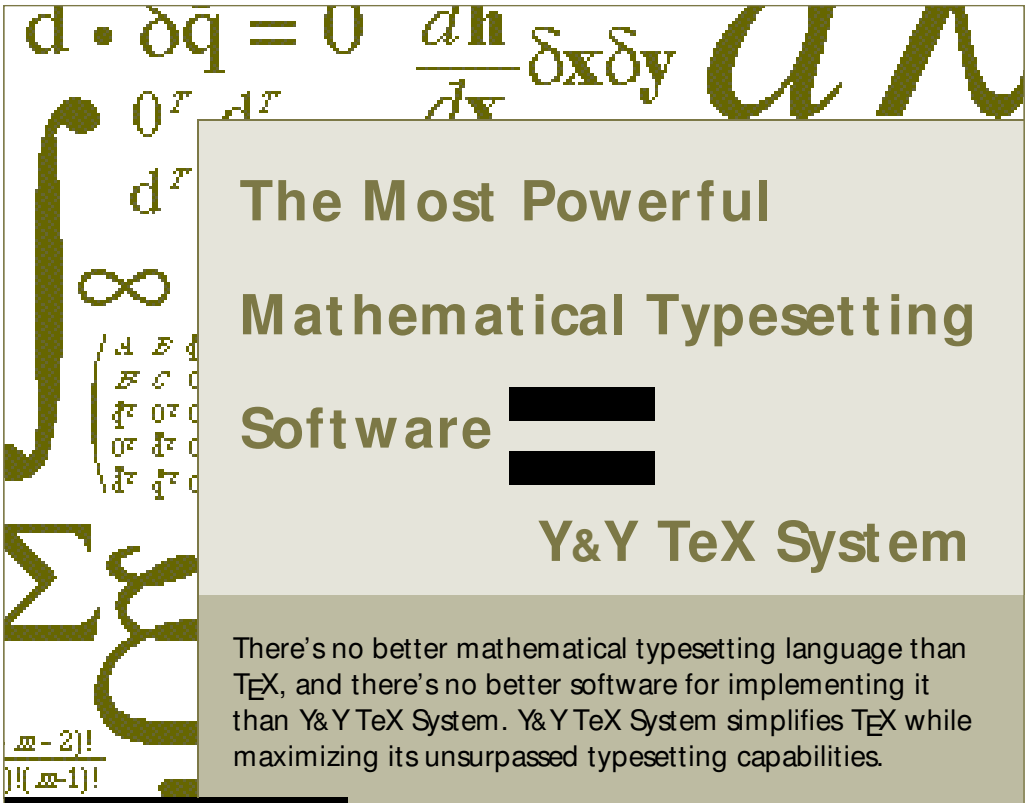
Maybe by the time this article is published we will have already found a coordinator, but in case we have not, anyone willing to do that please mail me at: `<W.Hennings@fz-juelich.de>`. Of course people with brillant ideas and the capability and time to implement them are always welcome.

Still on the to-do-list:

- support LaTeX 2_ε (the current version is based on LaTeX 2.09)

Waiting for volunteers...

◇ Wilfried Hennings
Förschungszentrum (Research Center)
Jülich GmbH, ISR D-52425
Jülich, Germany
`W.Hennings@fz-juelich.de`



The Most Powerful Mathematical Typesetting Software

Y&Y TeX System

There's no better mathematical typesetting language than TEX, and there's no better software for implementing it than Y&Y TeX System. Y&Y TeX System simplifies TEX while maximizing its unsurpassed typesetting capabilities.

Here's how:

Y&Y TeX System.

The Ultimate

Problem Solver.

- On-the-fly font re-encoding lets you specify unencoded characters otherwise inaccessible in Windows.
- Partial font downloading dramatically speeds up printing.
- Web publishing capabilities let you prepare documents in Acrobat PDF which appear on screen exactly as you designed them.
- Customizable TEX menu lets you link to an editor, spell-checker or any other DOS or Windows program.

But that's just part of the whole formula.

TEX is a trademark of the American Mathematical Society.

Y&Y Inc.
Concord, MA USA

For more information about Y&Y TeX System, check out our web site at http://www.YandY.com or e-mail sales-help@YandY.com

800-742-4059

http://www.YandY.com