TUGboat

Volume 31, Number 2 / 2010
TUG 2010 Conference Proceedings

TeX Users Group

TUGboat (ISSN 0896-3207) is published by the TeX Users Group.

Memberships and Subscriptions
2010 dues for individual members are as follows:
- Ordinary members: $95.
- Students/Seniors: $55.

The discounted rate of $55 is also available to citizens of countries with modest economies, as detailed on our web site.

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, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is $100 per year, including air mail delivery.

Institutional Membership
Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, see `http://tug.org/instmem.html` or contact the TUG office.

TeX is a trademark of the American Mathematical Society.

Addresses
TeX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone
+1 503 223-9994

Fax
+1 206 203-3960

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

Electronic Mail
(Internet)

General correspondence, membership, subscriptions:
`office@tug.org`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

Technical support for TeX users:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

Have a suggestion? Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: September 2010]

Printed in U.S.A.

2010 Conference Proceedings

TEX Users Group
Thirty-first Annual Meeting
Sir Francis Drake Hotel
San Francisco, California, USA
June 28–30, 2010

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON
PROCEEDINGS EDITOR KARL BERRY

VOLUME 31, NUMBER 2 • 2010
PORTLAND • OREGON • U.S.A.

TUG 2010

San Francisco ■ California, USA

June 28–June 30, 2010

Sponsors

TEX Users Group ■ DANTE e.V.

Addison-Wesley ■ CSLI ■ Green Lion Press ■ Principiae ■ River Valley Technologies
San Francisco State University ■ Stanford Computer Science Department
University Science Books, Inc. ■ von Hoerner & Sulger GmbH
with special assistance from individual contributors. *Thanks to all!*

Acknowledgments

Many thanks to all the speakers and teachers, without whom there would be no conference, and also special thanks to:

- Donald Knuth and all the panel participants, for making it such a special celebration.
- Kaveh Bazargan, for the A/V recordings and more.
- Duane Bibby, for the (as always) excellent and fun drawings.
- Sue DeMeritt and Cheryl Ponchin, for (once again) teaching the workshop.
- Dave Walden, for moderating the panel.
- the photographers: Alan Wetmore, Uwe Ziegenhagen, and Jennifer Claudio.
- the session chairs: Will Robertson, Klaus Höppner, Michael Doob, and Jim Hefferon.
- the Sir Francis Drake Hotel, for the facilities.

Bursary committee

Sam Rhoads, chair ■ Jana Chlebikova ■ Bogusław Jackowski ■ Alan Wetmore

Participants

William Adams, Mechanicsburg, PA
Leyla Akhmadeeva, Bashkir State Medical University
David Allen, University of Kentucky
Tim Arnold, SAS Institute
Pavneet Arora, Bolton, Canada
Sheldon Axler, San Francisco State University
Dave Bailey, MacKichan Software, Inc.
Kaveh Bazargan, River Valley Technologies
Nelson Beebe, University of Utah
Barbara Beeton, American Mathematical Society
Doris Behrendt, Erlangen, Germany
Karl Berry, TEX Users Group
Brian Blackmore, Seattle, WA
Mathieu Bourgeois, Université du Québec à Montréal
John Bowman, University of Alberta
Richard Bumby, Rutgers
Bill Cheswick, AT&T
Bart Childs, College Station, TX
Kaja Christiansen, Aarhus University
Dennis Claudio, Superior Court (Alameda County)
Jennifer Claudio, St. Lawrence Academy
Bruce Cohen, San Francisco, CA
Robert Cristel, Montréal, Canada
Jon Dechow, Westar Institute
Sue DeMeritt, Center for Communications Research, La Jolla, CA
Michael Doob, University of Manitoba
Jean-luc Doumont, Principiae, Belgium
Rebecca Elmo, Mathematical Association of America
Ronald Fehd, Centers for Disease Control
Thomas Feuerstack, FernUniversität in Hagen
Terri Fizer, Duke University Press
David Fuchs, Stanford TEX project
Walter Gander, ETH
Kevin Godby, Ames, IA
Steve Grathwohl, Duke University Press
Hans Hagen, Pragma ADE
Idris Hamid, Colorado State University
William Hammond, SUNY Albany
Illona Headrick, Texas State University-San Marcos
Jim Hefferon, Saint Michael's College
Hartmut Henkel, von Hoerner & Sulger GmbH
Stephen Hicks, Google Inc.
John Hobby, Stanford TEX project
Alan Hoenig, Huntington, NY
Joe Hogg, Los Angeles, CA
Morten Høgholm, LATEX3 Project and Technical University of Denmark
Klaus Höppner, DANTE e.V.
Ned Hummel, Indiana University-Purdue University Indianapolis
Mirko Janc, INFORMS
Oleg Katsitadze, Portland, OR
Adam Kelly, San Francisco, CA
Jonathan Kew, Mozilla Corp.
Timothy Kew, Cambridge University
Donald Knuth, Stanford TEX project
Dick Koch, University of Oregon

Igor Kozachenko, Mathematical Sciences Publishers
Johannes Küster, Typoma
Martha Kummerer, Notre Dame J. of Formal Logic
Robin Laakso, TEX Users Group
Raymond Lambert, Duke University Press
Richard Leigh, St Albans, UK
Frank Liang, Stanford TEX project
Manfred Lotz, DANTE e.V.
Filip Machi, Berkeley, CA
Barry MacKichan, MacKichan Software, Inc.
Wendy McKay, California Institute of Technology
Doug McKenna, Mathemaesthetics, Inc.
Lothar Meyer-Lerbs, Bremen, Germany
Frank Mittelbach, LATEX3 Project
Ross Moore, Macquarie University
Zolili Ndlela, California State Univ. Sacramento
Brian Papa, American Meteorological Society
Oren Patashnik, Stanford TEX project
Roy Pattishall, Duke University Press
Michael Plass, Stanford TEX project
Cheryl Ponchin, Center for Communications Research, Princeton, NJ
Roozbeh Pournader, San Jose, CA
Frank Quinn, eduTeX
Will Robertson, LATEX3 Project
Tom Rokicki, Stanford TEX project
Chris Rowley, LATEX3 Project
David Ruddy, Cornell University Library
Robert Rundell, Seattle, WA
Volker RW Schaa, DANTE e.V.
Martin Schröder, LuaTEX team
Herbert Schulz, Naperville, IL
Alex Scorpan, Mathematical Sciences Publishers
Heidi Sestrich, Carnegie-Mellon University
Chris Skeels, University of Melbourne
Paulo Ney de Souza, UC Berkeley
Tammy Stitz, University of Akron
Luis Trabb Pardo, Stanford TEX project
P. Alan Thiesen, Nevada City, CA
Howard Trickey, Stanford TEX project
Didier Verna, EPITA/LRDE
Boris Veytsman, George Mason University
Bruno Voisin, CNRS and University of Grenoble
Paul Vojta, UC Berkeley
Herbert Voß, DANTE e.V.
David Walden, E. Sandwich, MA
Joe Weening, Stanford TEX project
Alan Wetmore, US Army
Gio Wiederhold, Stanford University
Uwe Ziegenhagen, Cologne, Germany

LATEX workshop participants

Donna Aikins, Naval Postgraduate School
Ron Aikins, Naval Postgraduate School
Lori Boyters, Qualcomm Inc.
Bruce Cohen, San Francisco, CA
Sherry Davenport, Qualcomm Inc.
Illona Headrick, Texas State University-San Marcos
Susan Koskinen, UC Berkeley
Zolili Ndlela, California State Univ. Sacramento
Roy Pattishall, Duke University Press
Pamela Sexton, Qualcomm Inc.
Tammy Stitz, University of Akron
Mica Thomas, Alembic, Inc.
Sharon White, Qualcomm Inc.
Ronald Wickersham, Alembic, Inc.

TUG 2010 program

Monday June 28	8:00 am	*registration*	
	8:55 am	Karl Berry, TeX Users Group	*Welcome*
	9:00 am	Ross Moore, Macquarie University	*TeX+MathML for Tagged PDF, the next frontier in mathematical typesetting*
	9:35 am	Will Robertson, LaTeX3 Project	*Unicode mathematics in LaTeX: Advantages and challenges*
	10:10 am	Boris Veytsman, George Mason U.	*Are virtual fonts obsolete?*
	10:45 am	*break*	
	11:00 am	Steve Grathwohl, Duke U. Press & David Ruddy, Cornell U. Library	*Implementing MathJax in Project Euclid*
	11:35 am	Johannes Küster, Typoma	*Math never seen*
	12:50 pm	*lunch*	
	2:00 pm	Alan Hoenig, Huntington, NY	*TeX helps you learn Chinese character meanings!*
	2:35 pm	William Cheswick, AT&T	*Ebooks: New challenges for beautiful typesetting*
	3:10 pm	Hans Hagen, Pragma ADE	*Just in time: Things we can do only with LuaTeX*
	3:45 pm	*break*	
	4:00 pm	Hans Hagen	*Building paragraphs with the help of Lua*
	4:35 pm	Idris Hamid, Colorado State U.	*Oriental TeX: Culturally authentic typesetting of the Qur'an*
	5:10 pm	q&a	
Tuesday June 29	9:00 am	Michael Doob, U. of Manitoba	*A web-based TeX previewer: Ecstasy and agony*
	9:35 am	Jonathan Kew, Mozilla Corp.	*TeXworks for newcomers — and what's new for old hands*
	10:10 am	Kaveh Bazargan, River Valley Tech.	*Batch Commander: An interactive style writer for TeX*
	10:45 am	*break*	
	11:00 am	Boris Veytsman & Leyla Akhmadeeva, Bashkir State Medical University	*TeX in the GLAMP world: On-demand creation of documents online*
	11:35 am	Pavneet Arora, Bolton, Canada	*Using LaTeX to generate dynamic mathematics worksheets for the web*
	12:15 pm	Stephen Hicks, Google Inc.	*Improving margin paragraphs and float control*
	12:50 pm	*lunch*	
	2:00 pm	Herbert Voß, DANTE e.V.	*From PostScript to PDF*
	2:35 pm	Jim Hefferon, Saint Michael's College	*Characterizing CTAN packages*
	3:10 pm	Didier Verna, EPITA / LRDE	*Classes, styles, conflicts: The biological realm of LaTeX*
	3:45 pm	*break*	
	4:00 pm	Walter Gander, ETH	*Writing the first LaTeX book*
	4:30 pm	William Hammond, SUNY Albany	*LaTeX profiles as objects in the "category" of markup languages*
	4:50 pm	Chris Rowley, LaTeX3 Project	*A brief history of LaTeX — with a prediction*
	5:10 pm	q&a; TUG meeting	
Wednesday June 30	9:00 am	Uwe Ziegenhagen, Cologne, Germany	*Dynamic reporting with R/Sweave and LaTeX*
	9:35 am	John Bowman, U. of Alberta	*Interactive TeX-aware 3D vector graphics*
	10:10 am	Mathieu Bourgeois and Roger Villemaire, U. Québec à Montréal	*Introduction to drawing structured diagrams in SDDL*
	10:45 am	*break*	
	11:00 am	Jean-luc Doumont, Principiae	*Quantum space: Designing pages on grids*
	11:35 am	Robert Rundell, Seattle, WA	*Using the Knuth-Plass algorithm to help control widow and orphan lines*
	12:15 am	Bart Childs, College Station, TX	*Thirty years of literate programming and more?*
	12:50 pm	*lunch*	
	1:45 pm	*group photo*	
	2:00 pm	John Hobby, Stanford TeX Project	*Is boxes.mp the right way to draw diagrams?*
	2:35 pm	Hans Hagen and Taco Hoekwater	*How TeX and Meta finally got married*
	3:10 pm	Frank Mittelbach, LaTeX3 Project	*Exhuming coffins from the last century*
	3:45 pm	*break*	
	4:00 pm	Dave Walden, moderator	*panel:* Don Knuth & Stanford TeX Project members
	5:30 pm	Don Knuth, Stanford TeX Project	*An Earthshaking Announcement!*
	≈ 6:00 pm	*end*	
	7:30 pm	banquet	at *Le Colonial* (`lecolonialSF.com`)

TUG 2010 conference report

David Walden

The TUG 2010 annual conference was held Monday June 28 to Wednesday June 30 in San Francisco.[1] This annual conference celebrated the thirty-second anniversary of TeX. The conference venue was the Sir Francis Drake hotel,[2] a few steps from San Francisco's famous Union Square and with the Powell Street cable car available at the front door of the hotel.

Attendance and the opening reception

With Don Knuth and others of the Stanford group that helped Knuth develop TeX participating in the conference, attendance was high compared with other annual conferences in recent years. The conference's location in San Francisco likely also contributed to attendance; many conference participants brought along family members and many came before the conference or stayed beyond the conference to sightsee in the San Francisco area and other parts of northern California.

More than half of the attendees of the conference were present at the Sunday 5–7 pm reception, renewing friendships, meeting new members of the TeX community, comparing trips into San Francisco, and so forth. TUG president Karl Berry, executive director Robin Laakso, and Robin's daughter Sophia handled the registration table, handing out name tags and conference materials.

Three-day program

As they have at previous TUG annual conferences, Sue DeMeritt and Cheryl Ponchin led a one-day introductory/intermediate LaTeX workshop. This year the workshop was held in parallel with the first day of the regular conference program. From the reports I overheard, the workshop was well received.

The main conference program was chock-a-block with interesting presentations.[3] Once again Kaveh Bazargan of River Valley Technologies recorded all of the presentations, and the full set of videos will be posted on the River Valley TV website.[4]

Since all of the presentations will be on the River Valley with many being printed in this issue of *TUGboat*, I will not describe any of the individual presentations (while I had some favorites among the presentations that I would love to describe, other people undoubtedly had other favorites). Instead, I will try to describe the breadth of the presentations (Table 1). In some ways it seemed to me that there was a broader range of presentations this year compared with some others. The table shows my assessment of the areas covered by each presentation. There was a good bit of history, in keeping with the thirty-second anniversary theme of the conference. There was, to me, a surprising amount of philosophy. There were presentations by old timers and by young TeX developers and everyone in between. There were useful updates on widely used systems. Most interesting to me, beyond the impressive and useful characteristics of many of the systems and tools described, was the large numbers of instances where TeX was used in combination with other systems and tools. From the work described in the conference presentations, TeX certainly looks like it will be a viable and highly useful system for a lot of people for a long time.

The three days of the conference were divided into a morning session and afternoon session with a short mid-session break. Karl Berry coordinated the chairing of the morning and afternoon sessions with Michael Doob, Jim Hefferon, Klaus Höppner, Will Robertson, and me each taking care of all or part of a session. The TUG annual meeting was officially held at the end of the second day's sessions.

A group photo was taken after lunch of the third day, by Alan Wetmore and Uwe Ziegenhagen. Jennifer Claudio and Alan also took photographs throughout the conference. The group photo and others are included in this issue.

Throughout the conference various small meetings took place, such as a MacTeX meeting and more or less one-on-one meetings regarding collaboration on various projects. One of the benefits of attending TeX conferences is the opportunity to do bits of TeX or other business in person.

Stanford TeX developers panel and Knuth presentation

The second afternoon session of Wednesday was a unique event — a panel consisting of Don Knuth and nine of the Stanford students who helped create TeX as we know it today. The panelists, aside from Don (who needs no introduction to TUG members), were (in alphabetic order and mentioning only one or two of their contributions):

- David Fuchs was called "my right hand man for TeX82" by Don Knuth. David also did the initial development of the `DVI` format.
- John Hobby developed METAFONT's polygonal pens and other aspects of METAFONT as part

[1] `http://tug.org/tug2010/`

[2] The hotel staff were very attentive to the needs of the conference, and provided excellent food service for breaks and lunch.

[3] `http://tug.org/tug2010/program.html`

[4] `http://river-valley.tv/conferences/tug-2010`

Table 1: Breadth of presentations

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
First morning														
Ross Moore: TEX+MathML for Tagged PDF					E									N
Will Robertson: Unicode mathematics in LATEX													M	N
Boris Veytsman: Are virtual fonts obsolete?										J			M	
Steve Grathwohl, David Ruddy: Math on the Web: Implementing MathJax in Project Euclid											K			N
Johannes Kúster: Math never seen												L	M	
First afternoon														
Alan Hoenig: TEX helps you learn Chinese character meanings		B							I					
Bill Cheswick: Ebooks: New challenges for beautiful typesetting									I					N
Hans Hagen: Just in Time: Things we can only do with LuaTEX							G	H		J			M	
Hans Hagen: Building paragraphs with the help of Lua							G	H						
Idris Hamid: Oriental TEX: Culturally authentic typesetting of the Qur'an								H	I				M	
Second morning														
Michael Doob: A web-based TEX previewer: Ecstasy and agony									I	J				N
Jonathan Kew: TEXworks for newcomers — and what's new for old hands								H	I	J				
Kaveh Bazargan: Batch Commander: an interactive style writer for TEX										J				
Boris Veytsman, Leyla Akhmadeeva: TEX in the GLAMP world: On-demand creation of documents online									I					
Pavneet Arora: Using LATEX to generate dynamic mathematics worksheets									I			L		
Stephen Hicks: Improving margin paragraphs and float control						F				J				
Second afternoon														
Herbert Voss: From PostScript to PDF					E	F								N
Jim Hefferon: Characterizing CTAN packages								H		J				
Didier Verna: Classes, styles, conflicts: The biological realm of LATEX												L		
Walter Gander: Writing the first LATEX book				D										
William Hammond: LATEX profiles as objects in the "category" of markup languages												L		
Chris Rowley: A brief history of LATEX with a prediction				D										
Third morning														
Uwe Ziegenhagen: Dynamic reporting with R/Sweave and LATEX			C						I	J	K			
John Bowman: Interactive TEX-aware 3D vector graphics			C					H		J				
Mathieu Bourgeois, Roger Villemaire: Introduction to drawing structured diagrams in SDDL			C											
Jean-luc Doumont: Quantum spaces: Designing pages on grids		B										L		
Robert Rundell: Using the Knuth-Plass algorithm to help control widow and orphan lines						F								
Bart Childs: Thirty years of literate programming and more?				D								L		
Third afternoon														
John Hobby: Is boxes.mp the right way to draw diagrams?				D		F								
Hans Hagen, Taco Hoekwater: How TEX and Meta finally got married		B								J				
Frank Mittelbach: Exhuming coffins from the last century		B								J				
Don Knuth & Stanford TEX Project members: panel				D								L		
Don Knuth: A Special Announcement!	A													

Legend for columns A–N

A. Unclassified
B. Book design
C. Graphics
D. History
E. PDF
F. Ideas for new typesetting algorithms
G. Distributions and formats
H. Status report on big on-going projects
I. Applications of TEX
J. Tools and approaches to aid TEX use
K. Systems using TEX
L. Philosophy
M. Fonts
N. More output devices and file formats for TEX

part of his PhD thesis research. Later John developed MetaPost.

- Frank Liang worked with Don Knuth on the hyphenation algorithm for TEX78, and Frank's PhD thesis presented a better hyphenation algorithm which is used in the TEX we know today and in many other typesetting systems.
- Oren Patashnik developed BIBTEX and is co-author of the *Concrete Mathematics* book.
- Michael Plass was co-implementor (with Frank) of the original prototype for TEX. His PhD thesis presented methods for line breaking and pagination with floats, methods which of course are used in TEX.
- Tom Rokicki developed the original Pascal-to-C converter for the TEX system and developed `dvips`.
- Luis Trabb Pardo was called "my right hand man" for the development of TEX78 by Don Knuth. Luis was also involved in interfacing to the early laser printers.
- Howard Trickey did one of the first ports of TEX to Unix. He also wrote the first four BIBTEX styles and related utility software.
- Joe Weening was involved in various ways in the transition from TEX78 to TEX82, and maintained the well-known `labrea.stanford.edu` FTP site.

I moderated this session, in which audience members asked questions of all of the panelists, the panelists reacted to each other's answers and suggested topics about which other panelists should comment, and so forth. The discussion, lasting for an hour and a half, was fascinating. The video of the panel discuss is available[5] as is a written transcript.

Following the panel discussion, Don Knuth took the floor and presented the "Special announcement" that was listed in the conference program and about which there had been speculation at the conference and on the World Wide Web in the days before the conference. Don's presentation, entitled "An Earthshaking Announcement", *must be seen* to be appreciated: view the video.[6]

Banquet

The banquet was held at the restaurant Le Colonial about two-and-one-half blocks from the hotel, and the Vietnamese/French food was excellent.

As in previous years, Kaveh Bazargan MC'd the presentations after dinner and dessert.

First Kaveh introduced Karl Berry who presented a commemorative book created especially for this thirty-second anniversary of TEX to Don Knuth and the other nine Stanford developers of TEX. Karl's and my hack in creating this commemorative book was to mimic the design of *The TEXbook*, including a cover illustration by Duane Bibby and reprints of previous Bibby illustrations throughout the book. The text of the book included a foreword by Barbara Beeton, introductions to Knuth and the other Stanford people and reprints of papers from *TUGboat* by Knuth and the others. Titled *TEX's* 2^5 *Anniversary: A Commemorative Collection*, the book is for sale (with a discount for members) from the TUG store,[7] and is available online to members.[8] The original drawing for the book's cover was given by random selection to one of the other Stanford developers, and went to Michael Plass.

Next, Kaveh managed the usual "soapbox" opportunity for anyone at the banquet to say something about TEX, the conference, ..., with careful timing by Jennifer Claudio and the penalty of going over or under of having to buy Kaveh a drink. For this year, the required interval was 32 seconds minimum to 128 seconds maximum. Several people took an opportunity to speak, notably David Fuchs recalling a bit of history he had not had an opportunity to mention during the afternoon panel. TUG was just coming into existence as *The TEXbook* was being finished. The book already included Appendix A (Answers to exercises), Appendix B (Basic control sequences), and so on through Appendix I (Index). David mentioned to Don that perhaps something about TUG should be included in the book. Don said that sounded good, but David would have to come up with a title beginning with 'J'. David thought about it overnight and in the morning came back with the title for Appendix J: Joining the TEX Community.

Next, Hans Hagen presented an original Duane Bibby drawing to Don Knuth and John Hobby from the LuaTEX team — for Don's and John's contributions that underly LuaTEX with its embedded MetaPost. The LuaTEX team commissioned the drawing showing Don and John proofing a Punk Font[9] sheet just processed by a LuaTEX driven printing press with `mplib` inside.

Hans also presented Don with a gift from the ConTEXt user community: a mockup of the SET® game (made by ConTEXt development group member Mojca Miklavec) and packaged in a special box made by Willi Egger (the ConTEXt group's expert on

[5] `http://river-valley.tv/tug-2010-panel`
[6] `http://river-valley.tv/an-earthshaking-announcement`
[7] `http://tug.org/store`
[8] `https://www.tug.org/members`
[9] `http://tug.org/TUGboat/Articles/tb09-2/ tb21knut.pdf`

bookbinding, printing, and packaging). The set uses the cow font[10] and uses the words "LUA", "TEX", and "MP" in several colors and variants.

As usual, TEX-related vendors provided books and other products to TUG which are raffled off to people at the banquet (all registered conference attendees' names are in the bowl from which names are randomly drawn). This year a copy of each of Don's books published by CSLI was raffled off, along with books donated by Green Lion Press, University Science Books, and Addison-Wesley. Two copies of Jean-luc Doumont's beautiful book which he used to illustrate his conference presentation on "Quantum Spaces" were also in the raffle.

Karl then acknowledged my contributions to TUG, this conference, and the commemorative book, and he gave me a hardbound copy of Knuth's book *Digital Typography* including a dust cover. Don's publisher at Stanford Center for the Study of Language and Information (CSLI) had contributed this special edition to TUG, it being one of only five hardbound copies of *Digital Typography* in existence with dust covers. Karl then acknowledged the contributions of Kaja Christiansen, longtime TUG vice-president from Aarhus University which provides space, electricity, and Internet connectivity for the main TUG server, for which Kaja is co-system administrator. Karl gave the original of the conference poster by Duane Bibby to Kaja. Finally, Karl acknowledged the efforts over the years and especially for this conference of Robin Laakso, TUG's executive director (observing Robin at the conference, there didn't seem to be so much that was "executive" about her job — she was working at a pretty nitty-gritty, hands-on level).

We then heard a violin solo by Zhenya and Morten Høgholm's young son, David.

Don Knuth ended the evening's formal presentations by exhibiting copies of "*A keepsake in honor of TEX's 32nd anniversary, 30 June 2010*". The keepsake was a piece of embroidery of an image of the TEX lion sitting on a pedestal with the annotation, "This souvenir TEX lion was embroidered by a numerically controlled sewing machine using the remarkably simple EULER-TRAIL algorithm at `www-cs-faculty.stanford.edu/~knuth/programs.html`." Don gave copies of the keepsake to the nine other early Stanford TEX people in attendance, and to Barbara Beeton, Karl Berry, Hans Hagen, Jonathan Kew, and Frank Mittelbach for their work in pushing TEX. Finally, Don gave to all of us attending the banquet a 2.25-inch square framed image of his newly announced logo.[6]

I see a certain pattern in the execution of various of the items mentioned above — the items Don presented, the items Hans presented, and the commemorative book. It seems the members of the TEX community don't just use Don's typesetting capability; he is also our model for detail and precision of execution of even one-off projects.

The evening finished informally, with goodbyes, promises to see you next time, agreements to follow-up by email, and so forth. We had all been brought together by TEX and now we were departing, but we would remain connected by TEX and our memories of having spent a couple of days with its creator. John Bowman summed it up this way:

A professor from Stanford nearly went through the roof
On laying eyes on his very first galley proof
  He said "What the heck,
  I'll go invent TEX"
That man's name, my dear friends, was Donald E. Knuth!

All in all, from where I was sitting, the conference was a smashing success.

◇ David Walden
E. Sandwich, MA
`http://www.walden-family.com`

[10] `http://tug.org/TUGboat/Articles/tb27-1/tb86hoekwater-cows.pdf`

An Earthshaking Announcement

Donald E. Knuth

Ladies and gentlemen, distinguished guests, dear friends: How appropriate it is for us to be meeting here in the city where Steve Jobs has made so many dramatic announcements. Today I have the honor of unveiling for you something that, in Steve's words, is "truly incredible" — a successor to TEX that I've been working on in secret for quite some time.

All of us know that computers and the Internet have been changing the world at a dizzying pace. Consequently few, if any, of the assumptions that I made when I first got TEX to work in 1978 are valid today. Day after day I've been becoming more and more convinced that a totally different approach is now needed. Finally I woke up one morning with the realization that I couldn't be happy unless I came up with a new system that rectifies my former mistakes — a system that leads to real progress.

Thus I've decided to scrap TEX78 and TEX82 and to start over from scratch. Of course the first thing that I wanted to fix was the most egregious design error that I'd made in the early system: The old TEX was internally based on binary arithmetic, although its user interface was entirely decimal! Thus one could write, for example,

```
\ifdim .4pt = .39999pt \message{yes} \fi
```

and get the response '`yes`'. Furthermore a construction like

```
\dimen0=.4pt \multiply \dimen0 by 10
\showthe\dimen0
```

would give the answer '`3.99994pt`'; how ridiculous can you get? Are mathematicians supposed to like this? Are computer scientists supposed to like this? Is anybody supposed to like this?

(By the way, I apologize for using handwritten overhead-projector slides in this presentation, instead of making PowerPoint points. Some unexpected problems arose with my computer at home, and I didn't want to risk security leaks by putting any of this material on someone else's machine.)

Returning to my story, many of you may recall that the old TEX represented all dimensions as integer multiples of a so-called "scaled point", defined to be 1/65536th of a printer's point, where a printer's point was defined to be exactly 1/72.27th of an inch. How bizarre and anachronistic! Nobody remembers or cares about the old-fashioned units that printers used in the pre-Internet era. The graphic designers of today all know that there are exactly 72 points to an inch, as specified by Adobe Systems; why should TEX insist on calling that now-universal unit a "big point"? Indeed, what relevance will points of *any* kind be to anybody, ten years from now?

Moreover, TEX never has allowed dimensions to exceed `\maxdimen`, or 16383.9999847412109375pt, which is roughly 18.8921175 feet (5.7583 meters). Today's graphic devices make posters and banners much larger than this, so TEX cannot cope.

At the other extreme, advances in nanotechnology mean that TEX's minimum dimension of one scaled point is far too large to accommodate 21st-century applications: A scaled point is huge, more than two farshimmelt potrzebies; it's more than 53 Ångstrom units, 'way bigger than a hydrogen atom.

Thus I'm pleased to say that my new typesetting system finally gets it right: Dimensions can be arbitrarily large or arbitrarily small multiples of internationally accepted units. They can be expressed as exact rational quantities, like 3/7 of a yard; they can also be expressed in terms of irrational numbers like π and $\sqrt{2}$, so that circles and other objects can at last be rendered with perfect accuracy.

My design from 32 years ago was heavily influenced by what we used to call "efficiency". I didn't understand the implications of Moore's Law. I didn't realize that, in a few years, I wouldn't care whether *The Art of Computer Programming* could be typeset in half a second, rather than waiting five seconds.

Examples of my tunnel vision abound, on almost every page of *The TEXbook*. For example, I used backslashes and other strange characters to define what I called "control sequences". Does any other system you know have control sequences? Of course not.

With my old rules people never knew whether or not a blank space really meant a blank space.

Therefore the basic input language for my new system is entirely a subset of XML, a widely accepted standard. However, XML is really only necessary at the lowest level, and most users won't need to be aware of it, because we'll see in a moment that there are many other ways to provide input.

Of course the character set for my new system is Unicode, so that there is 100% support for all of the languages and metalanguages of the world. Automatic spelling and grammar correction are built in for each language, as well as automatic correction to page layout and design. Different languages can freely be intermixed at will, always with appropriate ligatures, kerning, and hyphenation.

The old TeX was limited to left-to-right typesetting; and some of its extensions also now handle the right-to-left conventions of many languages that my original implementation didn't consider. But my new system has been designed from the beginning to produce output in any direction whatsoever, whether horizontally or vertically or diagonally or along any kind of curved lines.

In fact, since 3D printing technology is now widespread, I decided at the outset that there was no reason to limit my new system to only two dimensions. Three dimensions are now standard in the new system; in other words, we deal with voxels instead of mere pixels. I've also provided hooks to allow future extensions to four or more dimensions, in case the string theorists prove to be right.

From a virtual standpoint, the notion of hypertext already gives us the equivalent of unlimited dimensionality, and the production of hyperdocuments and web pages will be one of the chief thrusts of my new system. TeX's old principles of boxes, glue, and penalties turn out to yield fantastic new ways to create multimedia documents, including animated videos and stereophonic sound.

Indeed, the input and output aspects of the new system aren't confined to traditional forms of text. Audio input and camera input are now seamlessly integrated, as well as sensor devices of all sorts. The system uses your GPS coordinates intelligently, if you are a mobile user, and senses your motions and gestures with accelerometers, etc. Complete support of haptics is also fully implemented. Instead of "what you see is what you get," we now also have "what you hear is what you get," and "what you feel is what you get."

There really is little difference between input and output in the new system, because any input can be output; conversely, any output from one hyperdocument can be input to another, or to itself. For example, music can be input from one or more MIDI devices, then either output to a conventional printed score, or to another MIDI device or group of devices — optionally segmented into individual parts, or transposed, or whatever you want. Going the other way, a printed score can be used as the input to a synthesizer, etc. I'll say more about these dynamic aspects later.

Does my new system have macros? No. Macros are *passé*; they're *so* mid-20th-century. Nowadays no one really needs macros, which we all know can be difficult to write and even dangerous. Everything in the new system is menu-driven, somewhat in the style of "Microsoft Word" but considerably enhanced: Experts have prepared recipes for everything you'll ever want to do, and these features keep growing and getting better and better. The menus needn't appear on your computer screens in traditional pull-down or pop-up form; my system also responds to spoken commands and to gestures. And it quickly learns your preferences, so that it's customized to your own wishes, thereby making document preparation almost instantaneous.

You may have noticed that I've been referring a lot to my new system, but I haven't yet told you its name. I had to explain some of its characteristics before you could fully understand the name. But now I'm ready to reveal it, and more importantly to show you its logo:

iTEX*

How does one properly pronounce this name? Listen carefully: "ĭ-téx" ["ĕe-técks"]. In the first place, you'll notice that it should be said musically, with *tones* as in Mandarin. (The first vowel is spoken with a dipping tone, "ĭ", where the pitch falls and then rises. The second vowel has a strictly rising tone, "é", almost as if you're asking a question.) In the second place, you'll notice that I've also rung a bell when saying the name. The bell is also part of the logo:

iTEX*

It reminds us that iTEX* is not limited by obsolete conventions, not hampered by the days when documents were only seen but not heard. (However, the bell is optional, and it is omitted in documents that have no audio. Conversely, the logo is actually three-dimensional in a 3D document.) In the third place, did you notice that I said "tecks" instead of "techhhh"? I've decided to go with the flow, since almost nobody outside of Greece has ever pronounced 'TeX' with the correct 'X' sound.

Some of you may recall that I wrote the entire program for TeX78 and TeX82 all by myself, and you may be wondering whether I've done the same for iTEX*. Don't worry: This time around I'm having the job done by people who know what they're doing. After many years I've finally come to realize that my main strength lies in an ability to delegate work and to lead large projects, rather than to go it alone. Programming has never really been my forté — for example, I've had to remove 1289 bugs from TeX, and 571 from METAFONT.

I made a very fortunate discovery during the summer of 2006 when I visited the Academy of

* [A bell rings at this point.]

Sciences in Armenia. There I learned that a huge amount of highly sophisticated but classified defense work had been done secretly at a large institute in Yerevan during the Soviet era. I met many of the people who had participated in those activities, and found that they were extraordinarily good programmers. Moreover, they were anxious to apply their skills to new domains. So they were a perfect match for my desire to make iTEX* a reality.

I had long envied my colleagues at Stanford who had started up their own companies and gotten rich. Now it was my turn, and without much difficulty I formed a clandestine group called *Project Marianne*, comprising more than 100 of the top programmers in the world.

A few weeks ago some of you apparently discovered our website portal at `projectmarianne.com`. I've also seen blogs that wondered about the Armenian letter M on that page (Unicode #0544, "Men"). But as far as I know, nobody outside of our group has yet been able to penetrate the firewall that we built into that site, nor to look at any of our planning documents or initial demos. Needless to say, I'm pleased at this success, because the ability to create secure documents is another feature of iTEX*.

Furthermore, I believe that nobody else has realized until now that 'Marianne' is an anagram of 'Armenian'.

After some deliberation, our group decided that all of the code for iTEX* should be written in Scheme. We also decided to guarantee success by using all of the silver bullets that have been discovered by software engineers during the past decades: Information Hiding, Agile Software Development, Extreme Programming, Use Case Modeling, Bebugging, Look Ahead Design, Waterfall Modeling, Unit Testing, Refactoring, Rapid Prototyping, the whole shebang. We're going beyond ordinary Object-Oriented Programming to Aspect-Oriented Programming. But we're not using any formal methods, because everybody knows that formal methods are strictly academic. And we're abandoning the old notion of "literate programming" that I used when developing TEX82, because documentation has proved to be too much of a pain.

Naturally it's out of the question for a system like iTEX* to be freely available and essentially in the public domain, as the old TEX system was. These talented programmers certainly deserve to be paid handsomely for their hard work. We have therefore devised some innovative pricing strategies, so that I'm sure you will consider iTEX* to be an unbeatable bargain, considering the enormous value of its new features.

Here's the way it will work: Payments will be by monthly subscription, which will entitle you to unlimited use of iTEX* on one or two of your own computers, or up to 40 hours × 16 gigabytes of computing in our cloud of approved service providers. During the first year we're offering a one-month free introductory trial; thereafter your costs will depend on the quality of Internet access that is available in your area. For example, California users will pay $99 per month, and German users will pay €69; but the monthly fee in Armenia will be only ֏1999.

There are substantial discounts for senior citizens and for children under five years of age, as well as educational discounts for students.

Moreover — and this is the main innovation — you get a 10% discount for every new member that you can convince to join, lasting as long as you and that person are both enrolled in the plan. Thus if you can sign up just ten new subscribers, your access to iTEX* will be free; and if you bring in eleven, you've essentially garnered a lifetime income.

My new enterprise operates by monthly subscription, instead of actually selling copies of the software, in part because the software is proprietary, but mainly because iTEX* will change every day, due to constant improvements and upgrades to the system. Once upon a time I took great care in order to ensure that TEX82 would be truly archival, so that the results obtainable today will produce identical output 50 years from now. But that was manifestly foolish. Let's face it: Who's going to care one whit for what I do today, after even 5 years have elapsed, let alone 50? Life is too short to reread anything anymore; in the Internet Age, nothing over 30 months old is trustworthy or interesting. We're best off just enjoying each moment as it happens.

iTEX* will benefit the entire world's economy, because it will lead to tens of thousands of new jobs. For example, independent developers will be able to design and sell plugins that are distributed online and available for only a few pennies per week. Any iTEX* user will be able to sell his or her own documents online, without leaving the iTEX* system, because iTEX* includes facilities for ordering, billing, manufacturing, and shipping. You can, for instance, write a blog, and others can package as many chapters of that blog as they wish into a customized book that is nicely printed and bound. iTEX* will collect the appropriate payments from each customer and divide them fairly between you, the printer, the binder, and the shipper; the finished book will then arrive promptly at the customer's residence. The operation will be something like the old Sears and Roebuck catalog, but now each item will be custom-tailored to an extent never before seen.

More importantly, there will be a large network of certified iTEX* consultants, at various graded levels of certification. iTEX* has no user manual, in the old sense, because the system changes daily. But it does have three varieties of online help: There's online help for dummies, online help for wizards, and personalized online help — in which you get to chat one-on-one with a certified iTEX* helper. (Your membership fee entitles you to an hour's worth of one-on-one help each month.) Such helpers can arrange to work part-time for the iTEX* consortium, out of their own homes and with flexible hours, in order to supplement their other income.

Let me conclude by describing a few more of iTEX*'s features, so that you can begin to get a glimpse of how truly revolutionary it is. I've already told you that dimensions can be specified as arbitrary multiples of standard units; but that's just a tiny part of the story. iTEX* actually is able to do arbitrary symbolic calculations, with polynomials and power series and matrices and partial differential equation solvers and convex optimization, etc., all integrated with graphics for automatic curve plotting and statistical charts, together with maps and satellite photographs of the world. When combined with iTEX*'s synthesized voice output, you can do things like find a shortest route and navigate your car, all as part of an iTEX* hyperdocument. If you're a professor like me, you can write math texts in which the formulas are changeable by each individual reader, who can evaluate them and plot their graphs interactively. (Incidentally I've changed math mode so that formulas must now be specified unambiguously, in such a way that they can be evaluated as well as printed; think MathML. This makes the formulas longer and more difficult to type, but that's a small price to pay for the added functionality.)

The hyperdocuments of iTEX* can have any number of users, who can interact with each other and render images of themselves as avatars. This capability goes beyond the traditional kinds of virtual reality that are offered by systems such as Second Life®, not only because of iTEX*'s haptics but also because iTEX* uses hyperbolic geometry — in which exponentially many avatars can be within a bounded distance of each other.

Such interactive documents obviously enable videoconferencing as a simple special case. I mentioned earlier that iTEX* can receive input from all kinds of sources: news feeds, webcams, traffic and weather sensors, heart monitors, seismographs, astronomical observations, you name it. All of these can be captured, mixed, and/or converted to other forms, such as audio or video or both. World-class tools are provided for photo retouching and image processing, computer-aided design, character and face recognition, as well as sophisticated filters for all sorts of data — including, for example, audio tracks and email. Output can be automatically formatted for lasercutters, embroidery machines, 3D printers, milling machines, and other CNC devices ... and shipped directly to consumers, as mentioned earlier.

One of our early plugins will feature an interactive cookbook that interfaces directly to your kitchen stove, oven, pantry, and refrigerator, so that you can prepare meals automatically with the ingredients that you already have on hand, and/or replenish your supplies by online ordering.

iTEX* naturally incorporates extensive facilities for social networking. You can easily read the hyperdocuments prepared by others, and it's even easier to send and receive "tweets". (Your tweets needn't be limited to 140 characters of Unicode; the actual limit is a parameter. For example, you can set things up so that you receive only tweets of 50 characters or less.) With iTEX* your entire life can be encapsulated into a dynamic hyperdocument, downloadable by anybody you designate.

I had intended to give you a live demonstration of iTEX* today, instead of merely talking about its features. Indeed, iTEX* was supposed to have provided all of my slides for this lecture, because the illustrations for a technical talk are among the simplest of all documents to create. Unfortunately, however, that has turned out to be impossible, because of hardware glitches and breakdowns in communication that I had no way to anticipate. (You can well imagine how difficult it has been to get all the pieces of iTEX* to work together.)

But my coworkers assure me that the system is almost ready for its first major release, and we plan a worldwide press conference when iTEX* is officially launched — hopefully next month.

Well, I've got to stop now: I can't tell you any more until our patent applications have all been filed. But I'm sure that, once you've tried iTEX*, you'll immediately want to become a charter member of ĭTÙG*.

◇ Donald E. Knuth
Founder of *Project Marianne*

TUG 2010 Panel: Don Knuth & Stanford TEX Project members

David Walden, moderator

[This transcript has been lightly edited. The session is available on video at `http://river-valley.tv/tug-2010-panel`.]

Karl Berry. I will briefly introduce our panel moderator, Dave Walden, who as you know handles the Interview Corner and all kinds of interviews ... so I asked him to handle this one too.

Dave Walden (moderator). Thank you, Karl. I was really pleased to be invited by Karl to chair this panel, for three reasons.

First, despite the fact I've lived in Boston for 46 years, I grew up in the San Francisco Bay Area and graduated in mathematics from San Francisco State. So, it's a real pleasure to be back at a math meeting in San Francisco. This is probably the first time I've thought about math in San Francisco in 46 years. It's good to be home.

Second, I've always admired Donald Knuth. I bought his *Art of Computer Programming* in the late 60s and used it in my daily work. When volumes 2 and 3 came out, I immediately bought them, and we used those in our daily work. More recently, when I decided to stop using Word and go to some kind of a text processing system that didn't have hidden proprietary undocumented markup, I chose TEX because I admired Don Knuth and I thought I'd like to try something that he created.

And, of course, the third reason is that that brought me in contact with this community, and through interviews and so on with everyone on this panel. So, third, it's a real pleasure today to get to get to meet everyone here today in person.

With that I'd like to introduce the panel members. I'll first mention Don. It's a cliche to say, "he needs no introduction", but with this group and this man, he needs no introduction. I'm sure each of us knows of several things in his massive set of accomplishments in a variety of areas. And, in fact, his publisher conveniently gave us this list [holds up advertising page from CSLI] of nine different books of his collected works in different areas, and that does not include *The Art of Computer Programming* which has a different publisher.

So I'll go through the rest of the panel in alphabetical order, and I will introduce you to them saying a word or two about their TEX accomplishments. Naturally, they have had full careers in other areas and have done many other things, and I commend the interview series to you. And the couple of you [panelists] who have not yet participated — it's time.

[At this point the moderator introduced David Fuchs, John Hobby, Frank Liang, Oren Patashnik, Michael Plass, Tom Rokicki, Luis Trabb Pardo, Howard Trickey, and Joe Weening using essentially the descriptions of the "TUG 2010 Conference Report" on page 117 of this issue.]

Regarding the format of this panel, Don asked me to say, "I am going to have a half hour to myself later. This is the only time the rest of the panelists get to talk, so please focus as many questions as you can on the rest of the panel" — said Don, and I say that to you [members of the audience]. What I'd like to do is go through the panel one-by-one. I won't do a strict rotation but let's have a question for each panel member before we go to a more open format. And, panelists, I encourage you, if you have something to add to an answer of a fellow panelist, please chip in. I think that hearing different sides of these historical stories is often interesting both because it elaborates on the stories or sometimes it shows some conflict in the stories.

May I have the first question from the audience. Oh, one more thing, unless we have a mike in the audience, please say the question loud enough for me to hear and I'll then repeat the question for Kaveh's videotape.

Unknown. The question for Frank is, "How did you discover your hyphenation algorithm?"

Frank Liang. Well, I was assigned this problem as a thesis in 1978, I believe. As I have mentioned in my thesis, there was an initial suggestion to use a kind of statistical algorithm which looked at two letters — well, actually four letters — surrounding a potential break point, and then you were supposed to make tables using two letters before, the middle two letters, and the last two letters, and then combine these tables somehow to do hyphenation. So I started experimenting with some word lists, and I quickly found that that wasn't sufficient; you needed more context. One example I mentioned in my thesis is that sometimes a letter seven letters away from the hyphen point can alter the breakpoint.

Anyway, I'm playing with word lists a lot and came upon the idea that just patterns of letters was a very simple way. Because I had started with just these two letter digrams, it was natural to extend that to longer letter sequences. And then through a long process of evolution I came up with patterns and then with the rules and exceptions. Don actually came up with the idea of assigning the numbers and having them at one of my thesis review meetings. I sort of had the idea of having rules and exceptions, and he said, “Oh, you could assign numbers to them.” So that’s part of the answer.

One of the hardest parts of the whole thing was acquiring a suitable word list. I didn’t have access to every database, and there weren’t that many at the time. I got a copy of the Merriam-Webster dictionary that had hyphenation points. But upon looking through it, there were many errors and lots of typos and things that just weren’t quite right. So I had to hand edit the dictionary, and that took about three months. So that is where most of the work was actually.

Moderator. Any of the other panelists have a comment on this?

Howard. And the tries ...?

Frank continues. The tries — that sort of came up separately, after the pattern idea. As I was playing around with the word lists all the time, I needed some kind of relatively fast algorithm to quickly collect information about the patterns in the word lists with hyphenation points and then to test out various theories. Because these were just simple strings, it was natural to look at variations of standard data structures like tries for that and to read related papers like Don’s on pattern matching.

The problem with the tries — tries are very fast because it is just based on indexing, but they tend to be very sparse so you then have to use various tricks to speed that up. And the idea, actually, for doing this weird packing, if that is what you are referring to, was I read another paper by a Stanford professor at the time, Andrew Yao, which was talking about storing a sparse table. It was a somewhat different application, but he had this idea of when you have these sparse things you sort of interleave them all in one array and thereby save space while maintaining speed. So that’s where I got that idea.

Moderator. A question for another panelist?

William Adams. My question is for Tom Rokicki. Ages ago, when the WorldWideWeb.app was written on the NeXTstep, there was also available on that same platform TEXview.app, and we’ve seen an awful lot of effort in trying to get mathematics and nice fonts and nice settings onto the web. Why didn’t we just start out with an extended version of hyperTEXview.app and cut to the chase?

Tom Rokicki. [to the moderator] I’m sorry. Can you repeat the question?

Moderator. I don’t understand the question. Maybe William can say it again, and I can try to repeat it.

William. We started out with TEXview.app on NeXTstep and on that same platform, WorldWideWeb.app was developed by Sir Tim Berners-Lee. Why wasn’t TEXview.app used as the basis for the WorldWideWeb.app so that mathematics and so forth would have always just worked instead of us constantly working to try to make them work on the web?

Tom Rokicki. Basically, you’re asking why TEX was not chosen as the basis for mathematics on the web, based on the NeXTstep. Boy, I’m not really the right person to ask. Gosh, you know, I really don’t have a good answer to that. The HTML stuff was really crude in the beginning, and it’s still pretty crude, but it’s getting there. So I don’t really think anybody spent a lot of focus at that time. Back then, it was just “let’s get the links working, let’s get the text working, let’s draw around images, and stuff like that, and call it a day.” Which was pretty amazing in itself. As far as what went after, I really can’t say.

I’m not sure that it really fits though. Because the web, HTML, is all XML-based. And TEX is rather different. And I think there was a very strong reason to keep it as a markup language like SGML or XML, that could be easily automated, and restructured, and all this type of stuff. So I think that there were good reasons why TEX was never used at that point. But I was never really part of that, so I can’t say for sure.

Moderator. Another question? Nelson.

Nelson Beebe. Question for David Fuchs. It’s important to remember that in this audience there are a lot of younger people here who have grown up with laptop computers, that TEX was designed on a machine that cost roughly half a million dollars at its entry level price. And there were two people, one of whom is sitting here, who really changed that and made life different for an awful lot of people, and Dave Fuchs is one of those; the other one is Lance Carnes, who unfortunately isn’t here today. I’d just like Dave to comment on his work with MicroTEX, which brought to you the first live preview, while TEX was running, of what was happening, and how difficult it was, and how much hair he lost trying to get TEX to work on the little machines of the day.

David Fuchs. Well, that was fun. It was a lot of kind of hackery and trickery down at kind of the bit level, and the intricacies of the 8086. At the time, there were no good Pascal compilers, if there ever were, ..., well, that’s not true — there was a good DEC one. So that work involved writing kind of a limited Pascal-to-C translator that had special hacks — it didn’t bother with the parsable language that TeX didn't use, it had special hacks in it so that I could insert using the change file scheme from WEB. I could insert magical keywords that I had laid out that said, oh, here's one of the big arrays, and then I could write some assembly code.... Gee, there was even a version, there were different instantiations of it. One early version, I just pretended to the C compiler that the arrays weren't really big. It always produced one of three different sets of machine code instructions to address those arrays; I had a post-processor that would look through the object code that the compiler created, look for those three or four possible patterns, insert some special other code that called some of *my* code to look at the real array thing, ... That only lived for a version or two. Then there were other cute tricks: it kind of used a sort of VME system where I chopped up the arrays into pieces, and those got swapped in and out of, I think off of the disk. I even stuffed some of them, as an experiment, into the ... it turns out the video cards that you had, that drove your displays, they had some extra RAM in them, so you could swap stuff out, and it was kind of fun, 'cause if you put the video card into a different mode you could actually see that stuff on your screen going by.

So, the early PCs had a maximum of 640 K, and a lot of people for a while had 512 K, so that was marginally not quite enough to do everything, so you had to replace all the run-time libraries that came with the compiler. I was down to ... I wasn't even using ... what's it called that's usually linked in with the C program, even the startup code, so you got ... there was no standard files, there was no anything. So that was all to cram it down into 512–640 K.

Lance did a job too. Because his stuff was rather commercial — that's his livelihood — he was always somewhat circumspect about it. But, obviously, it wasn't complete black magic.

Tom also had some work in this direction, if I remember correctly?

Tom. Absolutely not, absolutely not. [laughter] This accomplishment of David Fuchs' was one of the most amazing things I've ever seen. All my platforms had *plenty* of memory. I never had those issues. I cannot *believe* what Dave accomplished — it was absolutely amazing! Okay, truly a hero.

Moderator. Anybody else have experiences with these tiny machines?

Another question, please. Hans.

Hans Hagen. Aren't you somewhat disappointed after 32 years that not more people made fonts using METAFONT?

Don Knuth. Well, I can't say I'm disappointed in the fonts we have now. And I'm happy with the ones that my students have made, and I made. So I'm not .... When I wrote it, I just had the idea ... everybody's entitled to have some mistakes in their life, and so I didn't have to worry about the fact that not everybody would use every program I wrote, and this one happens to be a very personal thing, and so the way I look at it is, how wonderful that John extended it to METAPOST, which I use a hundred times more than I use METAFONT. I've already done most of what I ever need to do with METAFONT.

Hans. But didn't you overestimate the font designers then?

Don. Well, I thought it would be easier to teach the font designers about the notion of parameters and metadesign than it was. Computer scientists, we're used to writing something that's going to work under many different conditions as the parameters change. But to most of the rest of the world, to my big surprise, they never heard the word "parameter" — they thought it meant "perimeter".

Moderator. John?

John Hobby. Yes, I certainly agree that the METAPOST application was more popular, but Don has a very unusual set of skills, and indeed, there aren't many other people who are good at that kind of thing. I think the real thing is, it's just too hard to create a really meta-font, as far as the art community is concerned.

Moderator. Any other comments on that from the panel?

Okay, another question — for Oren, or Michael, or Luis, or Howard?

Karl Berry. I have a question for Howard, which is, I've spent a lot of my life looking at TEX.CH, and at the top in all those change files, pretty much all of them say "Howard Trickey and Pavel Curtis". I see Howard Trickey, who I never quite understood was at Stanford before this conference. I wonder if you could tell us if you worked directly with Pavel, or if it was two independent things, or how that came about.

Howard Trickey. This is pretty interesting. The fact that that change file has my name in it meant for many years it was really easy to find me on the World Wide Web. There was, like, 300,000 references. So thank you all for putting that page up on the web.

Don had done his thing on the DEC computer, and I had worked on VAXes — I didn't like this computer. And I wanted this thing to work on VAX.

Moderator. The VAX was from DEC. [laughter from the audience]

Howard. You're right! I'm sorry!

The DEC 20 as opposed to the VAX. And so I did the work that was necessary, which turned out to be evil, although not nearly as evil as the things Dave had to do because he had to change the Pascal compiler default clauses to fool around with the [inaudible]. And I had to do the change file that did the system-y stuff that Unix needed, so that's where that change file came from. And then I found out, hey!, this guy named Pavel Curtis had done the same thing, unknown to me. It happened almost simultaneously. So we were hooked up together and cooperated together.

Moderator. Anything else on that from the panel?

Don. Can I ask a question of Michael Plass, to describe his experiences in 1978 when I went to China and asked him to implement the prototype of TEX.

Michael. Don had this trip to China, and he left Frank and me with a few pages of what his ideas were for implementing — what he would like us to implement over the summer. (I actually wonder whether I still have those pages somewhere. I tend to keep stuff so it's possible — that would be interesting.) Frank was tasked with the hyphenation, and I was tasked with building up starting with storage allocation, I guess (the very bottom), the macro processor, and up through ... I think by the time he got back it was about ready to start implementing some of the line breaking stuff. I guess Frank was also working on the output — the printer driver end of this so we were able to make some prints by the end of that summer. One thing I remember is with the macro language the way Don had spec'd it out, I did some experiments, and he decided it was too powerful — that you could get too tricky with it and do too many things. So he redesigned it to be much more token oriented than it was in the original.

Moderator. Frank, do you have anything to add to that?

Frank. Well, what I remember in addition to the hyphenation which actually I did while Don was here was that, after looking at his notes, Mike decided — he was sort of in charge — we decided to split it up and I would do the output and he would do the rest. And I said this didn't sound like much. At the time he thought output sounded pretty difficult because maybe he didn't know how to do it right off the bat, and of course I had already been playing around with the XGP so I knew how to do it. So for me actually it wasn't that much work and he ended up with much more than he thought. What he gave me was a list of boxes, graphics boxes, and I said, okay, I'll just put them on the printer so that wasn't much work for me. Obviously we way underestimated how much work it was going to be and it was two more, or several more years of Don's work later.

Moderator. I have a followup question. In Michael's interview, on the TUG web site, he says that after Don got back, then he rewrote it all. And so my question for Don is, you didn't like Frank and Michael's work?

Don. Oh, no, actually I liked it, although there were basic changes made. I think control sequences were sort of considered as one character at a time instead — this tokenization idea was coming along at the end — so really the main thing is it gave me the idea for an architecture for the program. But I never expected that I was actually going to use that exact code. I wanted to see it in place; I wanted to see how big it was, what kind of subroutines you needed, and things like that, so that was a key step in getting going. But I knew my sabbatical year was coming up, and that during that time I would ... I always intended to look at what they had and then work over again, and say, okay, now back to square one. Now we know what it's going to be like. Now let's design the right data structures that go with this kind of architecture.

Moderator. Question for Oren or Luis or Joe?

Boris Veytsman. Question for Oren. I always wanted to know, what was the inspiration for the style of BIBTEX language?

Oren Patashnik. For the style of the BIBTEX language?

Boris. No, for the BIBTEX language itself — for the `.bst` files.

Oren. Leslie Lamport needed somebody to do a bibliography program to go along with LATEX, and his idea was to use, sort of as a model, Scribe — it had a bibliography program. So he and I sat down, and he had some ideas about things that he'd want in this `.bst` language. So he and I sat down and kind of discussed it, and I was the one who implemented that. So BIBTEX itself is really, to a first approximation, an interpreter for this `.bst` language. That's really what BIBTEX is.

Leslie Lamport — I think the main ideas for what (he had thought about it before) what was going to go into that language, came from Leslie. And then I implemented it.

I just wanted actually to follow up, in addition to the discussion of literate programming from earlier [Bart Childs's presentation, "Thirty years of literate programming and more?", pp. 183–188], I sort of thought about this a little bit over the break, and I went up to my room and got this — it's my copy of `bibtex.web`. I think `tex.web` is probably the largest piece of software in `WEB`, and `bibtex.web` is maybe the second or third, I'm not sure. But it's a third or maybe two-fifths the size of TeX. Actually, I hadn't — all the stuff I've been doing with BibTeX since, has been looking at the stuff with the `.bst` language, and not with BibTeX itself. The only bugs have been really, really minor, except for one; there was one kind of majorish bug, which is that it didn't handle URLs.

Well, back when BibTeX was written, there were no URLs, so I didn't have a chance to test it out on that. I think I misunderstood something that either Don said, or Dave said, I don't remember, about how control sequences are handled, and so basically, BibTeX doesn't handle the line-breaking right for very long URLs; that's the issue. And so finally, I was convinced that I should . . . , and rather than release a version of BibTeX with all these kind of minorish things, that's probably not worth wasting people's time to install a new version for that. Rather than just doing that, I thought I would finally release a new version of BibTeX that had as its only change that change to how BibTeX handled the long URLs. And so I recently did that a couple of months ago — I was working with Karl. And this is getting back to the literate programming in `WEB`. Every time I look at BibTeX itself — sometimes I look at the bits on my computer, sometimes I look at the hard copy — every time I look at it I think, kind of, what's going on here? But pretty quickly I then sort of get into it. But I hadn't really had to make a change before, until this time. And I realized . . . same experience, I looked at it, I knew I'd taken a peek at this, and I thought, well it's not completely trivial, so I'll do this eventually. So now is the time to do it; we finally decided now's the time to fix that bug.

I looked at the code, and after not very much time — it always takes me a long time to do context switching — I looked at it and said, oh, gee!, the structure of what's going on in the program became completely clear. It was like I was in a zone; you hear athletes talk about "being in the zone", a basketball player, all of a sudden the basket looks huge, and it's easy for them to make a basket, or a baseball player, the pitch coming in from the pitcher looks like a grapefruit, and it's easy for them to hit it, or a soccer player knows they're going to have a 28-yard direct free kick, bend it around the wall and bury it into the upright corner of the net. I mean, talk about athletes getting into the zone, and I sort of felt like that was the experience I had here: After a little bit of looking at this program, all of a sudden the structure and what was going on became completely clear, and it was really easy for me to make the change.

I had just switched computers, so I didn't actually have a TeX implementation on my computer, and I had to use Karl as my debugger — KBDB or something like that — so in the change I made there was one minor mistake. So it took two passes. But I was amazed at how, initially you look at this code and think, what's going on? But very quickly I completely realized the structure.

I think what happens is, when you do literate programming, it imposes in your mind a map of what's going on in the program. It had been 22 years since I'd looked at this code, I think; after 22 years, it didn't take very much, and all of a sudden it was crystal clear. I've never felt that experience before. When looking at `.bst` code, that doesn't happen to me. [laughter] But with a literate program, I think it's because literate programming imposes on you a structure that, even when you haven't looked at it for 22 years, it comes back; it's still there.

Moderator. Don, you have something to say?

Don. Well, speaking of literate programming reminds me of a question for Joe Weening, because it was Joe who suggested the idea of mini-indexes that I used in the `TWILL` program for literate programs, and Joe made some mockups, so maybe he can remember something more than I can.

Joe Weening. I remember them, but I don't think there's much to say beyond what you just said. It's a pretty simple idea: looking at a page of a literate program, there'd be a lot of names you hadn't seen before — names of variables, names of other sections, and so on — and so rather than go from each page to an index and then to another page, the idea was, let's go there directly. Of course, this was before hyperlinking. You must have heard about hyperlinking! Nowadays, what you'd want to do, you would be looking at this on line, and you'd just click on something. That's probably what [. . .]

Don. Certainly we do have the hyperlinks and the clicking now, but a lot of times there's still a value for this in hard copy, when you have a book and you're sitting in your chair, or you want to keep coding without clicking to the other part. But the thing was, you not only suggested the idea, but you also showed a really nice way to present it. I guess it seems simple to you, but it was a real revelation to me.

David. `TWILL`?

Don. Yeah, TWILL. It's on my web site. It's not that easy to use, so I don't advertise it much. It requires running in several passes. When you have a literate program and you have a variable called '*x*', there might be thirty variables named '*x*', so you have to disambiguate which one you're talking about, at least when you write code the way I do, which is maybe not the best. So you have to go through several passes, and then give hints, and say "No, I didn't mean *that* '*x*', I meant *this* '*x*'." And you have to tell it to say "This is something in such-and-such a C library." So there's a bit of hand-tuning that goes on, and it's not a trivial thing. I just went through a book coming out later this year called *Selected Papers on Fun and Games*. In there I have a hundred pages — I took the original program of "Adventure", the cave game, of Don Woods, and I rewrote it as a literate program. And it appears with these mini-indexes, but I had to go through carefully and do it. But the original program used to do Volume B and Volume D, you know, *TEX: The Program*, *METAFONT: The Program*, it was called TWILL. And now I have CTWILL.

David. Wasn't there an early version of WEB? Did Luis work on it? Wasn't there a version of WEB that was before WEB?

Moderator. Perhaps one of you could speak a bit about that? Luis, perhaps?

Don. Yes, I was going to ask Luis about the original ... I mean Luis was involved with this project so early on that you don't know any more all the key things that happened. There are, for example, questions of how did we port TEX to a hundred different computers and get the tapes out and everything like this. People were asking about Maria Code the other day, and somebody said they didn't know if she was a real person. And also, you might be able to also speak as to what Ignaki [Ignacio Zabala] did — he's the main person of the original team who isn't here today.

Luis Trabb Pardo. The question about the literate programming, it was originally a much more mundane thing. We were distributing tapes, and people wanted to know where was the "Main", where does the program start? I said, "At the bottom." And my primary function at the TEX Project was to answer the phone. And I essentially answered that question, "It's at the bottom." In some discussions with Don .... Also, there was the issue of documentation. So, the idea of blending that came very naturally as the necessity to not answer questions but have the questions essentially be answered by what you said. That was the suggestion; essentially Don did the whole thing.

The issue about what Ignaki worked on originally was to start thinking in terms of graphic objects that were going to be put on a workstation. We were in an environment at Stanford that was a timeshared system, and we didn't have much ... well, we did have interesting things going on there, but the concept of a resource available to you like we have today, on our desktop, on our personal computers, was not there. So he started working from that component. He actually put together a system that had all kinds of graphic things. He called them graphic objects. And you could do what you could do today in a display environment on an existing system. I think that answers more or less the level of what Ignaki did.

Unknown. Was Maria Code a real person?

David. I believe that was Ron Code's wife. I remember dealing with Ron more than Maria. The ARPAnet was there, but that was only academic and government institutions, so it used to be, you'd send in a tape to, I think, Ron and Maria Code, who were entrepreneurs, I suppose; I don't know how they hooked up with us. And they would spin off a copy for you and send it.

Unknown. So they weren't members of the Stanford CS department?

David. No, no, I saw them in person.

Moderator. The statement from a member of the audience [Gio Wiederhold] is that Ron [Code] worked for him in medical information systems.

Gio Wiederhold. Maria did all the hard work, and Ron managed her.

David. I believe I gave him the tapes after a while, whenever there was a new release. Following up on the thing that Luis said, when he stopped answering the phone, I was the one who started answering the phone.

John. Actually, I answered the phone for you.

David. I apologize. [laughter] It turns out, there's really only five questions that anybody ever asks, so after awhile, if you called, and you happened to get me, you could start asking your question, and I'd be able to answer it before you were done. And, not only that, after awhile it got so I could say, "And, by the way, the next question you're going to ask ...." So people thought I was brilliant from this, but it turns out it's all fake.

Moderator. There's a question out there.

Didier Verna. Something totally different. The first part of this question is probably for Don, and the second part for everyone. I would like to know why TeX was designed as a macro expansion system, and the second part of the question is for all of you: How do you regard that design decision thirty years later?

Don. The way TeX was designed was the following. I thought I would have a language for myself and my secretary, and I sat down one night and I wrote out — I chose about seven pages of *The Art of Computer Programming* that had different kinds of features on them, and I said, how, if I were entering this into a computer, how would I like to do it? And I wrote that down; it's all there in the book *Digital Typography*, the memo that I stayed up late one night writing it out. And then I figured out, I changed it a little bit to something I thought I could implement, and gave it to Michael and Frank while I went to China. But the design, it was natural to have macros rather than procedure calls, the way I looked at things, because of the way I could conceive of writing this program.

So, the second question is, is it worthwhile? Well, you'll have to wait for my next talk. But I don't know what the other people on the panel [think].

Moderator. Does anybody else have a comment on that?

David. Yeah. One of the things to keep in mind, people going, oh, my gosh, how can you possibly fit it in 640 K back in the PC days? Well, it's important to realize that the big DEC-10 that it was developed on was a 36-bit-word machine — let's call that 4 bytes, more or less — and you only got 2^{18} of those, so that's only a megabyte. And the whole thing, even in the big version, fit in a megabyte, and, boy, when you look at a lot of the decisions, at least from my perspective, it was driven by that. It's amazing that you can do that much in that little memory.

Don. But when I got to METAFONT, it was macros gone berserk, because I had object-oriented macros in there.

Moderator. Another question from the audience?

Hartmut Henkel. This is for Don Knuth. Was it backslash from the beginning? Why, actually, is it backslash introducing any TeX command?

Moderator. The question is, was it always the backslash that introduced TeX commands?

Don. You can see exactly what it always was just by reading that chapter — I guess it's *two* chapters — in the book *Digital Typography*. It answers every question about what was there in the beginning.

Actually, I think, in my very first draft I did *not* have reserved words, because I was thinking of nroff. They didn't have any backslashes, so then I wouldn't be able to use certain words. But that didn't last very long.

Moderator. Question over here.

Unknown. This is partly for Don, but partly for anybody else. When did you first become aware of the PostScript language or its predecessor JAM and to what extent was there any influence of that design to TeX, or perhaps backwards?

Moderator. Michael has an answer, maybe?

Michael. After I graduated from Stanford I started at Xerox PARC in the lab where Chuck Geschke and John Warnock were, and at the time JAM was in use. By that time, they had already translated TeX78 into Mesa, so it could run on Altos and the other machines in use there. So I really don't know how much they influenced each other, but I think the TeX stuff probably predated a lot of the ...

John. I did learn JAM one summer very early in my graduate career, but it was just part of my education.

Don. I didn't look at PostScript very much myself, but I *did* visit PARC rather often, and I saw, well I remember one of the first times I went there, going by a room, somebody sitting by a terminal, and there was a big letter "B" he was measuring. So when I took my sabbatical year, that first year in 1978, I asked Xerox PARC if I could work there, to do my font work. And I was going to measure all the letters in my book, and fit splines and everything, and they said, well, that would be fine, but then all of your fonts belong to Xerox. So I went back to the Stanford AI lab and decided to do it myself. So there was a lot of work going on at PARC. And Warnock actually brought his stuff from Utah before, which I only learned later. But then the other main influences — afterwards, in the '80s I'm meeting the leaders of the font industry. Mike Parker comes from Mergenthaler, and says, boy, there's some guys over ... that have this PostScript language that renders fonts in an incredibly fast way from outlines, and things like this. And he was all excited about it. And they had new ways of tuning the fonts to the raster dynamically; hints, they call it now. And so that's when I first learned, myself, about that kind of work, the PostScript.

Moderator. I have a question. When in his interview, John Hobby says he worked with Don on METAFONT, that he primarily worked on the algorithms, and Don did the coding himself, I'm wondering with Frank and Michael, with hyphenation and paragraphing, was it the same situation there, or did any of the rest of you actually touch the TeX code?

Don. So, let me say that they were watching me, over my shoulder. [laughter] Especially David. But if you look at the error log you'll see, for example, like I think there will be several entries that say "HWT", and that's Howard Trickey, and so on. And the error log is also printed in *Digital Typography*. So the idea was, really, I was the filter and the final end. And we had this group that would meet once a week, and we would discuss over lunch, maybe two or three hours, and we would toss around whatever the topic of the week was — everybody was participating and looking at these decisions during that time, and then I would have to go back. Usually I would have a new chapter of the manual with me that they would look at, and say "Well, why did you do that?" Or they would shoot ideas, "Let's put this feature in." But then I didn't want the project to diverge, so I always decided basically, okay, we're going to make sure this is a little bit unified by being the only person who wrote the code. And I guess it's also because I guess I'm a little afraid of using something that I don't really understand all the way through.

Moderator. Anybody else want to comment on the collaboration with Don?

David. Let me just follow up quick with that. You know for awhile when I was answering the phone, I was also kind of the gatekeeper — people thought they found a bug, which frequently they hadn't, but when they reported it, I'd go, okay, I'd check it out, and see that it seemed like it didn't work, and I'd go into the code — it didn't happen that often — but I'd try and come up and find the exact line that was the problem and come up with a suggested solution, you know, in real code and test it out, and then I'd send it along to Professor Knuth, and not once did a corrected version come back that matched what I had suggested. [laughter] Never happened. It's all his code.

Moderator. Luis?

Luis. I think I have a comment in the opposite direction, which is the influence of what was happening, what Don's area of work and expertise meant to all of us who were working in there. I want to counter that with my experience in industry later on.

One of the things that you see in the development of TeX and METAFONT is the concurrent solution of a large body of problems that were not solved in a particularly efficient way, or were solved in different places. And many algorithms were redone, or adopted or whatever, but there was always this ... this group of people is the group of people who had learned about algorithms and about how to do them efficiently, and to solve *complex* algorithms, not trivial [ones], and solve them in an efficient way, not just take the trivial answer and be just happy with it. One of the things I've seen repeatedly in my experience in industry is that many, many engineers just decide, "Oh, it works. Bye." If you try TeX or METAFONT that way, it will not work, because maybe some things will be solved, but the entirety will not work.

The other thing is the issue about efficiency and optimality of things. My own little experience on this was, at the beginning we needed to interface things. And I remember a conversation with the engineers at Canon, who had brought to Stanford a printer, an OEM printer with no controller, and they said, "We did the printer, but the controller is very difficult." So we said, "We'll do the controller. Can you give us the printer?" We had a discussion about it. And what we did in there was essentially to think about how to use things that we had, like a little microprocessor, and built a few things, and we got a controller in there. The industry was not willing to go that way. They would say, "Oh, it's a complicated thing, we need a lot of memory, a lot of power, ..." Well, what we had at that point were the people around who just figured out how to solve it with the things that they had there. And I guess that that's what made it possible.

Moderator. Karl?

Karl Berry. I guess this is a question for everybody except Don, per his request. Given all the comments about literate programming, both by Don and now by Oren, I wonder if *any* of you have seen literate programming after your TeX life? [silence] I was afraid of that. [laughter]

Moderator. Has anybody seen literate programming in their life after Stanford, after TeX?

Karl. Used by, done by somebody else?

Tom. I don't claim to be capable of writing very well, but on a number of our projects I *have* used literate programming techniques for a delivered product. In my modern life, I don't. For fun, I do, but not for anything I do commercially, or anything like that. But for a number of projects I delivered both during and after grad school, I based on literate programming ideas. I didn't necessarily use CWEB. For instance, one project I did — it was a SCHEME-language program — I did use CWEB for that. But certainly the concept of the linear exposition of the ideas, small sections, and focusing the effort on allowing readers to understand the program was the goal I was shooting for, as opposed to just letting the compiler accept it and letting it pass all the unit tests and stuff like that.

Moderator. Luis?

Luis. I think if you just take a sample of what you have out there, you would say that probably documentation is never done within two or three years of release of a project. So essentially, there is no priority anywhere in the workplace to anything close to this idea of presenting the totality, of the concept, the implementation details, and the actual implementation in one place. You'll be fired if you try to do it.

Moderator. There's a question back there?

Oh, wait a second, Don wanted to say something. Hold that question.

Don. About 20 years ago when Bill Gates visited Stanford I gave him a copy of the book *Literate Programming*. I just wonder, Frank, if anybody in Redmond ever saw it.

Frank. Twenty years ago I think I had already left Microsoft. My office was right next to Bill Gates's for about a year. But I hardly ever saw him except he would walk by in the morning and he would walk out in the evening. So I'm sorry. We used at Microsoft, when I was there, we had all our own development system which was kind of inspired by another Stanford graduate, Charles Simonyi. He had something called Hungarian. I don't now how many of you are familiar with that. We used those conventions at Microsoft in the early days. Now the organization is so huge I don't really know.

Moderator. Let's go back to the question there....

Idris Hamid. This is a question for Donald Knuth, and it relates to, I guess, what we might loosely call the mysticism or spirituality of TEX, what I would loosely call the mysticism or the spirituality of the topic. One of the classes that I teach in the philosophy department is religions of the West. And in the Christianity segment, I actually used *Bible Texts Illuminated*. We don't have time to read the whole Bible, and of course that covers a number of religions, and even if it only covered Christianity we wouldn't have enough time to cover the entire Bible. But taking your own approach, this is one of the texts I used *in* that class. And, before I ask the question, I want to make one more brief, contextual comment about this. In my own work, which involves the study of Arabic manuscripts, a lot of which relate to spiritual literature, one of the things that I've encountered is the need to begin to deal philosophically with the æsthetics, for example, of the Arabic script. And then I start coming across certain sayings in my own tradition, as a Muslim, for example, a beautiful script makes the reality that it represents become more obvious. Or, that God is beautiful, and he loves beauty, loves to see his creation create beauty. So my question for you is, when you reflect on your years of work developing TEX, and when you look at your interaction with the written word, what spiritual or æsthetic reflections or wisdoms would you like to share at this juncture, given these years of digression that turned into such a beautiful product; what kind of spiritual or æsthetic reflections or philosophical reflections would you like to share with us?

Don. So, in the first place, it sounds like we're on the same page with respect to our feelings about the primacy of having beautiful things. The second thought that came to mind quickly was, when TEX itself became a reality was the moment that it had a name, and that also goes into the religious concepts, of naming something. When Duane Bibby came along, it actually got more of some kind of a soul; I don't know. But anyway, the project from the beginning was definitely driven by the idea that I wanted to have the things that I myself was writing would be something that people would enjoy looking at — not just reading, but also somehow the idea that I liked it enough to also present it well. I would dot the 'i's and cross the 't's and care about ligatures and things like that, instead of just getting [inaudible]. So that's not shared by everybody, and all of these questions are very personal; so also, I think that the Muses would agree with this kind of opinion.

Moderator. Bart?

Bart Childs. This is for everybody *except* Don. When Don was about to release, started in to do 3.0, several of us got an e-mail, "are there any features that you think should be added to 3.0?" I sent a list off, and later I thought, well, Don, mine didn't make it, but did any of you say, "here are features that you ought to be putting in TEX" that didn't get there, and, if so, what were they?

John?. I remember one that made it there, which was the one-character font name argument. I remember the day we all yelled about that back and forth, and it was nice to get multiple character font names.

Don. No, that was before 1.0. The original TEX78, there was `\font a`, `\font b`, `\font c`; there wasn't room for storing more than 32 fonts, so why should we allow multiple [character font names]?

Joe. I was trying to remember, what year was TEX 3.0? 1990. Yes, and the big thing was 256-character fonts and features like that. I think most of us were gone from Stanford by then. Tom — maybe you were still around.

Don. What ideas did you guys propose at our weekly meetings that I ignored? [laughter]

Joe. I think I wanted to make control sequence names *not* be case sensitive, and you insisted that they be case sensitive. [audience: why?] Just a matter of personal preference. [audience: I agree with Don! laughter] I guess I was just a Fortran programmer for too long.

Moderator. I have kind of a follow-on question here. The question that was asked was kind of a legacy question. I have a slightly more down-to-earth legacy question — I'll direct it to Tom, but I think it's probably true for all the rest of you in one way or another. Tom, at one time you developed `WEB2C`, and you developed `DVIPS`, and then somehow you got somebody else to take it over, or somebody else took it away from you. How do you feel about the separation, and the continuing life [of your project]?

Tom. Oh, boy. Appreciative, I guess, is the word. And truly so! I mean, not actively using that much, actively supporting a bunch of people — it's a different world when you get out of it. And it was hard for me to deal with some of the problems people were having, because I wasn't even running anything like that. And there were a number of issues with `DVIPS`. Part of it was that there was a certain effort to make it be GNU, and at the same time I really wanted to keep it free of the copyleft. And it turned out to be what I ended up doing. But I really tried to keep a separation there for awhile, and it turned out to be a bad idea .... I'm just really grateful that people took up the leadership role and made it all happen and kept it all working 'cause I wasn't doing it.

John. I could add a comment to that, in that I clearly also gave up a leadership role. In my case, clearly I was eager to give it up simply because I didn't have time to adequately maintain the program. But anyway, I think we're all glad to give up the leadership role.

David. Oren needs to comment on that. [laughter]

Oren. Yeah. My comment was that I never let that stop me. [laughter]

Moderator. You have a new release of BibTeX coming out?

Oren. BibTeX 1.0 is going to come out any decade now. [laughter]

Moderator. Another question from the audience, please? There's one way back there. Frank — is it Frank?

Frank Quinn. Leslie Lamport has been a big influence on all of these developments. Could you perhaps comment on how you saw his motivation, and what sort of interaction he had with the group?

Oren. Well, I'll comment a little bit, since my real first .... Other people here really know more of the TeX internals, everybody else does than I do, and so my first contact with a lot of the TeX stuff was through Leslie, and as I said before, he needed somebody to do a bibliography processor for LaTeX. And I know his thinking was, he kind of liked the path that Brian Reid took with Scribe; it was kind of a very simple interface, you could describe things fairly succinctly. Somebody who's an English major could easily use it and get nice output. And so, I think that was the first contact I had was with him. I don't know how much he was involved with the TeX project before that. I think people at — where was he then? I think he was at SRI — he had written some macros that people liked there, and I think they encouraged him to do something with it. I'm not sure where his other influences were, but certainly, once he got going, I think people, obviously they're fairly happy with it.

Moderator. Anybody else?

Don. He's a very independent spirit, like I am. He does a lot of work on his own. Every once in a while he would run into something he couldn't do, and so then I would have to put in another feature, while kicking and screaming. But it was totally independent work from Stanford.

Moderator. Right here — front row.

Robert Cristel. We know how the problem of ... *The Art of Computer Programming* [printing] caused TeX to be more [æsthetic]. So my question really is, apart from making *The Art of Computer Programming* more beautiful, how, in other ways, did TeX affect *The Art of Computer Programming*?

Don. Well, it set it back about fifteen years. [laughter] On the other hand, I'm writing a little bit faster now, so maybe it'll save twenty years if we amortize the whole thing.

I thought you were going to ask about other things *besides The Art of Computer Programming*.

Robert. I mean the stuff that's inside, not necessarily the other.

Don. To my great surprise, right from the get-go for example, Barbara Beeton came with a few other people during the summer of 1978,[1] and she showed me all the kind of things that she wanted to do with *Math. Reviews*, and then also the AMS

[1] Actually 1980. — bb

was having trouble typesetting their journals and things like this, so I started looking at applications of other users. And so then this meant that TEX had to grow in lots of ways. But it was sort of going from one user to ten users, and then from ten users to a hundred users, from a hundred users to a thousand users, ... each time the language had to change in some way. And I *think* the fact that all these things had to be filtered, had come down to me, helped to keep the thing from diverging, although of course for complicated systems ... it would have been a lot worse if we hadn't done it that way.

Moderator. Frank?

Frank Mittelbach. A question for Michael, I guess. I consider the paragraph algorithm as one of the very central algorithms within what actually has been achieved with TEX. It grew in time. I know your thesis, and I've seen other papers around it. My question is, as far as anybody can remember, have there been things you were sort of experimenting with that you *would* like to have in addition to that kind of algorithm, like in parameterization or something that you either never got around to doing, or found too difficult, or got shut down for other reasons? Is this the ultimate thing you wanted to have, as a group, in terms of being able to do this kind of thing, or is there some stuff that back then was not possible for some reason, but conceptually was on the horizon?

Michael. A lot of the features that were built into the line-breaking algorithm itself that's in TEX, I think Knuth ended up putting in there based on experience. As far as the actual coding of the algorithm, he did that. The origin of the problem was actually in the first graduate seminar programming class, where there was a problem for doing this; I guess he was thinking ahead a little bit to a sabbatical year at that time. But that was for breaking up — I think the problem was musical composition, if I remember right, but it's a very similar thing.

I guess as far as something that it would be nice if more of it were used in practice is the stuff that's in my thesis about arranging figures, moving figures from page to page, which at the time, it was certainly way too expensive to actually consider using in a real typesetting program. Maybe today it's not.

Don. So homework problem, Frank, go look at Michael's paper. He wrote a short version of our joint paper, which was published in another book about typography at the time. He generalized what we had and had an idea of a kerf (spelled 'k e r f'), and I don't remember what it was, except that it was good.

Moderator. A kerf is something to do with a saw, in real life.

Don. Okay, but anyway, it was in his paper, and I can't remember it today either, but anyway, I think it's worth resurrecting.

I wanted to mention something before I forget it. Although I wrote the main code that people saw, for TEX and METAFONT, there were also drivers and many other programs that had to be written, like TFTOPL and all kind of other what we call utility things. And Tom Rokicki did the things associated with PK fonts, for example. But David remarked briefly about having to take all the TEX code and convert it to C; well, he wrote a long WEB program that did this, and then I modified it slightly so that it would make profiles of the TEX system, so that it could instrument the whole program and find out how many times every instruction was done. And then David worked out a very clever thing that would work on our computer, and it — I think Joe Weening worked on this too — there would be daemons that would keep track of these statistics, and so over a whole year's time, every time anybody ran TEX at Stanford, these statistics were kept, and the counts were accumulated, and carefully saved, with machines crashing every day, but still pretty good stuff altogether. And then, using David's profiling program, I could make a pretty-printed version which would associate with every line of TEX exactly how many times people had used that line. And it was really important, for example, how many times did each error message get issued during the year. And we could figure out what the bottlenecks were. So anyway, to make a long story short, there's lots of other programs that were written at that time that were necessary for the development, that didn't go out to the world.

Moderator. Is there another comment down there?

Joe. I had completely forgotten all about that; it sounds familiar. I don't remember the details.

Don. The DEC-20 had memory that was divided into two parts, and there was one part that was sort of always there for the system libraries and things like this, and that's where all these statistics were living. There might be ten people using TEX, but only one copy of TEX is running somehow on the machine. And that took a lot of system wizardry, and I have no idea how they did it.

David. You wanna know? [laughter] So this was actually on the DEC-10, which had the WAITS operating system, which was custom-built at Stanford. That 2^{18} address space, it was half code, half data.

And the code segments were all shared, so if many people on this time-sharing system were running TEX at the same time, or any program — you know, the editor, the system editor — there'd only be in physical memory one copy of the code, but everybody had their own separate data, so I was editing my file, and you're editing your file. Well, the trick was, we wanted to count every execution of every basic block of TEX — every line of code, more or less. So the compiler used on that machine was this terrible thing from Hamburg — half the comments were in German, which I suppose was okay, but the compiler was not very good — but I managed to modify it so that — what! a lot of modified compilers here [laughter] — so that it would spit out little increment, atomic increment instructions every time it entered a basic block. The trick was that the place, the memory locations it would increment were in what should have been the read-only code segment that was shared among all the users. Yeah, right. People who know hardware are raising their eyebrows. [laughter] So that was fun, and every day or so, it would save itself out to disk, and there was special code that could retrieve this stuff out of the code segment, and it would actually increment; if the program counter was dot, it would increment dot plus two and then jump over it. So there was enough room for all the data. So that was another great piece of fun; this was back in the day ... the point is that the resources were really tight, and it was hard to get stuff in, and you had to do all sorts of hackery.

Moderator. I think we may be getting a little overly sentimental now. [laughter]

Well, over the last couple of days, trying to help this panel go more smoothly, we invited questions to be submitted in advance. And I have two, so I'd better ask them, or else I'll be very rude. And I'm going to combine them, in a sense.

The one question is, in your legacy, where does your work with TEX stack up? Now, that's kind of a TEX-centric question, but that's the question. And the other question is, because people are curious, what are you doing now? I think those two questions go well together — you've done a lot of things since then. Where does TEX fit into your life adventure? If anybody would like to answer that.

John. You say you'd like us all to answer?

Moderator. No, no, anybody who wants to answer.... We don't have to have everybody answer.

John. Well, okay, I'll give you the answer first of all. Certainly, METAPOST is one of the most visible things I've done. I'd sort of not like it to be the highlight of my scientific career, but all I can say is, sure, it was a great experience and I'm happy to have worked on it, and I'm not surprised that it can somehow overshadow a little bit of the other stuff.

Tom. Well, let's see. Coming and working on the TEX project for me was absolutely changing, because TeXas A&M ... I was a bit of a cowboy, I wasn't really — I wasn't CS, I was EE. I was always pushing electronics, not bits. But I enjoyed programming a lot, and I learned a lot myself, and all that, but coming to Stanford and being with some of these people really taught me a lot about how to program correctly, and the importance of literate programming, and that sort of thing. So, absolutely critical. As far as what I'm doing now — I'm a web programmer nowadays. I've got a little startup down in Santa Clara. We write huge enterprise applications for Fortune 500 companies.

As far as legacy, you know, I could not ask for anything better than to be associated with this group of people and this project. So I have absolutely no problem with this being "the big thing".

Moderator. Anybody else have a comment? Not required. Luis?

Luis. It essentially sent me in a direction in life that I had never expected to be. I was a student of Don's, and I was going to do something academically "tainted". [laughter] And I ended up in industry. And essentially, the one thing I think I have the most ... what I recall my time at Stanford as part of the project is the quality of what was being done. The point I was never able to achieve in industry, because I want into the printing industry, who were creating laser printers and doing some architecture for that. You can never do it; you can never go back to that level of excellence and I have to thank all the people here and of course Don, for that.

Howard. I was going to say something. I'm think that I'm very proud to be associated with this because there's the comparison between things like Scribe and things like TEX when the kind of utilitarian thing that was easy to use and then TEX appealed to me because of the beauty part of it, which I mentioned earlier. And I'm proud to have been part of something that brought a lot of beauty to texts for many, many years, that everybody has been producing. I think the world would have been a much uglier place if we hadn't done what we had done, but especially, of course, Don, to do this. I like the fact that many, many people have run a piece of code that I have written because of this project; so that's the legacy part there.

Now I work on Google maps, and wrote business ranking code, which maybe is starting to exceed the number of instructions executed.

Joe. Yeah, I guess just thinking about that, what I enjoyed the most — it's always been sort of part of what I like to do — is to see something that I like and make it better by adding features or doing things that I would like to see in my project. And I think that all of us at this table have done that. So we were each able, in our own way, to contribute to the TEX project, and just thinking of it in the larger sense, this is what Don has done for this whole community, is to take what he needed and what he wanted to see for his books and then make a really big software project out of that, that had a lasting [effect]. So I think we're all proud to be part of it in that sense.

Moderator. I think we're almost at time out, and I'd like to make an observation. I joined the TEX community, I don't know, ten years ago, roughly, something like that. And at the time, it seemed to me that there was some depression I sensed. You know, things weren't changing; things were getting .... Of course, a lot of development was going on all the time, but today, at this meeting, I have a sense that it's a *very* active development community. People are excited about things, and I'm just so impressed that something that started thirty-two years ago was done in a way that enabled that group of people to pass it on to another set of people, that group of people to pass it on to yet *another* set of people — I'm not sure what generation we're on now. Surely there's some people who have been involved more or less the whole time. And today it remains a vibrant community trying to achieve the beauty that this group of people set out to achieve.

With that, I think we should call an end to this session. The panelists — I'd like to thank you all for both your participation today and for everything you've done for us, that led us to be here today.

[applause]

From left: Luis Trabb-Pardo, Michael Plass, Tom Rokicki, John Hobby, David Fuchs, Don Knuth, Howard Trickey, Oren Patashnik, Joe Weening, Frank Liang.

Thoughts on TUG 2010

Barbara Beeton

Well, another annual meeting has come and gone. This one has been very special. For TUG's 2^5 anniversary, almost all the original members of the Stanford TeX Project were located and came to the meeting for at least the final day, when they participated in a panel discussion (see the transcription of the panel discussion earlier in this issue) ranging over topics from recollections of the project itself to what they thought has been the lasting value of TeX. I was privileged to work with quite a few of these individuals, principally David Fuchs and Don Knuth himself, to learn TeX from the source. I doubt that such a group will gather again.

What was the environment in 1980?

- Most computers had limited memory, no more that 1 Mb. The most popular Unix platform (the DEC PDP-11) was too small to install TeX. Personal computers were still several years in the future.
- TeX78 was written in SAIL; this ran only on the DECSystem-10 or 20.
- There were no desktop laser printers. The highest resolution of the existing raster printers (at least the ones available to ordinary mortals) was 200 dpi.
- Although the ARPAnet existed, this was available only at major research universities. (I was granted an account at Stanford, `bb@sail`, but could access it only through a long-distance telephone hookup and `telnet`.) TeX was distributed on reels of ½-inch magnetic tape, ordered from Maria Code (yes, there really was such a person).

Progress came relatively quickly in some areas, more slowly in others:

- The personal computer revolution arrived in the mid-1980s. Although the first ones had limited memory, requiring that any attempt to port TeX would have to use overlays and other tricks even to fit, Moore's law rapidly took over. Now your cell phone has more memory than the supercomputers of 1980, and TeX can actually be installed on at least the iPhone, as demonstrated by Kaveh Bazargan at TUG 2009. And speed of compilation is no longer an issue; where it used to take 5 seconds to compile a page, now 500 pages or more can be compiled in less than a second.
- By 1980 it was already realized that, if TeX was to spread beyond the limited bounds of the DEC-10/20 community, it would have to be recast in another language. Pascal was the base decided by Knuth for TeX82, followed by mostly-automatic translations to C, which is now the norm for most TeX implementations.
- In 1984, the Apple LaserWriter arrived, along with PostScript. The days of raster fonts were numbered. While commercial printers (the companies that print books and journals, not the hardware) have usually required Type 1 fonts, the standoff between Type 1 and TrueType fonts has been settled with the adoption of OpenType. As for resolution, personal printers at 1200 dpi or higher are no longer uncommon. METAFONT can still be used to develop glyphs and fonts, but the superiority of outline fonts in applications such as browsers means that the "final" image is best not limited to rasters, and METAFONT's offspring, METAPOST, is now used for much TeX font development.
- The advent of the World Wide Web and powerful browsers brings convenience as well as the ability to share information — including downloading an entire TeX Live distribution — in real time. Universal connectivity via the Internet brings bad things (e.g., spam) as well as good, but it's not likely to go away soon. New users of TeX and friends wouldn't recognize the old world. In fact, TeX is now used "under the covers" in some places for producing ad hoc commercial documents like train schedules and phone bills, totally without the knowledge of the end user. But it is still the language of choice for most mathematicians and physicists, and likely to remain so until a user-friendly and semantically meaningful front end for XML/MathML appears.

My guess is that there will still be solid uses for TeX when its 2^6th birthday rolls around.

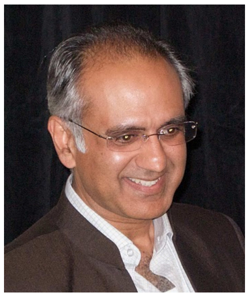
The other signal event at the conference was Don's introduction to the next generation of TeX — (see his paper, also in this issue). It was earth-shaking indeed! Although it perhaps owes more to a date of April 1[1] than to its actual date of delivery, we shouldn't reject its "design objectives" out of hand.

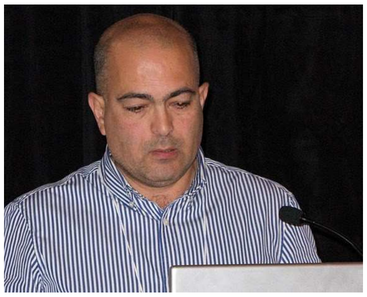
Even if you couldn't come to San Francisco, you can enjoy much of the excitement via video: `river-valley.tv/conferences/tug-2010`. Once again, thanks to Kaveh Bazargan for making this possible.

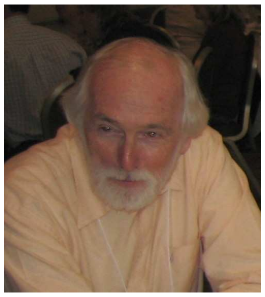
◇ Barbara Beeton
`bnb (at) ams dot org`

[1] We've been fooled before: *TUGboat* 19(2):95–96 (1998), `tug.org/TUGboat/Articles/tb19-2/tb59hoax.pdf`

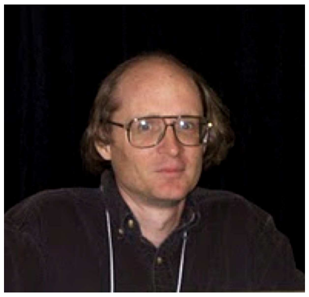
Leila Akhmadeeva


Pavneet Arora


Kaveh Bazargan

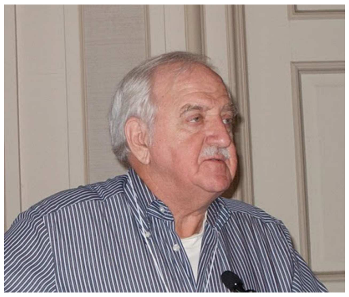

Nelson Beebe

Barbara Beeton


Karl Berry

Mathieu Bourgeois

Bill Cheswick


Bart Childs

Kaja Christiansen

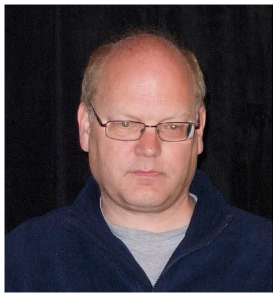
Jennifer Claudio

Michael Doob

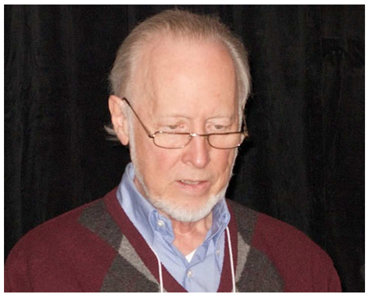
Jean-luc Doumont

Walter Gander

Steve Grathwohl


Hans Hagen

Idris Hamid

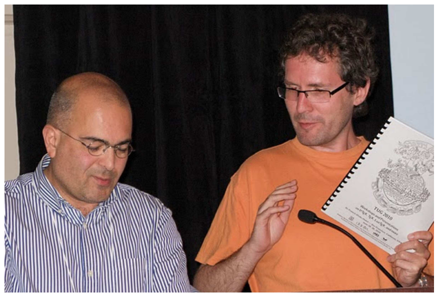

William Hammond

Jim Hefferon

Stephen Hicks

John Hobby

Morten Høgholm


Kaveh Bazargan, Klaus Höppner

Mirko Janc

Jonathan Kew

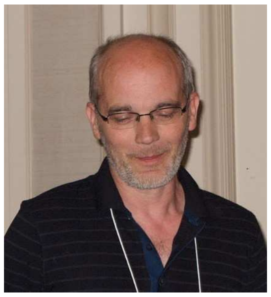
Donald Knuth

Martha Kummerer

Robin Laakso

Manfred Lotz

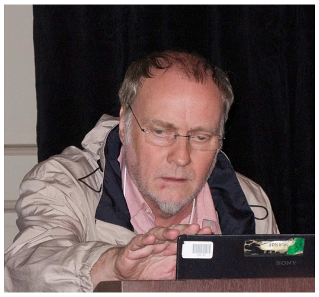
Filip Machi


Frank Mittelbach

Ross Moore

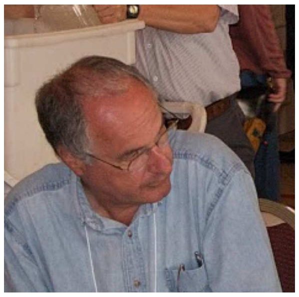
Will Robertson

Tom Rokicki

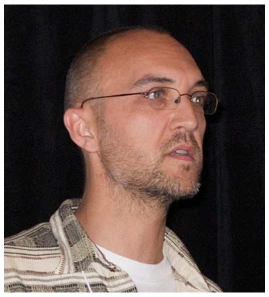

Chris Rowley

David Ruddy

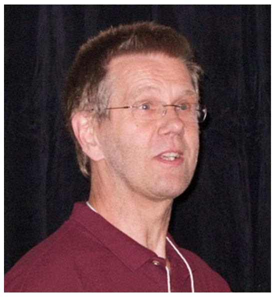
Volker Schaa

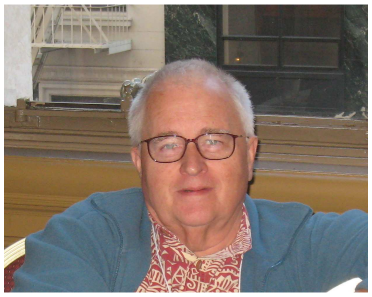

Herbert Schulz

Heidi Sestrich


Didier Verna

Boris Veytsman


Herbert Voß


David Walden

Alan Wetmore

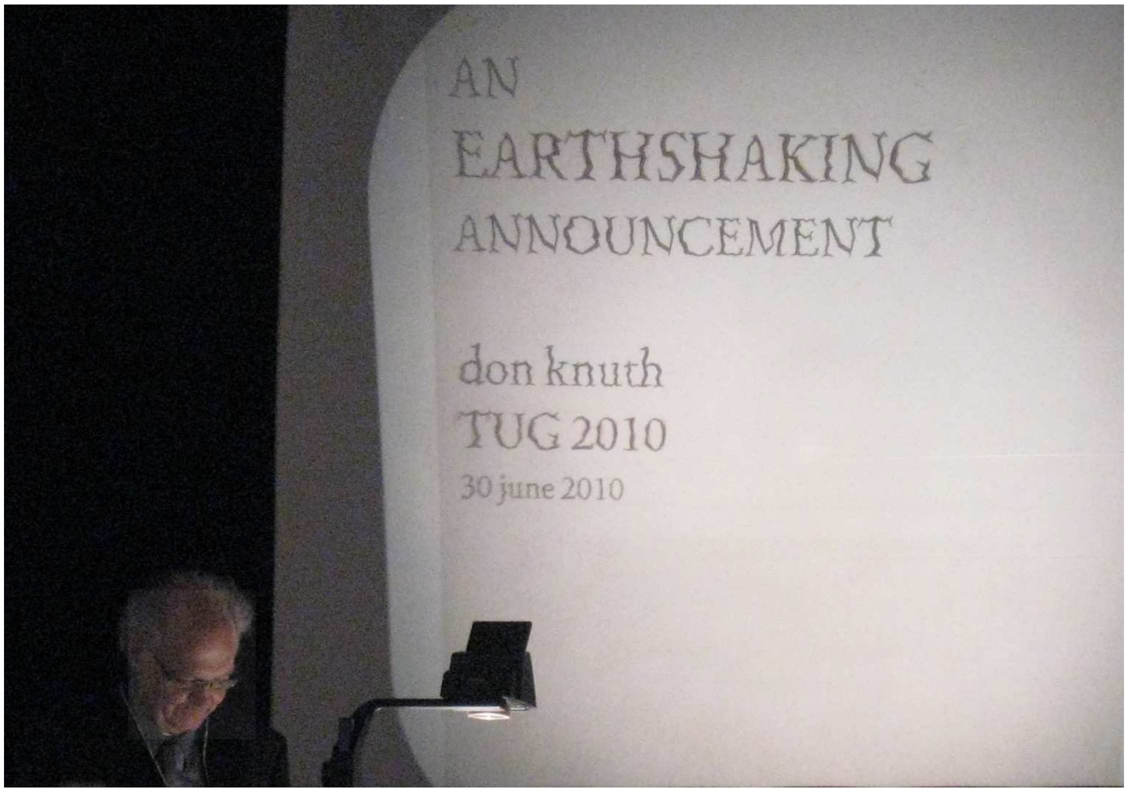
Uwe Ziegenhagen

Herbert Schulz, Morten Høgholm, John Bowman, Jim Hefferon

Bart Childs, Tom Rokicki, Nelson Beebe

Frank Mittelbach, Barbara Beeton, Tom Rokicki, Didier Verna, Don Knuth

Herbert Schulz, Dick Koch, Wendy McKay


AN
EARTHSHAKING
ANNOUNCEMENT

don knuth
TUG 2010
30 june 2010

An Earthshaking Announcement from Don Knuth

CTAN packages get keywords

Jim Hefferon

As part of a larger effort, I am adding to the web pages on the TUG CTAN node the ability to search by keyword and by hierarchical characterization.

1 Audience

The great majority of people who come to CTAN use TeX and LaTeX only as an adjunct to their main work. This kind of visitor comes to us with a specific need, looking to swoop into our holdings, grab just what helps, and then go back to what they were doing.

The good news for this person is that we offer a tremendous number of things — TeX and CTAN have been around for a long time so probably their problem has been solved and probably we hold that solution. But the bad news is that we offer a tremendous number of things, so leading visitors to exactly the materials that will help them is a challenge.

The current text search on `http://tug.ctan.org/search.html` is a good resource, as is the file name search on that page. But online users are also accustomed to two other query forms that we do not now offer, both of which suit swoopers. The first is a list of keywords, while the second is to have choices in a tree, as with the *Open Directory* (`http://www.dmoz.org`).

I have put up experimental versions of these two. Please have a look, but note that the web addresses are temporary. At the present, the keyword page is at `http://az.ctan.org/keyword` and `http://az.ctan.org/characterization` is the characterizations page. If this system becomes the default then those addresses will change to using `tug.ctan.org`.

For instance, from the keyword page a user can choose "tables" and get a list of links to packages that work with tables. It now finds 76. The user can narrow the results by selecting up to four keywords at a time and getting the intersection of the sets of associated packages.

Characterizations also work in the natural way. One example: a user picks the `primary` tree and then picks `Document types > Books > Publisher styles`, yielding links to our packages that provide publishers' book classes.

The top characterization page links to two trees besides `primary`. Many packages do more than one thing so in addition to having the main purpose of the package entered on the `primary` tree, other functionality is on the `secondary` tree (a package can be on at most one branch of the `primary` tree but can be on many `secondary` branches). The other tree is based on Jürgen Fenn's topical index of the *Catalogue*.

Figure 1 shows a screenshot of a package information page including keyword and characterization information, along with the standard *Catalogue* data.

2 Architecture

The keywords and characterization features are one aspect of a larger overhaul of the `tug.ctan.org` web pages. This project is named AZ, after my ham radio call sign KE1AZ, and also for the amusement of calling it AZTeX.

Behind the web pages is a database, where most of the data is provided by the *Catalogue*. Keyword and characterization data is in addition to that from the *Catalogue*.

I made up the list of keywords based on questions to `comp.text.tex` and `texhax`, and also based on the content of the packages that we hold — if I ran across a number of packages to do something then I guessed that there was a demand for the functionality.

I also made up the `primary` and `secondary` trees, which have the same entries. The `Fenn` tree is scraped from Jürgen's HTML page.

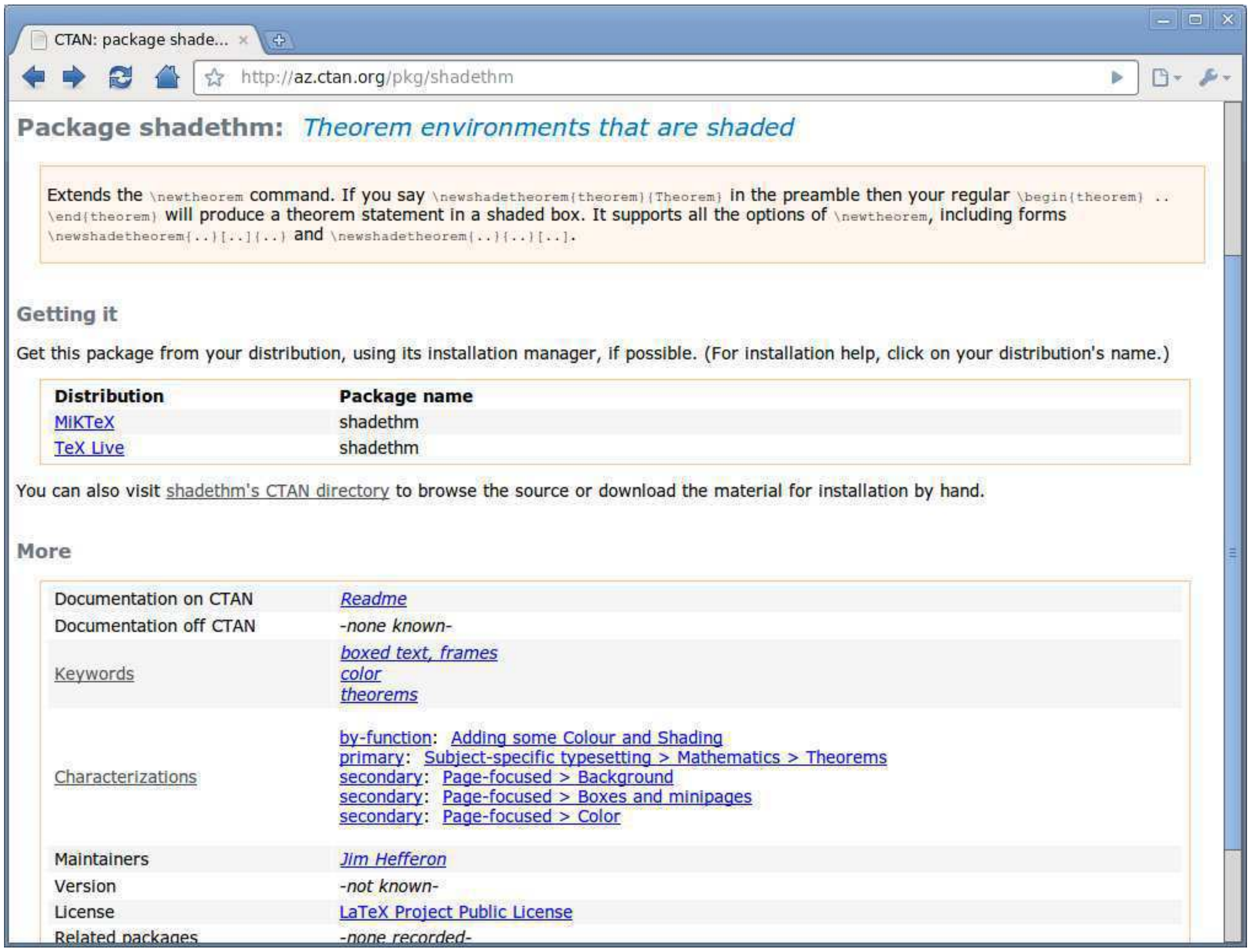
The selection of list entries is influenced by the choice of audience. For instance, a person interested in TeX development will find that all of the packages aimed at developers have gone into just one keyword and only a few categories. Most of the keywords and categories are for swoopers.

I also tried, in selecting the list entries, to avoid entries with too many associated packages, or too few. This set of keywords and characterizations is actually a second draft. I did a dry run with an earlier set last year and made adjustments to get to the current list. (Some of the entries seem to me to be time-sensitive, but not too many. I hope these sets last for a while.)

3 Accomplishment

CTAN holds about 4000 packages. I entered the keyword and categorization data using a web form, and while I sometimes needed to examine a package closely, more often I used only the *Catalogue* description. Still, even though a typical package was done quickly, the sheer number of packages means that the entire job took a long time. (I entered these in the evening while watching the Celtics play basketball and it took me all season, so the job size is several months of a couple of hours, three nights a week.)

I hope the list can be maintained in part by package authors entering or editing the data via a


CTAN: package shade...

http://az.ctan.org/pkg/shadethm

Package shadethm: *Theorem environments that are shaded*

Extends the `\newtheorem` command. If you say `\newshadetheorem{theorem}{Theorem}` in the preamble then your regular `\begin{theorem} .. \end{theorem}` will produce a theorem statement in a shaded box. It supports all the options of `\newtheorem`, including forms `\newshadetheorem{..}[..]{..}` and `\newshadetheorem{..}{..}[..]`.

Getting it

Get this package from your distribution, using its installation manager, if possible. (For installation help, click on your distribution's name.)

Distribution	Package name
MiKTeX	shadethm
TeX Live	shadethm

You can also visit shadethm's CTAN directory to browse the source or download the material for installation by hand.

More

Documentation on CTAN	*Readme*
Documentation off CTAN	*-none known-*
Keywords	*boxed text, frames* *color* *theorems*
Characterizations	by-function: Adding some Colour and Shading primary: Subject-specific typesetting > Mathematics > Theorems secondary: Page-focused > Background secondary: Page-focused > Boxes and minipages secondary: Page-focused > Color
Maintainers	*Jim Hefferon*
Version	*-not known-*
License	LaTeX Project Public License
Related packages	*-none recorded-*

Figure 1: Package description including keywords and characterizations, as displayed in a browser.

web form at the time that they upload; this is another feature of the AZ system.

After looking at my keywords and characterizations you may think of another or better list. For instance, you may think that a characterization into “works with LaTeX” or “not” or “not applicable” would help. I'm open to suggestions but above I've described the amount of scutwork required so that you will know that if you want to suggest entering and maintaining it then you would be committing to a significant effort.

4 Acknowledgements

I would like to thank my CTAN colleagues, Rainer Schöpf and Robin Fairbairns; in particular, Robin has done many years of yeoman's work on the *Catalogue*. Karl Berry was also invaluable here, as in so many places.

◇ Jim Hefferon
Saint Michael's College
Colchester, Vermont USA
`ftpmaint (at) tug dot ctan dot org`

[redacted]

```
  \DeclareGraphicsRule
    {.png}{eps}{.bb}
    {`convert #1 eps:-}
\makeatletter % more complex method for
              % programs other than convert:
  \let\Saved@Gin@base\Gin@base
  \let\Gin@base\relax
  \DeclareGraphicsRule{.gif}{eps}{.bb}
    {`convert #1 \Gin@base.eps &&
     cat \Gin@base.eps}
  \let\Gin@base\Saved@Gin@base
\makeatother
  \usepackage{grfext}
  \AppendGraphicsExtensions*{.png,.gif}
\begin{document}
  \includegraphics{lion}\qquad
  \includegraphics{knuth-tex}
\end{document}
```

Again, we need .bb files for the images:

```
knuth-tex.bb  knuth-tex.gif
lion.bb       lion.png
```

And shell escapes enabled:

```
latex -shell-escape testimgfmts.tex
```

4 Using PSTricks with pdflatex

Let's turn our attention now to some of the methods for using PSTricks packages specifically with pdflatex.

4.1 pdftricks

First, the pdftricks package. With this, you demarcate the preamble that should be used for the intermediate run with the psinputs environment:

```
\documentclass{article}
\usepackage{pdftricks}
\begin{psinputs}% preamble for latex runs!
  \usepackage{pst-node}
  \usepackage{graphicx}
\end{psinputs}
```

And then the usual:

```
pdflatex -shell-escape testpdftricks
```

We can thus use PSTricks packages together with EPS images, and as usual for pdflatex also JPEG, PNG, and PDF images.

4.2 pst-pdf

With the pst-pdf package, load PSTricks packages only when not producing PDF:

```
\documentclass{article}
\usepackage{pst-pdf,ifpdf}
\ifpdf\else
  \usepackage{pst-node}
\fi
```

Then there are several steps to the processing:

1. Run latex to create a dvi file with only the extracted pspicture and postscript environments. or \includegraphics for eps images.
2. The dvi output then is converted to a PostScript file which itself is of a special format and can only be used for the next step.
3. The ps output is converted to a pdf file which has one page per extracted image.
4. If needed, run pdfcrop to tightly crop.
5. The last pdflatex run replaces the pspicture and postscript environments and eps images with created pdf images.

For example:

```
latex ptest
dvips -o ptest-pics.ps ptest.dvi
ps2pdf ptest-pics.ps ptest-pics.pdf
#pdfcrop ptest-pics.pdf
#mv ptest-pics-crop.pdf ptest-pics.pdf
pdflatex ptest
```

4.3 auto-pst-pdf

The auto-pst-pdf package automates the above process.

```
\documentclass{article}
\usepackage{auto-pst-pdf,ifpdf}
\ifpdf\else
  \usepackage{pst-node}
\fi
```

We need only *one* pdflatex run, everything is done inside of the auto-pst-pdf package.

```
pdflatex -shell-escape ptest
```

4.4 Option pdf for PSTricks

The [pdf] option to pstricks works only for latex! It loads the package auto-pst-pdf.

```
\documentclass{article}
\usepackage[pdf]{pstricks}
\ifpdf\else
  \usepackage{pst-node}
\fi
...
```

As above, we need only one pdflatex run:

```
pdflatex -shell-escape ptest
```

[redacted]

Improving margin paragraphs

Stephen Hicks

1 Introduction

A common frustration among authors using LaTeX to typeset books with significant margin material is that long margin notes often run off the bottom of the page, as shown in Figure 1. Several years ago, Andy Ruina approached me with an idea for solving this problem by completely rewriting LaTeX's margin code, at which point I developed an early version of the marginfix package, the subject of this article.

In this article, I'll start by looking in more detail at the problem and some common workarounds. After this, I'll explain the basic idea behind marginfix's solution. Finally, I'll describe a variety of options allowed by marginfix to further tweak margin note placement.

2 The problem

For a quick glance at the problem, the reader is encouraged to typeset the document in Figure 2. Doing so will yield a full-page paragraph with a margin note tied to the end of the paragraph, and this note clearly overflows past the bottom of the page.

What's happening here is that when the user calls `\marginpar`, LaTeX goes through a number of steps (Lamport, Mittelbach, and Rowley, 2004). To start, the content of the margin note (in the example, `\Large\lipsum[2]`) is saved into a pair of floating boxes (one for right-hand notes and the other for left-hand). LaTeX then inserts a specific large negative penalty into the outer vertical list, invoking TeX's `\output` routine, where the amount of text typeset so far on the page allows determining the vertical position of the `\marginpar` callout. This vertical position is compared to the position of the bottom of the previous margin note (if there is one), allowing LaTeX to shift the new note downward in an effort to prevent intersection. After this shift is computed, the box containing the margin note is shifted all the way to the left or right of the page (as appropriate), smashed (its height and depth are set to zero), and added to the outer vertical list.

This algorithm has both advantages and disadvantages. First, it's relatively straightforward and easy to understand, even if the behavior is less than desirable. Another advantage is that it automatically deals with vertical glue in the middle of a page. If a `\vskip` is stretched or shrunk and there is a `\marginpar` beneath it, the note is automatically moved along with the stretch or shrink, since its

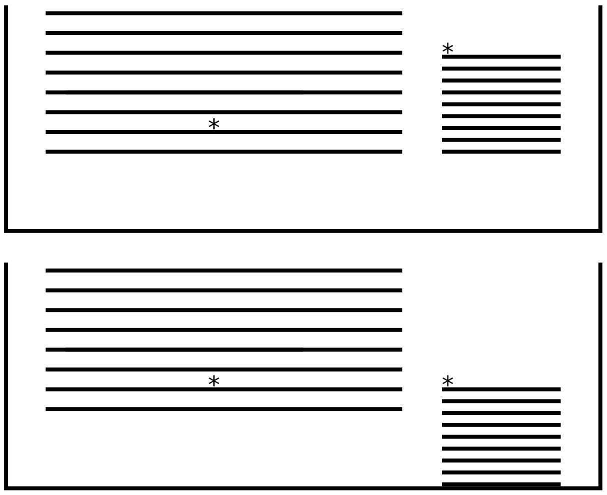


Figure 1: What the user expects (top) versus the default LaTeX output (bottom)

```
\documentclass{memoir}
\usepackage{lipsum}
\begin{document}
\sloppy\Huge\lipsum[1]
\marginpar{\Large\lipsum[2]}
\end{document}
```

Figure 2: Problems with LaTeX's built-in margin notes

box is inside the lower paragraph. There are also a number of disadvantages. First, it's still possible for margin notes to intersect, if a skip between paragraphs with nearly-intersecting notes is shrunk. More importantly, there is no provision for notes to move upward on a page or to defer to later pages. Thus, if a note starts on the bottom of a page, it has nowhere to go but through the bottom margin.

3 Common workarounds

Any author who has written a substantial amount of LaTeX has developed a bag of tricks to coerce LaTeX's behavior when it's being particularly stubborn. There are a few such tricks for dealing with problematic margins.

- The simplest workaround is inserting a negative `\vskip` at the start of the marginnote (i.e. just before `\Large` in Figure 2). In Figure 2, the proper value is something like `-30pc`. But if the margin note changes (say, to `\lipsum[3]`), or if the page breaks change, then the value must be re-tweaked.
- The memoir package (Wilson, 2010) provides a workaround in the `\sidebar` macro, a complete (and incompatible) alternative to `\marginpar`

that provides a flowing column in the margin. This sufficiently addresses the problems with overflowing margins, but it makes no attempt to localize the margin material anywhere near its callout location (see the effect by changing `\marginpar` to `\sidebar` in Figure 2).

- Occasionally authors will find margin notes appearing on the wrong side of the page. This is due to LaTeX deciding on which side a margin note should appear at a different time from when it decides the page breaks. If a note looks like it will be on the bottom of an odd page, LaTeX attaches it on the right; but if a page break is inserted before the margin note, it will be wrong. The common solution here is provided by the package mparhack (Sgouros and Ulrich, 2005), which stores the actual page on which a note appears in the `.aux` file.

4 A new approach

Since LaTeX's margins already make use of floating boxes (that is, *inserts*), it is a natural extension to allow the notes to actually float. We can state our goals simply (the parenthetical notes refer to the step in our routine that enforces each goal):

- Each margin note is close to its callout (3b)
- No notes intersect (3b) or fall off the page (3c)
- Margin notes *may* float to later pages (3a)

Thus, the basic outline of the margin routines in marginfix is as follows:

1. When `\marginpar` is called, no change from LaTeX's behavior.
2. When the `\output` routine is entered for the first time, rather than setting the note immediately, instead append its boxes and the callout's page position to a token register (`\marginlist`).
3. When the `\output` routine is called and TeX actually builds the page, insert our own code to assemble the margin column, described below.
4. Once the margin column is assembled, we attach it to either side of the main column with an `\hbox`.

The interesting work here happens in step 3. We break this up into several substeps, as shown in Figure 3:

3a. Build a list of margin notes that are guaranteed to go on the current page by taking as many notes as possible from the front of `\marginlist` such that the total height of the notes does not exceed `\textheight`.

3b. Working down from the top, insert compressible "glue" into the list of notes on this page so that stacking all the boxes and glue places each note at, or below (in the case of notes that would otherwise intersect), its callout location (note that if we performed this step alone, we would reproduce LaTeX's margin routine).

3c. Working upwards from the bottom, remove as much glue as needed so that no notes overflow into the bottom margin.

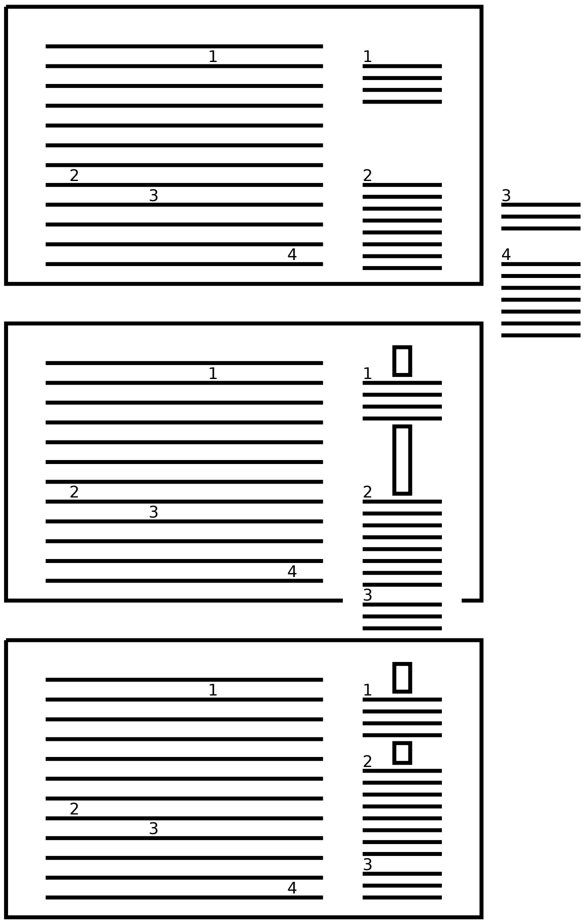


Figure 3: A new approach to building margins. We start by attaching each note to its callout position (top). We next push the notes down the margin until they don't intersect, and insert compressible glue (shown as narrow boxes) between non-abutting notes (middle). Note that the fourth note has been deferred because we only have room for 20 lines of margin material on the page. Finally, we work upwards, compressing glue as needed until all notes fit (bottom).

More concisely, we "expand down, compress up".

There are some interesting cases to consider here. If step 3a admits exactly `\textheight` of margin material then all the glue added in step 3b will be removed in 3c, resulting in a densely packed margin, comparable to `\sidebar`'s output. If there are not too many notes so that 3a doesn't defer anything, and if nothing occurs too close to the bottom of the page, so that 3c doesn't remove any glue, then we have reproduced LaTeX's output.

5 Options

The marginfix package provides a number of buttons and knobs that allow fine-tuning of the output.

- `\marginparpush` works the same as in standard LaTeX, adding a fixed incompressible space between notes.
- `\marginskip`⟨*length*⟩ appends an incompressible gap of a given *length* to the margin.
- `\mparshift`⟨*length*⟩ vertically shifts the next note by *length*.
- `\extendmargin`⟨*length*⟩ makes this page's margin longer by *length*.
- `\clearmargin` stops new material from going into this margin.
- `\blockmargin`...`\unblockmargin` makes a gap in the margin where no material can go, which is useful for extra-wide figures or equations.
- `\marginphantom`⟨*length*⟩ makes a gap, similar to `\(un)blockmargin`, except that one point is given as a vertical displacement from the other point, rather than specifying both points.

Outside of these macros, the package is simply a drop-in replacement of `\marginpar`.

6 Concerns and future work

A number of issues have recently come up in discussion with authors and developers. First, the initial version of step 2 used `\@pageht`, set by LaTeX's margin routines, to determine the callout position, but it was pointed out that this is inaccurate if any of the glue in the main column is stretched (or shrunk). Because LaTeX attaches the margin note to the main column immediately, the only effect this has in LaTeX is to allow the notes to possibly intersect if the glue shrinks enough; but since we put off note placement to the end, any stretch or shrink will cause misalignment. An initial thought is to calculate the stretchability above a note and then determine the glue set before building the margin column, but neither of these operations is possible (without doing cube roots of penalties, at least). Instead, we can use pdfTeX's `\pdflastypos` to determine (on the second pass) where the callout actually is. One benefit here is that we no longer need the premature output routines to find the vertical position.

Several features have been requested that are not yet implemented, but are not beyond the realm of possibility. LaTeX provides the ability to force a margin note to go into the opposite margin (with `\reversemarginpar`), so that both the left and right margin can have notes. This could be achieved by repeating the margin-building step twice, provided that each note clearly specifies into which margin it belongs (if notes are allowed to float between margins, the optimization seems to become more difficult). Another possible feature is allowing long notes to flow onto the following page, as memoir's `\sidebar` does. Because we can push margin notes upward, there is no reason to break a note unless the entire margin is full (much like the flowing of a full `\sidebar`). If necessary, however, a `\vsplit` would get the job done.

References

Lamport, Leslie, F. Mittelbach, and C. Rowley. "ltoutput.dtx". 2004. Available from CTAN, `macros/latex/base`.

Sgouros, Tom, and S. Ulrich. "mparhack.sty". 2005. Available from CTAN, `macros/latex/contrib/mparhack`.

Wilson, Peter. "The Memoir Class". 2010. Available from CTAN, `macros/latex/contrib/memoir`.

◇ Stephen Hicks
`sdh (at) google dot com`
(This work was conducted independent of the author's role at Google.)

Using LaTeX to generate dynamic mathematics worksheets for the web

Pavneet Arora

Abstract

Mathematics worksheet generators abound on the web. Many use static content and focus on graphics and animation in order to package the material in an appealing manner. This approach comes across as a fight for *eyeballs* — all too common when trying to attract the target audience on the Internet. The emphasis on form often displaces the basis of learning at the primary education level, which is simple practice. Beginning with an exploration of effective learning strategies for grade school mathematics, the use of LaTeX to generate dynamic mathematics worksheets — lots and lots of them — is discussed.

1 Introduction

The impetus of this project came from witnessing my daughter's struggles with mastering maths in early grades. The teachers — well intentioned as they were — felt that she was getting bogged down in mastering basic numeracy and that once freed from this mechanical process, she would be able to progress more quickly. Unfortunately, the evidence was to the contrary: she continued to struggle, and moreover slipped further and further behind.

In this paper, I discuss what I believe to be the importance of *numeracy* as a necessary step in achieving mathematical literacy, and the value of its co-conspirator, *practice*, which is the essential method of learning for young students.

The reader would be right in criticizing the paper's lack of statistical evidence to support the conclusions, and may in fact consider these conclusions to be mere conjectures. To this I plead *mea culpa*. I hope that a reasoned argument will suffice for the time being, and perhaps encourage others to take the exploration further. In my defence, however, I did not seek out to prove the correctness of my thesis; I was simply searching for techniques to help my daughter achieve tangible results.

2 The decline of practice

It is my belief that like for any skill, practice is at the heart of mastering elementary school mathematics. However, in many modern textbooks the number of practice problems is shrinking almost as if to imply that those who are unable to master the topic within the prescribed number of questions should consider themselves incapable of ever doing so. Best if they were to move on and try their hand with the next topic, or worse, consider mathematics as one of those areas that will forever remain inscrutable to them.

There are at least two possible reasons for this:

1. As the standardized curriculum grows, there is less time available to devote to each study unit. Lean manufacturing techniques seem to be permeating down into education.

2. As rote learning, a misnomer which will be discussed below, is removed from primary school education, the opportunity to reinforce learning with practice is also taken away from students who would otherwise greatly benefit.

But what if a student is capable of mastering a subject, but requires a great deal of reinforcement of the material in order to do so?

In order to fill the gap of what I felt was the diminishing availability of problem sets associated with each topic, the inspiration to generate dynamic worksheets struck and even more importantly making them available on the web. Naturally, this led me to seek out the use of TeX and LaTeX as the means by which to typeset mathematics worksheets effectively. Combining the two — that is generating problem sets dynamically and creating well typeset worksheets — will, I hope, tilt the balance back towards a reliance on practice, which for some students may mean much more than the median, and bring about a more inclusive approach to mathematics learning.

3 Rote vs. practice

It is important to distinguish from the onset a distinction between *rote* and *practice*. The reason being that rote learning is increasingly being considered superfluous in an age of technical wonderment. Why spend time memorizing anything when search engines are far more effective surrogates?

While rote and practice may be related in term of mechanics — they both rely on repetition — they differ in intent, and this difference strikes at the heart of the problem with tarnishing all forms of repetitious learning as rote. Rote implies tedium, or repetitious activity performed without purpose. Contrast that with practice, which is repetition done with the intent of gaining in capabilities. This in itself should be enough to show the wide gulf between the two.

However, there is a secondary aspect to the definition of practice which is that of gaining proficiency that leads us to consider just how we learn:

1. Do we first gain a kernel of comprehension when introduced to a topic, and then use practice

to reinforce that comprehension and achieve proficiency?

2. Or is it by the very act of practice that we get to even a basic level of comprehension? That is, does proficiency, even that acquired through mechanical repetition, lead then to comprehension?

It is often assumed, I feel, that the first approach is the only correct one.

When we reduce or remove the level of practice associated with learning a topic, we are adhering to this first approach, which may in fact be perfectly adequate for many students. However, I believe that there is a category of students for whom comprehension isn't so easy to come by through mere explanation. For these students, it is through the mechanical act of repetition that the topic reveals itself as patterns emerge.

Consider this as an example of that primary heuristic technique: that of trial and error.

When we denigrate all repetitious activity by labelling it as rote, we relegate this second important learning technique to the dustbin. As a consequence, students whose learning pattern mirrors the second approach are excluded from the learning process.

This distinction in approaches may be more easily recognized in the area of computer programming, where the relative numbers who fall into the two categories are reversed. It is a rare individual indeed who can proceed from learning a programming language or algorithm to immediate deep comprehension and proficiency. More often it requires the tentative steps of coding and stumbling with syntactic or semantic programming errors before the behaviour of the language and the program reveals itself. This trial and error method, along with isolating function to small digestible chunks is at the heart of software engineering. Should we not then elevate this learning approach to be on equal footing to the more linear one of elucidation leading to illumination?

4 The value of numeracy

If we accept that practice, even if in some cases this means much more practice than we might deem necessary, is essential for learning then what value does numeracy play towards the ultimate goal of mathematical literacy?

Calculators, which lost their novelty long ago, are now considered either relics of another earlier age or as commodities with little intrinsic value. There may be something to the concurrent decline in value of the once exotic calculator, and the value that we place on human numeracy. After all when a mechanical device that costs less than a fast-food meal is able to calculate numbers reliably and quickly, what need then of teaching numeracy to young students?

We are sometimes too quick at placing a value on a skill based solely on the economic cost needed to substitute for it. This mapping serves up, I believe, a grave fallacy when it comes to the learning of mathematics. Numeracy allows young children to play with and come to appreciate the behaviour of the most basic of mathematical elements, which are numbers. It gives them a chance to begin to recognize patterns and more importantly to gain a sense of mathematical intuition through a mastery of arithmetic.

Without this intuition, some students are unable to make the leap to symbolic manipulation as with algebra. But even before this, understanding fractions becomes difficult because unless one understands whole numbers how can one appreciate fractions of numbers and the delicate interplay that follows? And fractions are often a convenient entry into basic geometry. It is my assertion that once the foundation of numeracy is undermined, we make the resulting edifice of mathematical literacy shaky at best, and for some students impossible to construct.

Equally important, but in an entirely different plane, is that it takes away an important avenue through which young children are able to express mastery over their environment. Like mastery over language, basic numeracy allows them to interact with, recognize patterns within, give a name to, and to apply groupings to the physical world around them. The value of this should not be overlooked. It is an important aspect of giving a child self-confidence in their own ability to express their independent judgements.

5 Method

If one accepts the motivation outlined in the previous sections, the question remains, "How best to offer up practice when motivation for it is limited in a school setting?" This is the question that confronted me when I was trying to come up with learning materials for my daughter.

A search for mathematics worksheets on the web revealed an emphasis on interactive presentations, with insufficient variation in the problems themselves to fulfil my goal of increasing the level of practice in my daughter's study. Other sites had limited topics and very basic levels of difficulty.

From this search, I sought to construct worksheets that fulfilled the following design criteria:

1. The list of worksheets should be easily extensible. That is, if the worksheets were to be

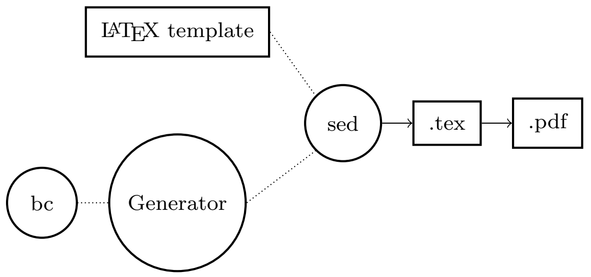
offered up on a web site, which was a stated goal, then it should be relatively easy to add to their numbers without having to affect the web site programming. This implied a database driven approach to cataloguing and invoking worksheets.

2. Parameterized worksheet generation. Being dynamic, the worksheet generators should be able to generate different groupings by merely specifying parameters. To put this in terms of concrete examples, multiplication worksheets could be limited to generate problems dealing with, for instance, multiplication tables between 1–4, 5–8, 9–12, or any specified range. Or in the case of geometry, worksheets on recognizing angles could be limited with the use of parameters to showcase only certain types of angles, i.e., only acute and obtuse.
3. Graduated levels of difficulty. An essential component of each worksheet generator was that not only would it allow for different operands, but that it should take the student from the basics of a topic gradually through more difficult aspects of the same topic.
4. The ability to create beautiful documents. This final criteria led me naturally to the doorstep of TeX and LaTeX.

Given this desired feature set, any number of programming languages and frameworks could have been chosen. Ruby on Rails was selected only because the author had some experience with Ruby, the language. This provided the necessary components to put the application on the web.

In order to animate the application, basic Unix tools were used: `bash`, `sed`, `bc`, `cron`. Development work was done under Ubuntu, and the application then deployed on a Sun Microsystems SPARCserver running Solaris 10. If nothing else, it showed the extent and ease with which frameworks and TeX Live accommodate cross-platform development.

The diagram below illustrates the architecture:



Each group of worksheets that are related by topic have a `bash` script associated with them. The script utilizes the built-in random number generator to generate, and `bc` to then normalize the operands. The operands are substituted into the base LaTeX template for that group with the use of `sed` to generate a spooled `.tex` file. This file is compiled into a `.pdf` file which is served back to the client browser. The invocation command, along with the associated parameters for a given worksheet at a given level of difficulty are stored in a database, as are the counters used to instantiate the spooled `.tex` file.

6 Conclusions

The value of both numeracy and practice in the teaching of grade school mathematics has been presented in this paper. The discussion has also touched upon the larger question of just how children learn. It is my hope that this work will encourage those children who may struggle with the subject to look upon it not as an insurmountable mountain, but only as a road less travelled, one that they have the privilege of walking along and in turn teaching us, by their example, just what learning really means.

7 Acknowledgements

The name of the web site, `www.bansisworld.org`, takes its inspiration from that of my late maternal grandfather, Prof. Bansi Lal, Professor Emeritus of the Department of Mathematics at D.A.V. College, Jalandhar, India. He spent much of his career focused on the teaching of mathematics, and it is my hope that Bansi's World in this modern web context will become synonymous with the earlier, original incantation of "Bansi Lal's world", which intimated the idea of open mathematics education.

◇ Pavneet Arora
Endeavour House
11 Kingsgate Pl
Bolton ON L7E 5Z5
Canada
`pavneet_arora (at) bansisworld dot org`
`http://www.bansisworld.org`

Writing the first LaTeX book

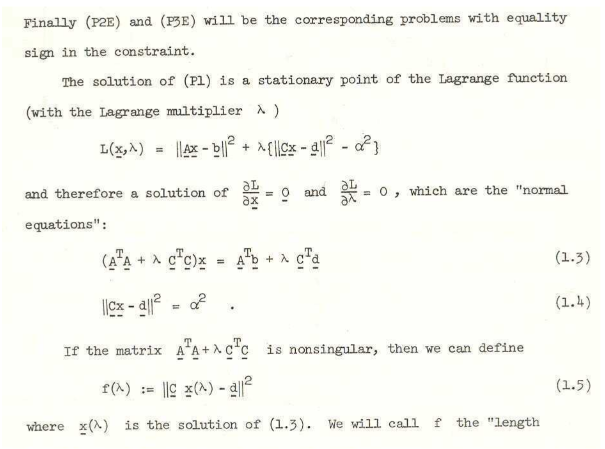
Walter Gander

Abstract

In 1984 I wanted to write a German textbook called "Computermathematik" using the typesetting system TeX developed by Don Knuth, which I have always admired and which I have been aware of since my first sabbatical year in Stanford in 1977. Mark Kent, a graduate student at Stanford in 1984, pointed out to me that Leslie Lamport had just finished a new typesetting system called LaTeX which I might want to use instead. I did and in fall 1984 I had finished the (at least I think) first book written in LaTeX. In this historical talk I will present some reminiscences how the book was produced.

1 First encounter with TeX

In 1977/78 I spent a year at Stanford as a postdoc writing my Habilitation. It was still the time when technical typists were typing in papers or books for their professors. I was very lucky to have had my Stanford report typed by Phyllis Winkler, the technical typist of Don Knuth, probably the best at Stanford. I gave her my hand-written manuscript and she typed it very efficiently using an electric typewriter. An excerpt is shown in Figure 1.



Finally (P2E) and (P3E) will be the corresponding problems with equality sign in the constraint.

The solution of (P1) is a stationary point of the Lagrange function (with the Lagrange multiplier λ)

$$L(\underline{x},\lambda) = \|\underline{A}\underline{x} - \underline{b}\|^2 + \lambda\{\|\underline{C}\underline{x} - \underline{d}\|^2 - \alpha^2\}$$

and therefore a solution of $\frac{\partial L}{\partial \underline{x}} = \underline{0}$ and $\frac{\partial L}{\partial \lambda} = 0$, which are the "normal equations":

$$(\underline{A}^T\underline{A} + \lambda\, \underline{C}^T\underline{C})\underline{x} = \underline{A}^T\underline{b} + \lambda\, \underline{C}^T\underline{d} \qquad (1.3)$$

$$\|\underline{C}\underline{x} - \underline{d}\|^2 = \alpha^2 \quad . \qquad (1.4)$$

If the matrix $\underline{A}^T\underline{A}+\lambda\underline{C}^T\underline{C}$ is nonsingular, then we can define

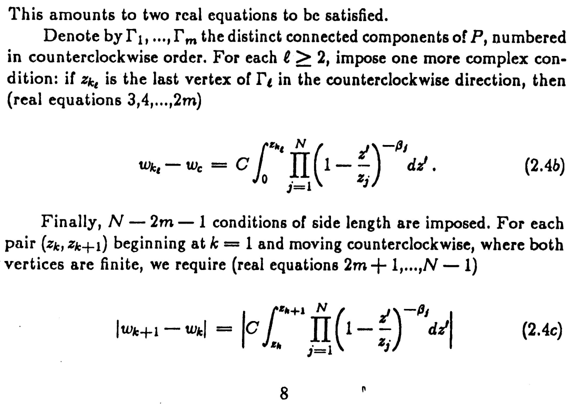
$$f(\lambda) := \|\underline{C}\ \underline{x}(\lambda) - \underline{d}\|^2 \qquad (1.5)$$

where $\underline{x}(\lambda)$ is the solution of (1.3). We will call f the "length

Figure 1: Mathematical typing: State of the art c. 1977 at Stanford

One day in the printer room, when I was retrieving some program output, I was really amazed to see a page of a printed book coming out of the printer. I could not really understand what this was, the only thing I could imagine was a photocopy of a page of a mathematical book. However, no, it was not that — it was some TeX output which Don had sent to the printer.

I returned home to Switzerland in fall 1978 and continued my job as professor at the University of Applied Sciences in Buchs in the Rhine-Valley. From Stanford I began to receive beautifully printed mathematical documents, not typeset in the traditional way but generated with TeX. One of those early ones was the PhD thesis of Nick Trefethen (see Figure 2).



This amounts to two real equations to be satisfied.

Denote by $\Gamma_1, ..., \Gamma_m$ the distinct connected components of P, numbered in counterclockwise order. For each $\ell \geq 2$, impose one more complex condition: if z_{k_ℓ} is the last vertex of Γ_ℓ in the counterclockwise direction, then (real equations 3,4,...,2m)

$$w_{k_\ell} - w_c = C\int_0^{z_{k_\ell}} \prod_{j=1}^{N}\left(1-\frac{z'}{z_j}\right)^{-\beta_j} dz' . \qquad (2.4b)$$

Finally, $N - 2m - 1$ conditions of side length are imposed. For each pair (z_k, z_{k+1}) beginning at $k = 1$ and moving counterclockwise, where both vertices are finite, we require (real equations $2m+1,...,N-1$)

$$|w_{k+1} - w_k| = \left|C\int_{z_k}^{z_{k+1}} \prod_{j=1}^{N}\left(1-\frac{z'}{z_j}\right)^{-\beta_j} dz'\right| \qquad (2.4c)$$

8

Figure 2: Part of Nick Trefethen's PhD thesis

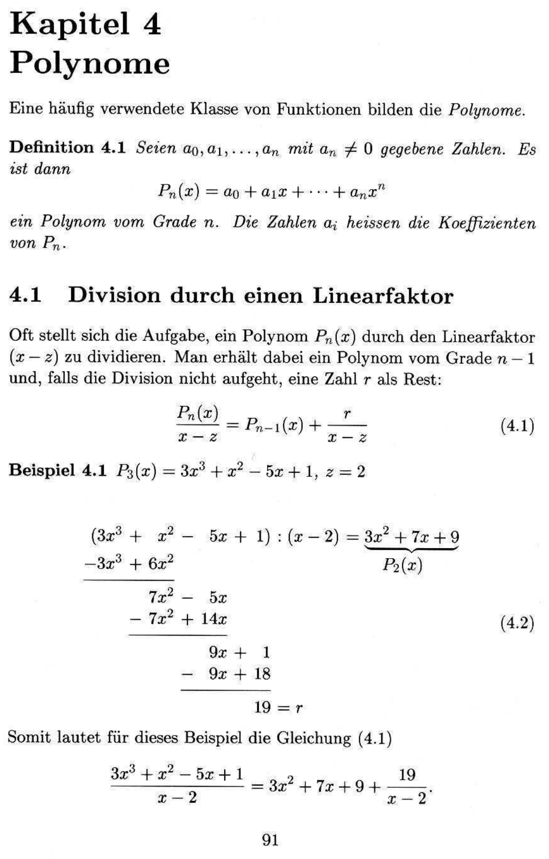
2 Writing the book

A few years later I was due for a sabbatical and I decided to use it to write a German textbook with the title "Computermathematik" which would teach algorithms written in Pascal, mostly focussed on topics in numerical analysis. Of course I was determined to learn TeX and write my book using this new text-processing system.

So I went with my two daughters of 9 and 11 years and a suitcase full of hand-written notes to Stanford. My wife Heidi had just started a new job and had to stay in Switzerland but would visit us during the vacations.

At the beginning I had to learn to use the computer (a UNIX VAX) on which TeX was installed. Mark Kent, a graduate student in the Computer Science Department, working in Numerical Analysis with Gene Golub, helped me in many ways to get me going. I learned to use the Emacs editor to write TeX source files. When Mark realized that I was going to write a book he pointed out to me that just a few weeks earlier Leslie Lamport had published a manual in which he described his system LaTeX, a collection of TeX macros which should help a book writer a lot since it would take him to a higher book-producing abstraction level. Simply write `\chapter{ }` and forget about the actual size of fonts, distance to text, numbering etc. It sounded good to me and since I had to learn anyway, either TEX or LATEX, I decided to go for LATEX.

The first chapter I started to write was Chapter 4 of the book with the title “Polynome”. This was already quite a challenge. Showing how to divide a polynomial by some factor in the form that one would write it up when doing it by hand is quite demanding for a LATEX beginner. The first page of this chapter is displayed in Figure 3.



Kapitel 4
Polynome

Eine häufig verwendete Klasse von Funktionen bilden die *Polynome*.

Definition 4.1 *Seien $a_0, a_1, \ldots, a_n$ mit $a_n \neq 0$ gegebene Zahlen. Es ist dann*

$$P_n(x) = a_0 + a_1 x + \cdots + a_n x^n$$

ein Polynom vom Grade n. Die Zahlen a_i heissen die Koeffizienten von P_n.

4.1 Division durch einen Linearfaktor

Oft stellt sich die Aufgabe, ein Polynom $P_n(x)$ durch den Linearfaktor $(x-z)$ zu dividieren. Man erhält dabei ein Polynom vom Grade $n-1$ und, falls die Division nicht aufgeht, eine Zahl r als Rest:

$$\frac{P_n(x)}{x-z} = P_{n-1}(x) + \frac{r}{x-z} \qquad (4.1)$$

Beispiel 4.1 $P_3(x) = 3x^3 + x^2 - 5x + 1,\ z = 2$

$$\begin{array}{rcrcrcrlc}
(3x^3 & + & x^2 & - & 5x & + & 1) & :(x-2) & = \underbrace{3x^2+7x+9}_{P_2(x)} \\
-3x^3 & + & 6x^2 & & & & & & \\
\hline
 & & 7x^2 & - & 5x & & & & \\
 & - & 7x^2 & + & 14x & & & & \\
\hline
 & & & & 9x & + & 1 & & \\
 & & & - & 9x & + & 18 & & \\
\hline
 & & & & & & 19 = r & &
\end{array} \qquad (4.2)$$

Somit lautet für dieses Beispiel die Gleichung (4.1)

$$\frac{3x^3 + x^2 - 5x + 1}{x-2} = 3x^2 + 7x + 9 + \frac{19}{x-2}.$$

91

Figure 3: First page typeset

And here is some of the corresponding source, which will still look quite familiar to LATEX users today (slightly reformatted for *TUGboat*):

```
\chapter{Polynome}
Eine h\"aufig verwendete Klasse von Funktionen
bilden die {\em ^{Polynome}}.

{\defi \it Seien $a_0, a_1, \ldots, a_n$ [...]
\section{Division durch einen Linearfaktor}
Oft stellt sich die Aufgabe [...] als Rest:
\begin{equation}
 \label{44.2}
 \frac{P_n(x)} {x-z} = P_{n-1}(x) + \frac{r}{x-z}
\end{equation}
```

```
\begin{bsp}\label{41}
$P_3(x)=3x^3+x^2-5x+1$,   $z=2$
\end{bsp}
\medskip

\begin{equation} \label{44.*}
\arraycolsep 2pt
   \begin{array}{rcrcrcrlc}
    (3x^3 &+& x^2   &-& 5x  &+& 1) & :(x-2)
                       & = \underbrace{3x^2+7x+9}\\
    -3x^3 &+& 6x^2  & &     & &    &   & P_2(x)
\\[-2\smallskipamount]
\multicolumn{3}{c}{\hrulefill&&&&&&\\
         & & 7x^2  &-& 5x  & &    &        &
         &-& 7x^2  &+&14x  & &    &        &
[...]
```

Since babel did not yet exist, issues relating to typesetting a book in German were resolved by hand. For example, I inserted manual hyphenations such as `Re\-chen\-ma\-schi\-nen`. Mark Kent wrote a style file `la-macros.sty` redefining internal LATEX commands so that ‘Kapitel’ would be typeset instead of ‘Chapter’, and so on.

Including graphics was not yet as convenient as it is today. Nowadays we use `\includegraphics` to include all possible graphical material in various formats. In 1984 I used basic graphic commands provided by LATEX to, for example, produce Figure 4:

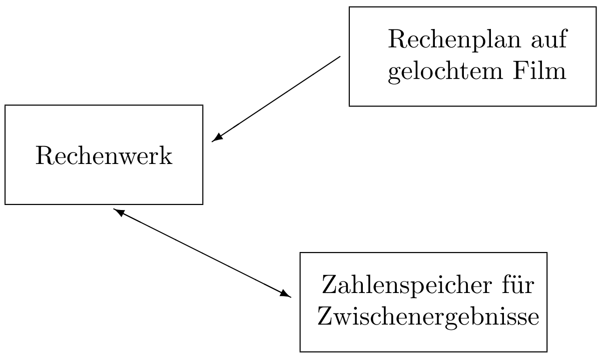

Rechenplan auf gelochtem Film
Rechenwerk
Zahlenspeicher für Zwischenergebnisse

Figure 4: Erste Computer

```
\begin{figure}[htb]
  \begin{center}
  {\setlength{\unitlength}{8mm}
  \begin{picture}(13,7)(0,0)
       \put(0,3){\framebox(4,2){Rechenwerk}}
       \put(4,2){\vector(2,-1){1.8}}
       \put(4,2){\vector(-2,1){1.8}}
       \put(6,0){\framebox(5,2){
          \shortstack{Zahlenspeicher f\"ur\\
                      Zwischenergebnisse}}}
       \put(7,5){\framebox(5,2){
          \shortstack{Rechenplan auf\\
                    gelochtem Film}}
      \put(6.8,6){\vector(-3,-2){2.6}}
  \end{picture}
  }
  \end{center}
\caption{Erste Computer}  \label{1F1}
\end{figure}
```

Again today the situation has completely changed. We have tools to convert formats, *e.g.* from eps to pdf and tools for graphical construction, most notably MetaPost.

Another challenge was to typeset Pascal programs. Today most of us do not bother too much. We simply use `\verbatim` or `\verbatiminput` to include programs. I had the idea to write the reserved words like **begin**, **end**, **for**, etc., in boldface and to indent always by three spaces after a **begin** or when using for-loops or if-statements. Of course I did not want to retype the Pascal programs, this would be too likely a source of errors. So I finally asked Leslie Lamport by e-mail what he would recommend. He suggested using the tabbing environment. My Pascal programs were written with capitalized reserved words. As an example consider the Pascal function to compute a square root:

```
FUNCTION quadratwurzel(a:real):real;
VAR xneu,xalt:real;
BEGIN
  xneu:=(1+a)/2;
  REPEAT
    xalt:=xneu; xneu:=(xalt+a/xalt)/2
  UNTIL xneu>=xalt;
  quadratwurzel:=xneu
END;
```

A pragmatic way to proceed was to replace a capitalized reserved word like `BEGIN` by `\BEGIN`, where I had done `\newcommand{\BEGIN}{{\bf begin }\+}`. The characters `\+` would cause the next line to be indented in the tabbing environment. More changes like writing `$` to use math-mode and re-indenting I did by hand using Emacs. Defining the LaTeX command

```
\newcommand{\SETTABS}
{123\=456\=789\=123\=456\=789
 \=123\=456\=789\=123\=\kill
 \>\>\>\+\+\+}
```

and using Emacs I transformed it to

```
\begin{alg} \label{3wurzel} \it
\begin{tabbing} \SETTABS \\
\FUNCTION quadratwurzel(a:real):real;\\
\VAR xneu, xalt : real ; \\
\BEGIN \\
   $ xneu := (1+a)/2;$ \\
   \REPEAT \\
    $xalt:=xneu; xneu:=(xalt+a/xalt)/2$ \\
\< \- \UNTIL $xneu \ge xalt;$\\
   quadratwurzel := xneu \\
\END
\end{tabbing}
\end{alg}
```

After processing with LaTeX the result looked quite satisfactory:[1]

```
function quadratwurzel(a:real):real;
var xneu, xalt : real ;
begin
   $xneu := (1 + a)/2;$
   repeat
      $xalt := xneu; xneu := (xalt + a/xalt)/2$
   until $xneu \geq xalt;$
   quadratwurzel := xneu
end
```

I typed the whole summer, the children were busy attending Escondido School on campus. During summer vacations Heidi came to visit us and look after our daughters. Finally in fall the book was finished. Voy and Gio Wiederhold invited us all to a party at their house to celebrate this event. Don Knuth was also with us and said: “Finally it is proved that LaTeX is useful!”

3 Book revision

The book was written. But of course I still needed to proofread it carefully. Back in Switzerland I offered the book to publishers for German textbooks, among them Birkhäuser in Switzerland, Springer and Oldenburg in Germany. All the publishers were amazed about the quality of typing and all of them accepted the book and made me an offer. For patriotic reasons I then chose Birkhäuser.

When proofreading I found of course typos and other minor things which needed to be fixed. There was no way to process LaTeX in Switzerland, I did not even know of a TeX installation. So I decided to fly back in the winter break at beginning of January 1985 to do the changes at Stanford and print the final camera ready version of the book on the best available printer, the Alphatype machine in the basement. This rather expensive way of doing changes was the only possibility that I had at that time. Switzerland was not yet connected to the Internet. So I spent a week at Stanford, produced a new corrected version of the book and wanted to print the final copy for the publisher. However, I did not succeed because the Alphatype printer was down. I discussed with Mark Kent what to do and we decided that I would

[1] Observant readers will note that the font used for punctuation varies. The typography before TeX was in such bad shape that such “minor flaws” in the otherwise wonderful output were simply overlooked and not taken care of.

return to Switzerland and that he would print the book when the printer was operating again and send me the manuscript by ordinary mail. Indeed this worked fine, two weeks later I received a beautifully typed manuscript.

Looking it through I was terrified: at some place the page break was different than what I had printed in Stanford before. One table was moved and there was a half page empty. Fix it and have Mark printed it again using this expensive printer and paper? I finally decided for a pragmatic solution. I took a scissor and glue and copy pasted the few pages by hand as I expected them to look in my first output.

What was the reason for this new page break? Well, in my absence Leslie Lamport made some small changes to LaTeX and installed a new version. We did not notice this and thus the different page break occurred.

4 Epilogue

I had to write a second volume of my book which included the solutions of all exercises, which are mostly programming assignment. I bought a desktop computer Olivetti M24 for some $6,000 with a 10 MB hard disk. There was a company called Micro-TeX who had ported TeX to the IBM PC. I bought their floppy disk and installed TeX on my Olivetti. It used up half of my disk-space! LaTeX was not available. So I wrote in 1985 my solution book using plain TeX on my own PC at home. Printing on the dot matrix printer did not look so nice as with Dover and furthermore was terribly slow. When the book was finished, I looked around to find a TeX installation in Switzerland. I found one in the Institute of Astronomy at ETH. Professor Jan Olof Stenflo was one of the first to have TeX and LaTeX installed in Switzerland. So I processed the final version of the second book written in plain TeX on his computer in Switzerland.

The first LaTeX book is no longer in print; it had a second edition in 1992. The publisher Birkhäuser has returned the copyrights to me. So I decided to give the book for free distribution to Google. This seems to be a very long procedure. Therefore I also made it available on `http://www.educ.ethz.ch/unt/um/inf/ad/cm` (figure 5).

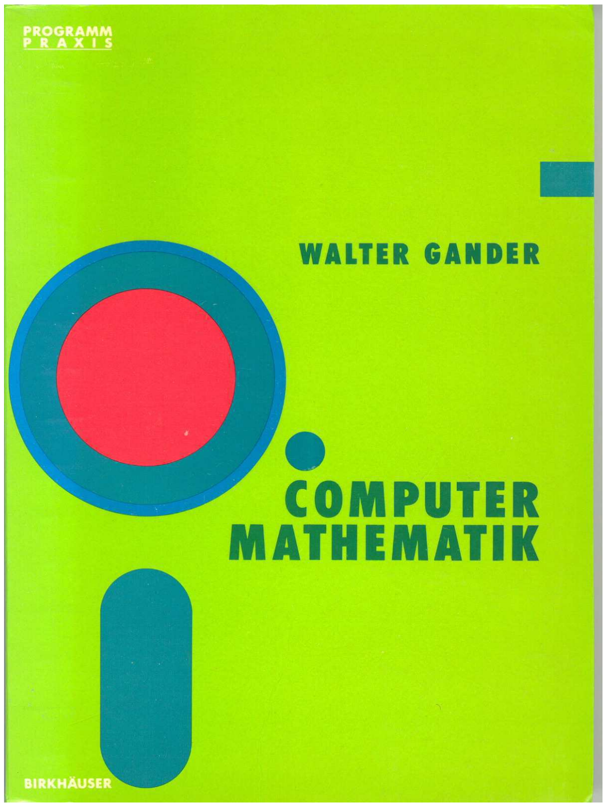


Figure 5: Cover of the now freely-available book

I wanted to produce a pdf file of the book for the web page. Now the amazing result: without any major changes the book compiled using `pdflatex`! I do not know of any other typesetting system that is as stable over more than 25 years.

◇ Walter Gander
ETH Zurich
`http://www.inf.ethz.ch/personal/gander/`

TEX helps you learn Chinese character meanings

Alan Hoenig

My story begins about 6 or 7 years ago now when, on a whim, I decided to study Mandarin Chinese. I've had these whims often over the years, and I know that the more intense it is, the sooner it burns itself out sooner or later they burn themselves out — which is why this time I focused on the interesting stuff and neglected the dull material, which in this case was Chinese characters themselves and their meanings. But, days, months, and now years passed, and I stayed intrigued, so it was a mistake to have ignored them.

But studying Chinese characters turned out to be tough — too tough. I couldn't seem to remember more than a handful accurately, not enough to make any real headway.

I found this frustrating, but fortunately before ditching everything, I realized it couldn't hurt to apply one of the great lessons of Metafont. That is, rather than to actually *study* this material, I took a step back and thought about *how* to study this material. I looked around to see what other people had to say and what methods they used, and then I came up with the following scheme, one that seems to work pretty well.

I took as an initial pool the 2000 most frequently used characters. (I used the so-called simplified character set, the characters in official use by the People's Republic of China.) Then I imposed an order on them — not an alphabetical or numerical order, but one based on how complex each character is — how easy (or not) it is to write. I arranged them from simplest to increasingly complicated.

Then I used an induction-based learning scheme relying on three essential platforms:

1. The form and meaning of any character depends only on characters and components that appear earlier in the sequence.
2. You can remember this form and its meaning by means of relatively simple/straightforward mnemonic narratives which use prior characters (characters that are already known) as elements in this story. These characters are already known because they precede the current character in the sequence.
3. Finally, the initial item in this sequence must be easy to remember all by itself.

As far as the mnemonic stories go, anything — any kind of story connecting the components, any pun or play on words, and any kind of outlandish sce-

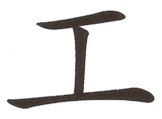


Figure 4: 'Labor' or 'work'.

This is Linux Libertine ff ffi ffl fi fl Th Abcde Fghij 0 1 2 3 4

Bold *Italic* ***Bold italic***

Figure 8: Linux Libertine: a small sample.

EZChinesey.com

info@EZChinesey.com
EZChinesey@gmail.com

Figure 9: Publishing venture and contact information.

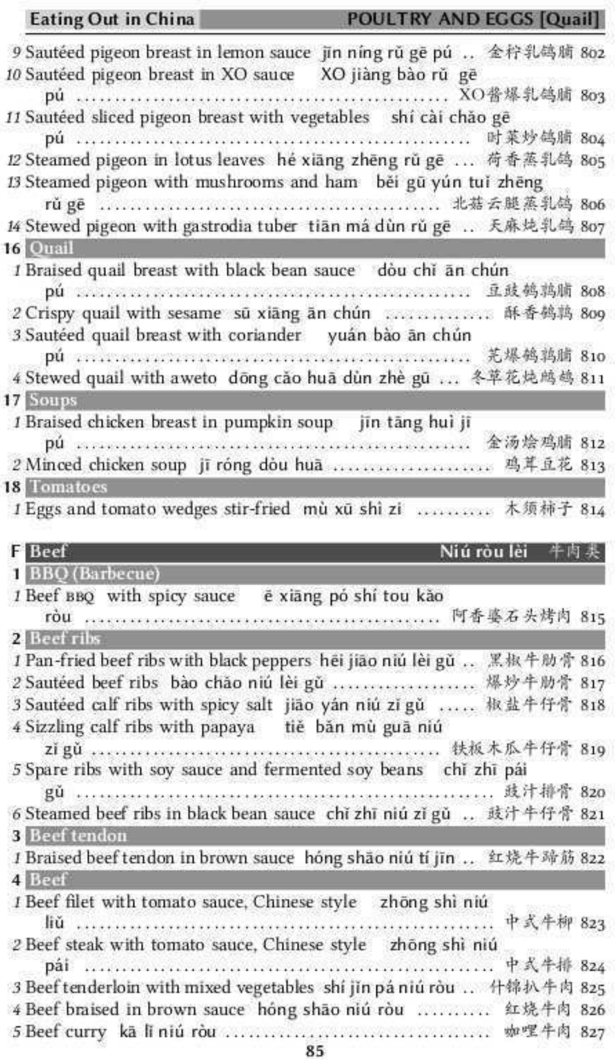
Finally, there's some other information just for fun — the stroke count and the frequency rank of each character.

Incidentally, the roman typeface I used for the book is the family Linux Libertine, an OpenType family designed by Philipp Poll and available for free use (fig. 8). It's a snap to install these fonts for use by XƎTEX (at least it is on the Mac platform), and when you do so properly you get the entire suite of proper TEX behavior, including small caps, all the standard ligatures, all special German ligatures, and an intriguing 'T-h' ligature which you can see in the figure.

Anyway, the result is an actual book containing the stories for about a hundred components and twenty-one hundred some-odd characters. For more fun, and for the thrill of fulfilling a long-held dream, I decided to print and publish this book on my own.

To that end, my wife and I set up a small publishing company `EZChinesey.com`; can you guess its associated web site? I'm not exactly rolling in royalties, but running your own company turns out to be a great adventure with lots of unexpected twists and turns. I recommend it highly (fig. 9). By the way, I know I'm no typographer, and so I am desperately seeking feedback to improve the format of the character panels, and I hope anyone and everyone with better ideas will feel free to bring them to my attention. I encourage anybody with suggestions for improvements to please get in touch with me.

This book is the first of what I hope will be a series of 'EZChinesey Guides'. A second one will appear later this summer, and it'll be a guide for travelers to China who want to eat in local restaurants. Think of it as a menu translation guide, with translations for over 3000 Chinese menu items. Its format is quite different from that of my character volume; figure displays a typical page.



Eating Out in China **POULTRY AND EGGS [Quail]**

9 Sautéed pigeon breast in lemon sauce jīn níng rǔ gē pú .. 金柠乳鸽脯 802
10 Sautéed pigeon breast in XO sauce XO jiàng bào rǔ gē pú .... XO酱爆乳鸽脯 803
11 Sautéed sliced pigeon breast with vegetables shí cài chǎo gē pú .... 时菜炒鸽脯 804
12 Steamed pigeon in lotus leaves hé xiāng zhēng rǔ gē ... 荷香蒸乳鸽 805
13 Steamed pigeon with mushrooms and ham běi gū yún tuǐ zhēng rǔ gē .... 北菇云腿蒸乳鸽 806
14 Stewed pigeon with gastrodia tuber tiān má dùn rǔ gē .. 天麻炖乳鸽 807

16 Quail

1 Braised quail breast with black bean sauce dòu chǐ ān chún pú .... 豆豉鹌鹑脯 808
2 Crispy quail with sesame sū xiāng ān chún .... 酥香鹌鹑 809
3 Sautéed quail breast with coriander yuán bào ān chún pú .... 芫爆鹌鹑脯 810
4 Stewed quail with aweto dōng cǎo huā dùn zhè gū ... 冬草花炖鹧鸪 811

17 Soups

1 Braised chicken breast in pumpkin soup jīn tāng huì jī pú .... 金汤烩鸡脯 812
2 Minced chicken soup jī róng dòu huā .... 鸡茸豆花 813

18 Tomatoes

1 Eggs and tomato wedges stir-fried mù xū shì zi .... 木须柿子 814

F **Beef** **Niú ròu lèi 牛肉类**

1 BBQ (Barbecue)

1 Beef BBQ with spicy sauce ē xiāng pó shí tou kǎo ròu .... 阿香婆石头烤肉 815

2 Beef ribs

1 Pan-fried beef ribs with black peppers hēi jiāo niú lèi gǔ .. 黑椒牛肋骨 816
2 Sautéed beef ribs bào chǎo niú lèi gǔ .... 爆炒牛肋骨 817
3 Sautéed calf ribs with spicy salt jiāo yán niú zǐ gǔ ..... 椒盐牛仔骨 818
4 Sizzling calf ribs with papaya tiě bǎn mù guā niú zǐ gǔ .... 铁板木瓜牛仔骨 819
5 Spare ribs with soy sauce and fermented soy beans chǐ zhī pái gǔ .... 豉汁排骨 820
6 Steamed beef ribs in black bean sauce chǐ zhī niú zǐ gǔ .. 豉汁牛仔骨 821

3 Beef tendon

1 Braised beef tendon in brown sauce hóng shāo niú tí jīn .. 红烧牛蹄筋 822

4 Beef

1 Beef filet with tomato sauce, Chinese style zhōng shì niú liǔ .... 中式牛柳 823
2 Beef steak with tomato sauce, Chinese style zhōng shì niú pái .... 中式牛排 824
3 Beef tenderloin with mixed vegetables shí jǐn pá niú ròu .. 什锦扒牛肉 825
4 Beef braised in brown sauce hóng shāo niú ròu .... 红烧牛肉 826
5 Beef curry kā lǐ niú ròu .... 咖喱牛肉 827

85

Figure 10: A typical page from the book *Eating Out in China.*

A third volume will be similar to the one I've spoken about today, but it will deal with the traditional characters, instead of the simplified ones in the current book.

I'm anxious to explore a different format for this upcoming volume, the traditional character volume, and I'd like to show a preliminary version to you, again with the goal of soliciting suggestions for improvement (fig. 11). New information includes the stroke order diagrams you see, which help when you review characters you've learned. Perhaps you recall this format is slightly different from the panels in figure 7.

I've typeset a sample panel twice, showcasing some interesting typefaces I've discovered. These fonts are from the collection that Google seems to

Classes, styles, conflicts: The biological realm of LaTeX

Didier Verna

Abstract

The LaTeX world is composed of thousands of software components, most notably classes and styles. Classes and styles are born, evolve or die, interact with each other, compete or cooperate, very much as living organisms do at the cellular level. This paper attempts to draw an extended analogy between the LaTeX biotope and cellular biology. By considering LaTeX documents as living organisms and styles as viruses that infect them, we are able to exhibit a set of behavioral patterns common to both worlds. We analyze infection methods, types and cures, and we show how LaTeX or cellular organisms are able to survive in a world of perpetual war.

1 Introduction

Every LaTeX user faces the "compatibility nightmare" one day or another. With such great intercession capability at hand (LaTeX code being able to redefine itself at will), a time comes inevitably when the compilation of a document fails, due to a class/style conflict. In an ideal world, class/style conflicts should only be a concern for package maintainers, not end users of LaTeX. Unfortunately, the world is real, not ideal, and end-user document compilation *does* break.

As both a class/style maintainer and a document author, I tried several times to come up with a systematic approach, or at least some general principles on how to handle class/style cross-compatibility in a smooth and gentle manner, but ultimately failed, because the situation is just too complex. Classes and styles evolve constantly, sometimes even in a backward-incompatible way. Classes and styles die, while new ones are born. Styles may conflict not only with classes but with other styles as well. Styles may be made aware of classes or other styles, but classes may be made aware of styles as well. Then, there is the influence of the end user who will combine all available material in a somewhat unpredictable way, possibly with his/her own personal additions, or even modifications to the available features.

This vicious circle basically never ends and leads to a paradoxical "If it ain't broke, then fix it" situation in which complex trickery is added to classes or styles, not to make them work out of the box, but to *prevent* potential breakages resulting from interactions with the outside world. In the end, the only realistic conclusion is that there is no solution to this problem, both because the system is too liberal, and because the human factor is too important. One cannot force a package author to write good quality (for some definition of "quality"), non-intrusive or even just bug-free code. One cannot force a package author to keep track of all potential conflicts with the rest of the LaTeX world, let alone fixing all of them by anticipation. One simply cannot prevent software evolution.

Facing this somewhat pessimistic conclusion, it is all the more intriguing to acknowledge the fact that the system still globally works. Despite the complexity of what happens behind the curtain, documents *are* being produced, and in some way, seeing a freshly compiled document pop up on the screen is just like witnessing a small miracle. When it doesn't compile, you don't really know why, but when it does compile, you really don't know why. This is the precise point at which the parallel with biology occurred to me. Any living being is by itself a miracle of complexity, and unfortunately, sometimes it breaks as well.

One Monday morning, I woke up with this vision of the LaTeX biotope, an emergent phenomenon whose global behavior cannot be comprehended, because it is in fact the result of a myriad of "macro"-interactions between smaller entities, themselves in perpetual evolution. In this paper, I would like to build bridges between LaTeX and biology, by viewing documents, classes and styles as living beings constantly mutating their geneTeX code in order to survive `\renewcommand` attacks....

The basis of our analogy is to consider LaTeX documents as living beings, and styles as viruses that infect them. Based on this picture, a number of puzzling similitudes can be found in the way organic/LaTeX material interact. In the following, we first describe how LaTeX documents can be morphologically compared to a specific kind of organic cells, then draw a parallel between genetic and programmatic material, and finally justify our view of styles as viruses. After that, we respectively draw interesting comparisons between existing viral or stylistic infection methods, infection types, and also possible cures.

2 Morphological analogy

In this section, we present a morphological analogy between LaTeX documents and a specific kind of organic cells from the so-called "eukaryotes" domain.

2.1 Eukaryotes

According to Whittaker's nomenclature [23], *eukaryotes* subsume four of the five "kingdoms" of life,

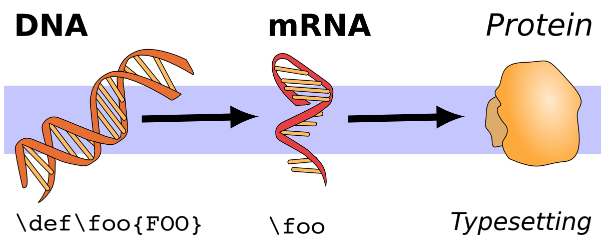


Figure 2: The geneTEX factory

information from deterioration. Next, the resulting mRNA is "read" by a ribosome, which will eventually produce the resulting protein.

3.2 TEX factory

The similarity with TEX macros (in fact, with functions in any programming language) is striking. The biologists themselves speak of genetic "code", because it is exactly that: just like a gene contains a formal specification for something to be synthesized, a TEX macro definition contains a formal specification for something to be executed. The result is not as concrete as a protein, but instead consists in some side-effect like the actual typesetting of a portion of a document.

Just like genes, TEX macros don't do anything on their own, but are available on demand. Figure 2 depicts the process of expressing a gene or executing a TEX macro. A macro *definition* is much like a gene, which encodes a formal specification. A macro *call* is much like a messenger: an actual instance or copy of the original information which is about to be concretely used. TEX does not use ribosomes, but a "mouth" and "stomach" instead (as per Donald Knuth's terminology in [6]), to perform macro expansion and (primitive) command execution.

In the remainder of this paper, we use the term "geneTEX material" to designate both genetic material in cells and programmatic material in documents.

4 Higher view of geneTEX material

So far, we have established a morphological ground on which to compare unicellular eukaryotes and LATEX documents, and we also have exhibited similarities in the way genetic and programmatic information is processed. It is now time to stand back a little and get some perspective on why this comparison is interesting.

4.1 Roles

We know that genes (or rather the information they encode) determine the way a cell will function. Let us consider two eukaryote cells, for instance resulting from the mitosis of a mother cell, and hence equipped with the same initial genome (in other words, the same functional potential). Why do these two cells eventually turn out to be different?

The difference comes from the fact that the set of actually expressed genes differ, because the orders emanating from the cytoplasms differ, in turn partly because ultimately, the cell's environments differ. In other words, different cytoplasms lead to cells functioning differently, even when the original genetic material is the same. In addition to that, variations in the environment also lead to more divergence from the two cytoplasms as time passes.

Now consider LATEX articles, reports, books, *etc.*, in the sense of their corresponding `\documentclass`. It is usually very easy to figure out the class of a document just by the look of it. Two articles look similar because their general layout is the same: it is dictated by the `article` class. Consequently, a document's class can be seen as its original geneTEX material. Two articles look roughly the same but are still different, just like two liver cells are *both* liver cells, but still different ones. If we extrapolate a little further, outside the unicellular world, the morphological similarities between documents of the same class are not unlike those between brothers and sisters or even twins (blue eyes, red hair, *etc.*) that may share the same genetic pool although expressed slightly differently.

Given this parallel, what makes two `article` documents different is also exactly what makes two liver cells different. The set of expressed genes/macros may differ (you might or might not use `\subsubsection`), the orders coming from the document's cytoplasm/body may differ (you may issue macro calls at different times and with different parameters), and in fact the whole documents' cytoplasms/bodies diverge (their respective text is not the same). The ultimate source of divergence is of course the documents' authors, who write different documents. A document author clearly takes the role of the cell's environment here.

4.2 Sources

A LATEX document. just like a cell, is a viable entity as soon as its initial geneTEX material is defined (its class), and it has a healthy body/cytoplasm. However, it is rare that a document is satisfied only with a class. In fact, a perhaps even more important and abundant source of geneTEX material is the use of *styles*. Styles provide the same kind of material as classes: macro definitions. The difference is that styles are not needed to give birth to a document. When some are used, however, the document may function slightly or sometimes very differently.

In this context, the idea of viewing styles as *viruses* that infect unicellular LaTeX documents turns out to be quite natural. Viruses are biological entities, mostly genetic components, that need a host cell to replicate themselves. Viruses are thus characterized by the fact they cannot perform their function on their own (this is why viruses are not considered as living entities [4, 13], although the debate is still open [11]). A LaTeX style has similar properties: it is basically useless as a standalone entity and needs to "infect" a host document in order to perform its function. Just like a virus, a style adds its own geneTeX material to the document's original pool.

Viruses are usually small compared to the organisms they infect, apart from two notable exceptions: the *mamavirus* and the *mimivirus*, which are twice as big as the average. A quick survey of the TeX Live 2009 distribution exhibits a surprising coincidence: among 2462 available `sty` files, the average size is around 327 lines of code (LoC), with a median at 134. Styles are indeed rather small. However, two of them are exceptionally bigger than the others: `texshade.sty` and `xq.sty`, with 14470 and 24535 LoC respectively. These styles can arguably be called the *mimistyle* and the *mamastyle* of LaTeX.

All these considerations make it interesting to analyze and compare the ways genetic material from cells and viruses, or programmatic material from classes and styles, interact. This is the purpose of the following sections.

5 Infection methods

In cells as in documents, there are many ways to be infected with new geneTeX material. The following methods come to mind in both worlds.

5.1 Exogenic

Perhaps the most common way for a document to be infected by a style is to "request" explicit infection by means of the `\usepackage` command. This results in the incorporation of the style into the document's preamble, the style being indeed an external LaTeX entity stored in a file of its own. We could even use the term "stylon" to denote the style file, in reference to the biological term "virion" which denotes the viral particle outside the cell it is bound to infect.

Such a style infection always occurs *after* the initial geneTeX material of the document has been defined, since `\documentclass` must appear first. As such, the style's material is not technically part of the original document's material. The infection occurs *afterwards*.

In biology, this process is said to be *exogenic*: the cell is infected by the virus after it has been created (for instance, after mitosis), and the genetic material brought by the virus is not part of the cell's original pool.

5.2 Endogenic

A document might however be infected by a style without even knowing it: a class may request infection by means of the `\RequirePackage` command. As such, when a document is created based on this class, even without explicit addition of any style in the preamble, the document is already infected. Arguably, this is a situation in which the geneTeX material brought by the style is indeed part of the original genome, because it is impossible to create a non-infected document based on that class.

This kind of infection is known to be *endogenic* in biology: when a new cell is born, it already contains some genetic material that would have required a former infection, for instance of the mother cell.

About 10% of our own genetic source material is currently estimated to be endogenic. A quick survey of TeX Live 2009 reveals that 259 out of the 271 available classes (95%) "suffer" from endogenic infections, by 4 styles on average (the median being 2). ***CurVe*** [21, 22], for instance, is infected by the `ltxtable`, `ifthen`, `calc` and `graphics` viruses. The QCM style is also endogenic to the QCM class [20].

5.3 Endosymbiosis

In biology, viruses are not the only source of external genetic material. *Endosymbiosis* is defined as the mutually beneficial cooperation between two living organisms, one (the *endosymbiont*) contained within the other. The so-called *endosymbiotic theory* [7] suggests that some organelles from eukaryote cells (*e.g.* mitochondria) actually come from the endosymbiosis of former prokaryotes (cells without a nucleus).

In the LaTeX world, we must acknowledge the fact that endosymbiosis is not as widespread as it should be. What we usually observe is the opposite phenomenon: the proliferation of a multitude of different packages that are meant to work together, or do more or less the same thing, instead of becoming one single and bigger animal. To mention a couple of examples, I have recently attempted twice to contact the author of `doc` about incorporating the features of `DoX` [17] (in other words, to turn `DoX` into an endosymbiont for `doc`) but got no response.[1] Not long ago, I also launched a thread on `comp.text.tex` entitled "Please, make it stop!" in which I mentioned

[1] It seems, however, that recent versions of `doc` do contain endosymbiotic versions of `newdoc`, so there is still hope....

a couple of packages doing a similar job (key/value processing). At the end of the thread, the number of such packages, as reported by different participants, amounted to 13. LaTeX definitely needs more endosymbiosis. Maybe the LaTeX 3 project will help in this regard.

As a final note on endosymbiosis, we must admit that our analogy falls short on one point: in biology, the symbiotic organisms are *living* creatures (mitochondria are semi-autonomous: they live in cells but have independent division capabilities). In our case, we are mostly talking of symbiotic relations between styles and/or classes, which are not considered “living” documents (see section 4.2 on page 164).

5.4 Exosymbiosis

An interesting phenomenon in package development seems to do the opposite of endosymbiosis. We call it *exosymbiosis*. Many existing styles originate from quick, local and often dirty hacks in specific documents, that are gradually abstracted away and cleaned up in order to become styles of their own, officially distributed in the form of `sty` files (a “bottom-up” development approach, in other words). In this situation, a symbiosis continues to exist, but one of the “organisms” is made external to the other, hence the choice of terminology.

Here is a concrete example of this. For many of my lectures, I use the Listings package for typesetting code excerpts, and include them in Beamer blocks. Providing nice shortcuts for doing so is not trivial if one wants to preserve control over both Beamer blocks and Listings options. The way I currently do this is to simply cut and paste the same 50 LoC into every new document I create over and over again. Soon, however, this will become a style of its own (probably called `lstblocks`) and released on CTAN.

This development process can indeed be regarded as the opposite of endosymbiosis: at first, a document features some geneTeX material that does not belong to its original pool, and by the way, just as mitochondria live in the cell's cytoplasm, LaTeX macros can be defined anywhere, including the document's body instead of preamble. However, if we let natural evolution happen for some time, this material will ultimately be extracted from the original document and become a style, which in turn will have the ability to infect other documents.

This is as if genetic material from a cell would have been extracted and became a virus.

5.5 Transduction *vs.* transfection

Biologists speak of *transduction* when genetic material is brought to a cell via a viral agent (when you use a style). When no viral agent is involved, the term *transfection* is used instead. Transfection corresponds to our version of endosymbiosis, where the geneTeX material is not brought to the document by a style, but simply by the document's author typing some macro definitions locally.

Interestingly enough, most cases of transfection are transitory (as opposed to stable): the genetic material is not copied into the cell's genome, so it is lost after the mitosis, just like you would need to cut and paste endosymbiotic macros over and over again into every new document, unless they are properly incorporated into the relevant class. Class evolution can hence be seen as a case of stable transfection (or transduction if `\RequirePackage` is involved).

5.6 Stylophages

What if styles could infect other styles instead of just documents? This is in fact very common practice, as `\RequirePackage` can be used in styles as well as in classes. Again surveying TeX Live 2009 reveals that 1111 out of 2462 (45%) available styles are themselves infected, by 2 other styles on average.

This kind of behavior has been observed in biology as well, although only very recently. The first virus capable of infecting another virus, called the “virophage” in reference to bacteriophages, has been discovered by Didier Raoult's team in 2008 [12]. This virus infects the *mimivirus* (see section 4.2 on page 164) and uses its machinery in lieu of a cell's one in order to replicate itself.

6 Infection types

In the previous section, we have looked at similarities in the way cells or documents can be infected. In this section, we will analyze the *effects* of infection, and draw some analogies again. In other words, we will now consider examples of what infection *does* rather than how it *spreads*.

6.1 Standalone

Perhaps the simplest (and most harmless) form of style infection is by what we call *standalone* styles. Standalone styles provide macros that do not modify, interact, or even require anything particular from a document class or other styles. They just use the usual TeX machinery to add new features, completely orthogonal to the rest; in other words, geneTeX material that you are free to use... or not. An example of this is the `clock` package which provides macros for drawing clocks of all sorts of visual appearances.

This situation is similar to that of viruses infecting non-permissive cells (in which they can't replicate themselves), but in which however their genetic material may remain in the form of free *episomes*, that is, not integrated into the cell's genome. Just like the information necessary to draw a clock is here but is really independent from the rest, (part of) the genetic information of the virus is also here, but does not interact with the cell's original genome. The genes brought by the virus may or may not be expressed depending on environmental conditions, just like you may choose to actually draw a clock or not in your document. If you don't, the presence of this exogenic material simply has no effect.

6.2 Prostyles

Instead of standing apart from the cell's DNA, viruses can have their genetic material incorporated into that of their host, in which case they are called *proviruses*. Because of this integration, proviruses passively replicate as part of their host's replication process, although just as for standalone viruses, infection can either remain latent or become active.

Because of this integration with the cell's original genome, the potential effects of a provirus are extremely wide: they can amplify, inhibit or even modify the different functions of their host. They can either have very little pathogenic effect, like AAV (Adeno-associated virus), or cause extremely serious diseases, like HIV.

The vast majority of LaTeX styles, including those mentioned in the following sections, qualify as *prostyles* in the sense that they incorporate their programmatic material into the existing one, instead of just contributing something new and orthogonal. Most styles in LaTeX indeed exist to enhance or modify an existing functionality.

The following very common programming idiom makes a style a prostyle:

```
\let\@oldfoo\foo
\def\foo{... \@oldfoo ...}
```

What this does is essentially to incorporate some new geneTeX material into the existing definition for `\foo`, thereby modifying its associated function. In a similar way, every time you `\renewcommand` something, you are creating a prostyle. The examples are innumerable in LaTeX. ***FiNK*** [18], for instance, is a prostyle because it modifies the behavior of `\InputIfFileExists`; `hyperref` [10] is another notable prostyle given the amount of semantic changes it inflicts on existing commands, *etc.*

In fact, the use of a prostyle as a means to alter the original programmatic material of a document looks very much like the use of a virus to *willingly* incorporate new genes into an organism. Such organisms are called GMO/GEO (Genetically Modified/Engineered Organisms). So we could say that using a prostyle in a LaTeX document makes it a GMD/GED (GeneTeXally Modified/Engineered Document).

6.3 Satellite

One (perhaps the most) important source of style proliferation in LaTeX is what we call *satellite* styles. A satellite style exists to amplify or extend the functionalities provided by another style. Examples of satellite styles are `DoX`, which extends the `doc` package, `graphicx` which extends `graphics` and `xkeyval` which extends `keyval`.

Satellite styles typically depend on the presence of their respective "sub-style" to work properly, because they build on top of them. They cannot work properly on their own.

This behavior exists in biology as well, as some viruses are considered to be satellites of others. For instance, the so-called *Delta* virus, or Hepatitis D Virus (HDV) is considered to be a satellite of HBV, Hepatitis B. The HDV cannot propagate without the presence of the HBV, but when both are present, the risk for complication, or the lethal rate increases. Other examples would be most of the avian sarcoma viruses, which require the help of a non-defective (see next section) leukemia virus.

Biologists distinguish between *co-infection*, when a patient is infected by both viruses at the same time, and *super-infection* when the two infections happen one after the other. In the LaTeX world, super-infection is or should be nonexistent because it would mean that a document author is required to explicitly `\usepackage` both styles, one after the other. A better practice for a satellite style is to `\RequirePackage` the style it depends on. This way, a document directly suffers from co-infection. In fact, satellite styles are almost always stylophages (see section 5.6 on the facing page).

6.4 Defective

Satellite viruses are in fact a sub-category of so-called *defective* viruses. A defective virus is a virus that lacks a complete genome and hence depends on another virus to provide the missing genetic function. While defective viruses are defective *by mutation*, defective styles are defective *by design*. Computer science does not usually like to reinvent the wheel; reusability is a key paradigm in software engineering. So when a package author creates a new satellite style, he or she usually avoids replicating the base functionality with "cut-and-paste", but relies on co-infection by `\RequirePackage`'ing the underlying functionality. The resulting style is indeed defective, but on purpose.

6.5 Host-dependent

When a virus is defective, it can rely on another virus to provide the missing genetic function, or it can rely on the host cell. Such viruses are not called satellite anymore, but are said to be *host-dependent*.

Host-dependent styles exist in the LaTeX world. Such styles would only work with a specific document class to provide the missing geneTeX functions. The example which comes to mind immediately is that of Beamer themes. Beamer is a class for writing slides, the appearance of which can be customized. Beamer themes are styles defining a specific set of morphological traits for slides, and are obviously specific to Beamer documents.

6.6 Cheaters

The defective styles presented so far work in a spirit of cooperation with their "helper": Beamer themes exist to enrich Beamer documents, `xkeyval` exists to improve `keyval`, *etc.* In a way, this is also the case for the HDV and HBV viruses which "work" together in the common and unfortunate goal of spreading hepatitis.

But what if some defective styles were in fact egoistically "stealing" functionality from their helper, and diverting it for a totally different purpose? What if, in other words, defective styles were working in a spirit of *competition* instead of *cooperation*?

In a very amusing way, there is at least one style that we know of which already cheats on itself: the `verbatim` package. This style provides an environment for outputting text as is, but also provides a `comment` environment which simply discards all its contents. Although comments are completely unrelated to verbatim text, the implementation of the `comment` environment *steals* the basic functionality used to produce verbatim text: the ability to have TeX read text without interpreting any commands or special characters.

The soon-to-come `lstblocks` package will do exactly the same. In order to properly integrate inline Listings and Beamer blocks (which cannot be nested out of the box), we use a trick based on `verbatim`: the inline text is first output to a file, and the file is later re-input by `\lstinputlisting`.[2] In other words, we steal functionality from `verbatim` in order to do something which is the exact opposite of what it is originally meant for: typesetting some text with heavy fontification instead of as is.

In a very puzzling way, cheaters also exist in the world of viruses. Experimental studies even show that cheaters often win the competition and overwhelm the cooperating ones [16]. An example of a cheating virus is the *umbravirus* [8, 15] that steals the coat protein of another virus, the *luteovirus*, in order to spread to other plants.

[2] The technical details of this process are explained in the following blog entry: `http://lrde.epita.fr/~didier/sciblog/index.php?entry=entry080604-120459`

7 Conflicts, diseases and cures

There is an angle from which the analogy between LaTeX and biology could be regarded as somewhat shaky: viruses are usually studied and well known for their pathogenic effects, while styles are supposed to do some good.

7.1 Good or bad, or both

The replication of a virus usually entails cellular *lysis* (the destruction of the host cell's plasma membrane) in order to disseminate new *virions* (complete virus particles) in the environment. The pathogenic potential of a virus (in other words its ability to lead to a disease) is described in terms of *virulence* and depends on the success of its replication. However, from a unicellular point of view, the presence of a single active viral entity is lethal to the cell. We, on the other hand, are more interested in the *benefits* of style infection: LaTeX styles are normally meant to be non-virulent and provide additional functionality: "useful viruses" in some way.

It turns out, however, that our analogy is not so shaky after all: a positive vision of viruses does exist, although it is still quite young. Only recently biologists have started to acknowledge the positive role of viruses as important factors of evolution or even as therapeutic tools. In fact, most viruses that we live with every day are harmless. Consequently, it appears that viruses, just like styles, have both a Dr. Jekyll and a Mr. Hyde face: viruses play a crucial role in evolution but they can cause diseases, while styles give you more power but can cause conflicts.

7.2 Conflicts and diseases

Just as proviruses can cause serious diseases, prostyles can cause the compilation to break, especially if some bad interaction happens between their geneTeX material and that of other styles or of the document class. Basically, a LaTeX document can be in three states: healthy, ill-formed or dead. An ill-formed document compiles successfully but displays incorrectly. A dead document is a document which could not compile, so TeX aborted. In a similar way, a cell can live normally, function improperly or be dead.

A very systematic and rather famous way of getting a living yet ill-formed LaTeX document is to infect it with the `a4wide` style. If your document

This is a test to see if the problems which seem to, for example?]FiXme Note: what does [this] do, for example? be caused by using square brackets in marginal fixme notes, even in a minimal document...

Figure 3: An ill-formed FiXme note

uses the `twoside` option, then the infection will render this option inoperative (odd and even pages get the same geometry). Another shameful example is that of FiXme [19] which seems to have problems typesetting square brackets in marginal notes, as depicted in figure 3. The next version, still under development, does not seem to suffer from this problem. The reason for this is currently unknown, but probably involves some kind of geneTeX mutation in the codebase....

The ways to break compilation because of style infection are too numerous to be listed here. The fact that you are reading this very paper can be considered a miracle in itself, given that the source involves both the `hyperref` and the `varioref` packages. Suffice to quote the README file from the `hyperref` distribution:

> There are too many problems with varioref. Nobody has time to sort them out. Therefore this package is now unsupported.

Note that viruses or styles are not a requirement for a cell to malfunction or die, or a document to be ill-formed or uncompilable. A cell can malfunction for many other reasons, including DNA mutation because of external conditions, *etc.* A document can turn out ill-formed or even die because there are bugs in its programming.

7.3 Cures

Just as in biology, facing the risk of possibly lethal diseases to documents entails the search for cures. In biology, viral infections are basically treated with either prevention, vaccines or antiviral agents.

7.3.1 Prevention

No cure is needed if no infection is present. In other words, if you don't want to get sick, then just don't get a disease. Knowing the risks in advance is hence the key to prevention, and this is probably much easier to do in LaTeX than in biology. In the LaTeX world, prevention will mostly be accomplished by *documentation.*

An example of prevention against a non-lethal disease is given in section 3.3 of the FiXme documentation: FiXme explicitly supports the standard LaTeX classes plus their KOMA-Script replacements for typesetting the list of FiXme's. For any other class, the `article` layout will be used, which will probably lead to an ill-formed list. In other words, some classes are known to be *immune* to FiXme infection, and for other cases, you know the risks.

Another example of prevention against lethal sickness is the quote from `hyperref`'s README file about `varioref` presented in section 7.2 on the facing page. What it really says is:

> *You are infected by Hyperref. If you want to live, don't be infected by Varioref as well: just don't use it.*

The other interesting aspect in this particular example is that the concern is about *super-infection* rather than just infection, since it deals with the presence of two styles simultaneously. This is a bit like saying:

> *You have got HBV. This is already serious enough. Just don't get HDV on top of that.*

7.3.2 Adaptive immune systems

Mentioning adaptive immune systems here is a bit borderline as it involves complex, multi-cellular organisms. However, there are still a number of interesting common patterns to mention.

Adaptive immunity (contrary to *innate* immunity) is the process by which an organism *acquires* defenses against a pathogen such as a virus. Immunological memory, materialized by the presence of so-called *B-* and *T-cells*, contains some kind of history of previously encountered infections, and how to fight them. We are principally interested in *active* immunological memory, which is a long-term memory of immunological responses. Such memory can be acquired naturally after a real infection (such as with measles or mumps), or artificially from vaccination. Vaccination typically consists of faking a real infection with a non-virulent form of a virus in order to trigger an immune system response.

What is interesting here is the pattern by which an organism suffers from an infection, learns to fight it, and then memorizes the "counter-measures" in order to be prepared for future attacks. This pattern exists in LaTeX and can be described by the following steps:

1. John writes a document of class `class`, but notices that when he uses the style `style`, compilation breaks.

2. John sends a bug report somewhere (such as `comp.text.tex`, the author of `class`, or more probably the author of `style`).

3. Some time later (for some definition of "some"), a new version of either `class` or `style` is released and everything works smoothly.

Now, depending on whether `class` or `style` mutates in order to circumvent the infection, we find ourselves in two very different situations.

When a new version of the *class* is released, we fall into the case of acquired immunity, as described above: the class remembers the infection and will know how to fight it in the future. The principal LaTeX organelle to implement acquired immunity is the `\@ifpackageloaded` macro. This macro tests for the presence of a known infection and lets you plug in the appropriate counter-measures. In the TeX Live 2009 distribution, only 37 out of 271 classes (13%) seem to be equipped with an active immune system, against 2 styles in average. The `memoir` class has the strongest active immune system, with antigens against 5 styles. `revtex4` has an interesting mechanism by which some incompatible styles are known although no immunity is provided. Instead, the class decides to commit suicide rather than fight the infection, not unlike what a too brutal viral agent would do: eradicate the virus but damage the cell, or even kill it in the process.

When a new version of the *style* is released, however, the situation looks more like a case of *adaptation* of the style to a new class of documents. Here, the analogy is more that of a virus acquiring the ability to infect new types of cells. Biologists have a term for that: *viral tropism.* The principal LaTeX organelle to increase style tropism is the `\@ifclassloaded` macro. This macro tests for the kind of document you are trying to infect, and lets you plug in the appropriate adaptation routines. As an example, the `sectsty` and FiXme packages have explicit adaptation routines for 8 classes known in advance (including the standard ones and their KOMA-Script replacement). Note however that when a style does not make use of `\@ifclassloaded`, this does not mean that it has a very low viral tropism, but usually the opposite: it means that its geneTeX material is universal enough that it doesn't need to know the type of its host document explicitly.

7.3.3 Acquired or innate?

A serious risk in any game of analogies is to miss their limits. We must acknowledge a very important divergence between biological and LaTeX adaptive immune systems as described in the previous section.

In a multi-cellular organism, the adaptive immune system builds a *functional* response to an aggression, for instance by selecting special kinds of lymphocytes. The problem is that the active immunological memory is specific to the individual. In other words, vaccinating a mother does not provide immunity to her children. The descendants need to be vaccinated as well. On the other hand, once a class has mutated in order to provide the proper immunological response to a style infection, *all* documents of that particular class will implicitly be vaccinated (including the original one). In fact, the immune response is now encoded into the original geneTeX material of every new document of that particular class, so that it essentially becomes an *innate* trait.

The interesting question that arises here is hence the following: can acquired traits become innate? More specifically in our case: is it possible that a response to a virus ends up encoded in a cell's genetic material? We currently don't have a firm answer to this question. A potential path for further study would be to investigate the field of *epigenetics* at the risk of opening the heated debate around the central dogma of molecular biology. Recent studies show for example that some plants are able to alter their genetic material in response to environmental stress (*e.g.* viral attacks) and that these alterations can be transmitted to the next generations [9].

7.3.4 Antistyle agents

In the LaTeX world, the fight between styles is even more violent than the fight between styles and classes. A particularly striking example is again that of section 6 of the `hyperref` README file, which is 8 pages long and describes incompatibilities (and cures) between `hyperref` itself and around 40 other packages. Sometimes, the cure simply consists in making sure that your document is infected in a specific order. Some other times, additional hacks are needed to make things work, for instance in the case of `bibentry`:

```
\makeatletter
\let\saved@bibitem\@bibitem
\makeatother
```

And then later:

```
\begingroup
\makeatletter
\let\@bibitem\saved@bibitem
\nobibliography{database}
\endgroup
```

This kind of very specialized and local "cure" can be compared to so-called *antiviral agents*: specific molecules that treat specific kinds of infections. Contrary to the case of an immune system, the organism is not prepared in advance to fight the infection. Instead it is provided with an external compound once infected. The compound in question does not provide any immunological memory either.

Antistyle agents such as those described in the `hyperref` README file behave exactly like that. The document is not immune to the infection, as the class is not prepared geneTEXally for it. Instead, every document needs an inoculation of the antistyle agent in order to fight the infection.

7.3.5 Curative infections

The use of `\@ifpackageloaded` is not restricted to class files. Style files can use it as well, hereby anticipating the co-existence of multiple infections within the same document. A quick survey of the TEX Live 2009 distribution shows that only 8% of the available styles make use of this feature. Some of them, however, rely on it quite heavily. For instance, `minitoc` knows about almost 30 other styles in advance, which helps avoid conflicts.

In most cases, this kind of style/style interaction exists for altruistic concerns, which is somehow the opposite approach to what `hyperref` does with its README file: a document is first infected with a style which could potentially cause problems in the case of super-infection, but this style also provides some geneTEX material in order to protect the whole document against those super-infections. In other words, one infection helps in fighting another.

Another previously mentioned example is that of the soon-to-come `lstblocks` package: Beamer blocks and Listings inline environments don't interact well with each other, and the purpose of `lstblocks` is to fix that. This is another form of "curative infection" although there is one fundamental difference with the previous one. In the first case, the style provided cures against potential diseases it could cause *itself*, in the presence of another style. In the case of `lstblocks`, its sole purpose is to cure a conflict caused not by itself, but by the conflicting presence of two other (and unrelated) infections.

A similar pattern of curative infection exists in biology: there are situations in which a virus (or at least part of its genome) helps fighting another. For instance, some mice are naturally protected against variants of a virus called *Friend*. The gene responsible for this protection, named *Fv1*, was identified in 1992 to be of endogenic retroviral origin. In other words, some genetic material from a virus helps fighting against another.

8 Breaking news

To end this paper on a positive note, we would like to proudly announce the recent discovery of the first *oncogenic* style ever. This style appears to be extremely virulent, as we were able to witness a mutation just one day after its discovery.

```
1 \ProvidesPackage{oncogenic}
2  [2010/07/28 v1.0 TUG Virus]
3 \expandafter\let\csname
4   ver@oncogenic.sty\endcsname\relax
5 \RequirePackage{oncogenic}
```

Listing 1: TUG virus strain #1

```
1 \ProvidesPackage{oncogenic}
2  [2010/07/29 v2.0 TUG Virus]
3 \def\@ifl@aded#1#2{%
4   \expandafter\@secondoftwo}
5 \RequirePackage{oncogenic}
```

Listing 2: TUG virus strain #2

The first strain of the virus was discovered on July 28th, 2010, at the TUG 2010 Conference, Sir Francis Drake Hotel, San Francisco, CA, USA, and is shown in listing 1. As you can see in line 2, this style makes the document forget it has been infected by removing the definition of `\ver@oncogenic.sty`, and then replicates by requiring itself. This results in the death of the document by resource exhaustion:

```
! TeX capacity exceeded, sorry
  [input stack size=5000].
<to be read again>
   \ver@oncogenic.sty
l.1 .../06/28 v1.0 TUG Virus]

No pages of output.
Transcript written on cancer.log.
zsh: exit 1     latex cancer.ltx
```

Just one day after its original discovery, we were able to isolate a new strain of the style, depicted in listing 2. This strain is much more aggressive and has a much more widespread effect. Line 2 exhibits a geneTEX mutation of a macro from the original document's genome: `\@ifl@aded`. This macro is responsible for controlling whether a file has been loaded before, and avoids loading it multiple times if it so happens. As such, it regulates the growth of the document, and hence qualifies as a *proto-oncogene*.

The mutation caused by the style makes this macro unconditionally load the file in question, resulting in the same proliferation of the style as before. What the style does is in effect turn the proto-oncogene into a full blown *oncogene* [14] that may affect the whole document.

So far, we have been able to synthesize an `\@ntibody` that will suppress the effect of line 3.

However, the oncogene in line 2 still remains latent in the document's genome, and might be expressed again if a circular chain of file requirements ever happens, in which case the style would become virulent again. We are confident that a definitive cure will be found in the months to come.

9 Conclusion

Drawing bridges between apparently unrelated disciplines is always interesting and fun to do, if not for the potential practical applications, at least for the sake of the mental exercise. This kind of cross-disciplinary thinking, however, has proven to be concretely useful in the past. For instance, we know the impact of *Design Patterns*, originating from Architecture [1], on the world of Computer Science [5, 3].

What we have done with this work is essentially exhibit a set of *behavioral* patterns that seem to equally rule the interactions between components of different domains. In doing so, we hope to have contributed to a better understanding of the world we live in, whether biological or digital. It is also very probable that we have only scratched the surface of this idea, and that many other patterns are left for us to discover.

Finally, we suspect that bridges with the same essence can be extended to other macro systems, such as `m4`, and beyond macro languages, programming languages which provide for deep intercession capabilities, such as the Lisp family of languages [2].

10 Acknowledgments

Alain Verna provided the original leukocyte photography in figure 1. Mireille Verna provided valuable comments on the ideas behind this work, as well as proofreading of this article.

References

[1] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

[2] ANSI. American National Standard: Programming Language — Common Lisp. ANSI X3.226:1994 (R1999), 1994.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture.* Wiley, 1996.

[4] François Jacob. La vie. In Yves Michaud, editor, *Qu'est-ce que la vie*, Université de tous les savoirs. Odile Jacob, 2000.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

[6] D.E. Knuth. *The TEXbook.* Addison-Wesley, 1989 (reprinted with corrections).

[7] Lynn Margulis. *Origin of Eukaryotic Cells.* Ignatius Press, San Francisco, 1970.

[8] R. Matthews. *Plant Virology.* Academic Press, 1991.

[9] J. Molinier, G. Ries, C. Zipfel, and B. Hohn. Transgeneration memory of stress in plants. *Nature*, 442:1046–1049, 2006.

[10] Heiko Oberdiek. PDF information and navigation elements with `hyperref`, pdfTEX, and `thumbpdf`. In *EuroTEX*, 1999.

[11] Ali Saïb. Les virus, inertes ou vivants ? *Pour la Science*, December 2006.

[12] Bernard La Scola, Christelle Desnues, Isabelle Pagnier, Catherine Robert, Lina Barrassi, Ghislain Fournous, Michèle Merchat, Marie Suzan-Monti, Patrick Forterre, Eugene Koonin, and Didier Raoult. The virophage, a unique parasite of the giant Mimivirus. *Nature*, August 2008.

[13] Wendell Stanley. Isolation of a crystalline protein possessing the properties of tobacco-mosaic virus. *Science*, 81:644–645, 1935.

[14] D. Stehelin, H.E. Varmus, J.M. Bishop, and P.K. Vogt. DNA related to the transforming gene(s) of avian sarcoma viruses is present in normal avian DNA. *Nature*, 260:170–173, 1976.

[15] Michael E. Taliansky and David J. Robinson. Molecular biology of umbraviruses: phantom warriors. *Journal of General Virology*, 84:1951–1960, 2003.

[16] Paul E. Turner. Cheating viruses and game theory. *American Scientist*, 93:428–435, September–October 2005.

[17] Didier Verna. The DoX package. `http://www.lrde.epita.fr/~didier/software/latex.php#dox`.

[18] Didier Verna. The ***FiNK*** package. `http://www.lrde.epita.fr/~didier/software/latex.php#fink`.

[19] Didier Verna. The FiXme package. `http://www.lrde.epita.fr/~didier/software/latex.php#fixme`.

[20] Didier Verna. The QCM package. `http://www.lrde.epita.fr/~didier/software/latex.php#qcm`.

[21] Didier Verna. CV formatting with ***CurVe***. *TUGboat*, 22(4):361–364, December 2001.

[22] Didier Verna. LATEX curricula vitae with the ***CurVe*** class. *The PracTEX Journal*, (3), August 2006.

[23] R. H. Whittaker. New concepts of kingdoms of organisms. *Science*, 163(3863):150–160, 1969.

◇ Didier Verna
EPITA / LRDE
14-16 rue Voltaire
94276 Le Kremlin-Bicêtre Cedex
France
`didier (at) lrde dot epita dot fr`
`http://www.lrde.epita.fr/~didier`

TeX Live 2010 news

Karl Berry

Abstract

Notable changes in the TeX Live 2010 release.

1 Automatic EPS conversion for pdf(LA)TeX

In TeX Live 2010, the most visible change is that pdf(LA)TeX now *automatically* converts a requested Encapsulated PostScript (EPS) file to PDF, via the `epstopdf` package, when and if the LaTeX configuration file `graphics.cfg` is loaded, and PDF is being output. The default options are intended to eliminate any chance of hand-created PDF files being overwritten, but you can also prevent `epstopdf` from being loaded at all by putting `\newcommand {\DoNotLoadEpstopdf}{}` (or `\def...`) before the `\documentclass` declaration. For details, see the package documentation for epstopdf (`http://ctan.org/pkg/epstopdf-pkg`).

A related important change is that execution of a very few external commands, via the `\write18` feature, is now enabled by default — for example, `epstopdf`, `makeindex`, and `bibtex`. The exact list of commands is defined in the `texmf.cnf` file. If you want to disallow all external command execution in your installation, you can deselect this option in the TeX Live installer, or override the value in the root `texmf.cnf` (`.../2010/texmf.cnf`) after installation. (Do not change `.../2010/texmf/web2c/texmf.cnf`, since it can be overwritten on updates.)

The above changes were announced for TL 2009, but disabled before the release was made. This year we sincerely hope and believe the bugs have been worked out.

2 PDF 1.5 output by default

The default version for PDF output is now 1.5, enabling more compression. This applies to all the TeX engines when used to produce PDF and to dvipdfmx.

Loading the `pdf14` LaTeX package changes back to PDF 1.4, or setting `\pdfminorversion=4` (in your document or macros).

3 XeTeX and margin kerning

XeTeX now supports margin kerning along the same lines as pdfTeX. Font expansion is not yet supported; perhaps next year.

4 New programs

New programs included:

- the pTeX engine for typesetting Japanese;
- BibTeXU, a Unicode-enabled BibTeX;
- `chktex`, for syntax checking (LA)TeX documents (`http://baruch.ev-en.org/proj/chktex`);
- `dvisvgm`, a DVI-to-SVG translator (`http://dvisvgm.sourceforge.net`).

5 No longer live

The TeX Live incarnation on the TeX Collection DVD (`http://tug.org/texcollections`) can no longer be run live. A single DVD no longer has enough room for all the software we want to include. One benefit is that installation from a physical DVD should be much faster.

6 Backups of updated packages

By default, `tlmgr` now saves one backup of each package updated, so broken packages updates can be easily reverted with `tlmgr restore`. If you don't have enough disk space for backups, and want to do post-installation updates, run:

```
tlmgr option autobackup 0
```

7 BibTeX and Makeindex output safer

By default, BibTeX and Makeindex will now, like TeX itself, refuse to write their output files to an arbitrary directory. This is so they could be enabled for use by the restricted `\write18` mentioned above. To change this, the `TEXMFOUTPUT` environment variable can be set, or the `openout_any` setting changed.

8 In summary

Thanks to all the many people involved in this and past releases. See the documentation for acknowledgements, as well as installation and usage information — `http://tug.org/texlive/doc.html`.

All releases of TeX Live, along with ancillary material such as CD labels, are available at `ftp://tug.org/historic/systems/texlive`.

TeX Live and MiKTeX account for a substantial portion of the total traffic to CTAN. So more mirrors are always welcome; if you can help, please see `http://ctan.org/mirroring.html`, and thanks.

◇ Karl Berry
`http://tug.org/texlive`

LuaTEX 0.60: An overview of changes

Taco Hoekwater and Hartmut Henkel

Abstract

TEX Live 2010 will contain LuaTEX 0.60. This article gives an overview of the changes between this version and the version in last year's TEX Live.

Highlights of this release: CWEB code base, dynamic loading of lua modules, various font subsystem improvements including support for Apple `.dfont` font collection files, braced input file names, extended PDF Lua table, and access to the line breaking algorithm from Lua code.

1 General changes

Some of the changes can be organised into sections, but not all. So first, here are the changes that are more or less standalone.

- Many of the source files have been converted into CWEB. Early versions of LuaTEX were based on Pascal WEB, but by 0.40 all code had been hand-converted to C. The literate programming comments were kept, and the relevant sources have now been converted back into CWEB, reinstating the literate documentation.

  This change does not make LuaTEX a literate program in the traditional sense because the typical C source code layout with pairs of header & implementation files has been kept and no code reshuffling takes place. But it does mean that it is much easier to keep the source documentation up-to-date, and it is possible to create nicely typeset program listings with indices.
- There are now source repository revision numbers in the banner again, which is a useful thing to have while tracking down bugs. For example, the LuaTEX binary being used to write this article starts up with (except all on one line):

  ```
  This is LuaTeX,
    Version beta-0.60.1-2010042817 (rev 3659)
  ```
- The horizontal nodes that are added during line breaking now inherit the attributes from the nodes inside the created line. Previously, these nodes (`\leftskip` and `\rightskip` in particular) inherited the attributes in effect at the end of the (partial) paragraph because that is where line breaking takes place.
- All Lua errors now report file and line numbers to aid in debugging, even if the error happens inside a callback.
- LuaTEX can now use the embedded Kpathsea library to find Lua `require()` files, and will do so by default if the Kpathsea library is enabled by the format (as is the case in plain LuaTEX and the various LuaLATEX formats).
- The print precision for small numbers in Lua code (the return value of `tostring()`) has been improved.
- Of course there were lots of code cleanups and improvements to the reference manual.

2 Embedded libraries and other third-party inclusions

The following are changes to third-party code that for the most part should not need much explanation.

- MetaPost is now at version 1.211.
- Libpng is now at version 1.2.40.
- New SyncTEX code is imported from TEX Live.
- The Lua source file from the `luamd5` library (which provides the `md5.hexsuma` function) is now embedded in the executable. Previously, this file was missing completely.
- The Lua co-routine patch (`coco`) is now disabled on `powerpc-linux` because of crashes on that platform due to a bad upstream implementation.

2.1 Dynamic loading of lua modules

LuaTEX now has support for dynamic loading of external compiled Lua libraries.

As with other `require()` files, LuaTEX can and will use Kpathsea if the format allows it to do so. For this purpose, Kpathsea has been extended with a new file type: `clua`. The associated `texmf.cnf` variable is defined like this by default:

```
CLUAINPUTS = \
  .:$SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

which means that if your LuaTEX binary lives in

```
/opt/tex/texmf-linux-64/bin/
```

then your compiled Lua modules should go into the local directory, or in a tree below

```
/opt/tex/texmf-linux-64/bin/lib/lua
```

Be warned that not all available Lua modules will work. LuaTEX is a command line program, and on some platforms that makes it nearly impossible to use GUI-based extensions.

3 Font related

Lots of small changes have taken place in the font processing.

- The backend message

  ```
  cannot open Type 1 font file for reading
  ```

  now reports the name of the Type1 font file it was looking for.

- It is no longer possible for fonts from included PDF files to be replaced by or merged with the document fonts of the enveloping PDF.
- Support for Type 3 `.pgc` files has been removed. This is just for the `.pgc` format invented by Hàn Thế Thành; bitmapped PK files still work.
- For TrueType font collections (`.ttc` files), the used subfont name and its index id are now printed to the terminal, and if the backend cannot find the font in the `.ttc`, the run is aborted.
- It is now possible to use Apple `.dfont` font collection files. Unfortunately, in Snow Leopard (a.k.a. Mac OS X 10.6) Apple switched to a `.ttc` format that is not quite compatible with the Microsoft version of `.ttc`. As a result, the system fonts from Snow Leopard cannot be used in LuaTEX 0.60.
- Faster loading of large fonts via the `fontloader` library, and faster inclusion for subsetting in the backend.
- Two new entries in the `MathConstants` table have been added. Suppose the Lua math font loading code produces a Lua table named `f`, then in that table, you can set

  ```
  f.MathConstants.FractionDelimiterSize
  f.MathConstants.
          FractionDelimiterDisplayStyleSize
  ```

  These new fields allow proper setting of the size parameters for LuaTEX's `...withdelims` math primitives, for which there is no ready replacement in the OpenType MATH table.
- Artificially slanted or extended fonts now work via the PDF text matrix so that this also works for non-Type 1 fonts. In other words: the Lua `f.slant` and `f.extend` font keys are now obeyed in all cases.
- Another new key is allowed: `f.psname`. When set, this value should be the original PostScript font name of the font. In the PDF generation backend, fonts inside `.dfont` and `.ttc` collections are fetched from the archive using this field, so in those cases the key is required.
- A related change to the font name discovery used by the backend for storage into the PDF file structure: now it tries `f.psname` first, as that is much less likely to contain spaces than `f.fontname` (which is the field that 0.40 used). If there is no `f.psname`, it falls back to the old behaviour.
- Finally, Lua-loaded fonts now support the key `f.nomath` to speed up loading the Lua table in the normal case of fonts that do not provide OpenType MATH data.

4 'TEX'-side extensions and changes

LuaTEX is not actually TEX even though it uses an input language that is very similar, hence the quotes in this section's title. Some of the following items are new LuaTEX extensions, others are adjustments to pre-existing pdfTEX or Aleph functionality.

- The primitives `\input` and `\openin` now accept braced file names, removing the need for double quote escapes in case of files with spaces in their name.
- The `\endlinechar` can now be set to any value between 0 and 127.
- The new primitives `\aligntab` and `\alignmark` are aliases for the characters with the category codes of `&` and `#` in alignments, respectively.
- `\latelua` is now allowed inside leaders. To be used with care, because the Lua code will be executed once for each generated leader item.
- The new primitive `\gleaders` provides 'globally aligned' leaders. These leaders are aligned on one side of the main output box instead of to the side of the immediately enclosing box.
- From now on LuaTEX handles only 4 direction specifiers:
  - `TLT` (latin),
  - `TRT` (arabic),
  - `RTT` (cjk), and
  - `LTL` (mongolian).

  Other direction specifiers generate an error.
- The `\pdfcompresslevel` is now effectively fixed as soon as any output to the PDF file has occurred.
- `\pdfobj` has gained an extra optional keyword: `uncompressed`. This forces the object to be written to the PDF in plain text, which is needed for certain objects containing metadata.
- Two new token lists are provided: `\pdfxformattr` and `\pdfxformresources`, as an alternative to `\pdfxform` keywords.
- The new syntax

  `\pdfrefxform` [`width` ⟨*dimen*⟩]
  [`height` ⟨*dimen*⟩] [`depth` ⟨*dimen*⟩] ⟨*formref*⟩

  scales a single form object using similar principle as with `\pdfximage`: depth alone doesn't scale, it shifts vertically.
- Similarly,

  `\pdfrefximage` [`width` ⟨*dimen*⟩]
  [`height` ⟨*dimen*⟩] [`depth` ⟨*dimen*⟩] ⟨*imageref*⟩

  overrules settings from `\pdfximage` for this image only.

- The following obsolete pdfTEX primitives have been removed:
  - `\pdfoptionalwaysusepdfpagebox`
  - `\pdfoptionpdfinclusionerrorlevel`
  - `\pdfforcepagebox`
  - `\pdfmovechars`

  These were already deprecated in pdfTEX itself.

5 Lua table extensions

In most of the Lua tables that LuaTEX provides, only small changes have taken place, so they do not deserve their own subsections.

- A new callback, `process_output_buffer`, allows post-processing of `\write` text to a file.
- The callbacks `hpack_filter`, `vpack_filter` and `pre_output_filter` pass on an extra string argument for the current direction.
- `fontloader.open()` previously cleared some of the font name strings during load that it should not do.
- The new function `font.id("tenrm")` returns the internal id number for that font. It takes a bare control sequence name as argument.
- The `os.name` variable now knows about `cygwin` and `kfreebsd`.
- `lfs.readlink("file")` returns the content of a symbolic link (Unix only). This extension is intended for use in `texlua` scripts.
- `lfs.shortname("file")` returns the short (FAT) name of a file (Windows only). This extension is intended for use in `texlua` scripts.
- `kpse.version()` returns the Kpathsea version string.
- `kpse.lookup(...)` offers a search interface similar to the kpsewhich program, an example call looks like this:

```
kpse.set_program_name('luatex')
print(kpse.lookup('plain.tex',
                  { ["format"] = "tex",
                    ["all"] = true,
                    ["must-exist"] = true }))
```

5.1 The node table

In the verbatim code below, `n` stands for a userdata node object.

- `node.vpack(n)` packs a list into a vlist node, like `\vbox`.
- `node.protrusion_skippable(n)` returns `true` if this node can be skipped for the purpose of protrusion discovery. This is useful if you want to (re)calculate protrusion in pure Lua.
- `node.dimensions(n)` returns the natural width, height and depth of a (horizontal) node list.
- `node.tail(n)` returns the tail node of a node list.
- Each glyph node now has three new virtual read-only fields: `width`, `height`, and `depth`. The values are the number of scaled points.
- `glue_spec` nodes now have an extra boolean read-only field: `writable`.

  Some glue specifications can be altered directly, but certain key glue specifications are shared among many nodes. Altering the values of those is prohibited because it would have unpredictable side-effects. For those cases, a copy must be made and assigned to the parent node.
- hlist nodes now have a subtype to distinguish between hlists generated by the paragraph breaking, explicit `\hbox` commands, and other sources.
- `node.copy_list(n)` now allows a second argument. This argument can be used to copy only part of a node list.
- `node.hpack(n)` now accepts `cal_expand_ratio` and `subst_ex_font` modifiers. This feature helps the implementation of font expansion in a pure Lua paragraph breaking code.
- `node.hpack(n)` and `node.vpack(n)` now also return the 'badness' of the created box, and accept an optional direction argument.

5.2 The pdf table

- The new functions `pdf.mapfile("...")` and `pdf.mapline("...")` are aliases for the corresponding pdfTEX primitives.
- `pdf.registerannot()` reserves a PDF object number and returns it.
- The functions `pdf.obj()`, `pdf.immediateobj()`, and `pdf.reserveobj()` are similar to the corresponding pdfTEX primitives. Full syntax details in the LuaTEX reference manual.
- New read-write string keys:
  - `pdf.catalog` in the Catalog dictionary.
  - `pdf.info` in the Info dictionary.
  - `pdf.names` in the Names dictionary referenced by the Catalog object.
  - `pdf.trailer` in the Trailer dictionary.
  - `pdf.pageattributes` in the Page dictionary.
  - `pdf.pageresources` in the Resources dictionary referenced by the Page object.
  - `pdf.pagesattributes` in the Pages dictionary.

5.3 The `tex` table

Finally, there are some extensions to the `tex` table that are worth mentioning.

- `tex.badness(f,s)` interfaces to the 'badness' internal function. (By accident, this disables access to the `\badness` internal parameter. This will be corrected in a future LuaTeX version.)
- `tex.sp("1in")` converts Lua-style string units to scaled points.
- `tex.tprint(...,...)` is like a sequence of `tex.sprint(...)` calls.
- `tex.shipout(n)` ships out a constructed box.
- `tex.nest[]` and `tex.nest.ptr` together allow read-write access to the semantic nest (mode nesting). For example, this prints the equivalent of `\prevdepth` at the current mode nesting level:

  ```
  print (tex.nest[tex.nest.ptr].prevdepth)
  ```

  `tex.nest.ptr` is the current level, and lower numbers are enclosing modes.

  Each of the items in the `tex.nest` array represents a mode nesting level and has a set of virtual keys that be accessed both for reading and writing, but you cannot change the actual `tex.nest` array itself. The possible keys are listed in the LuaTeX reference manual.
- `tex.linebreak(n, ...)` supports running the paragraph breaker from pure Lua. The second argument specifies a (potentially large) table of line breaking parameters: the parameters that are not passed explicitly are taken from the current typesetter state.

  The exact keys in the table are documented in the reference manual, but here is a simple yet complete example of how to run line breaking on the content of `\box0`:

  ```
  \setbox0=\hbox to \hsize{\input knuth }
  \startluacode
  local n = node.copy_list(tex.box[0].list)
  local t = node.tail(n)
  local final = node.new(node.id('glue'))
  final.spec = node.new(node.id('glue_spec'))
  final.spec.stretch_order = 2
  final.spec.stretch = 1
  node.insert_after(n,t, final)
  local m = tex.linebreak(n,
            { hangafter = 2,
              hangindent = tex.sp("2em")})
  local q = node.vpack(m)
  node.write(q)
  \stopluacode
  ```

  The result is:

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual. The separation of any of these four components would have hurt TeX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important. But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

6 Summary

All in all, there are not too many incompatible changes compared to LuaTeX 0.40, and the LuaTeX project is progressing nicely.

LuaTeX beta 0.70 will be released in the autumn of 2010. Our current plans for that release are: access to the actual PDF structures of included PDF images; a partial redesign of the mixed direction model; even more access to the LuaTeX internals from Lua; and probably some more ...

◇ Taco Hoekwater and Hartmut Henkel
`http://luatex.org`

LuaTEX: PDF merging

Hans Hagen

1 Introduction

It is tempting to add more and more features to the backend code of the engine but it is not really needed. Of course there are features that can best be supported natively, like including images. In order to include PDF images in LuaTEX the backend uses a library (xpdf or poppler) that can load a page from a file and embed that page into the final PDF, including all relevant (indirect) objects needed for rendering. In LuaTEX an experimental interface to this library is included, tagged as `epdf`. In this chapter I will spend a few words on my first attempt to use this new library.

2 The library

The interface is rather low level. I got the following example from Hartmut Henkel, who is responsible for the LuaTEX backend code and this library.

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc:getCatalog()
local pag = cat:getPage(3)
local box = pag:getMediaBox()

local w = pag:getMediaWidth()
local h = pag:getMediaHeight()
local n = cat:getNumPages()
local m = cat:readMetadata()

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ",box.x1,box.x2,box.y1,box.y2)
```

As you see, there are accessors for each interesting property of the file. Of course such an interface needs to be extended when the PDF standard evolves. However, once we have access to the so-called catalog, we can use regular accessors to the dictionaries, arrays and other data structures. So, in fact we don't need a full interface and can draw the line somewhere.

There are a couple of things that you normally do not want to deal with. A PDF file is in fact a collection of objects that form a tree and each object can be reached by an index using a table that links the index to a position in the file. You don't want to be bothered with that kind of housekeeping. Some data in the file, like page objects and annotations, are organized in a tree form that one does not want to access in that form, so again we have something that benefits from an interface. But the majority of the objects are simple dictionaries and arrays. Streams (these hold the document content, image data, etc.) are normally not of much interest, but the library provides an interface as you can bet on needing it someday. The library also provides ways to extend the loaded PDF file. I will not discuss that here.

Because in ConTEXt we already have the `lpdf` library for creating PDF structures, it makes sense to define a similar interface for accessing PDF. For that I wrote a wrapper that will be extended in due time (read: depending on needs). The previous code now looks as follows:

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc.Catalog
local pag = cat.Pages[3]
local box = pag.MediaBox

local llx, lly, urx, ury
  = box[1], box[2] box[3], box[4]

local w = urx - llx -- or: box.width
local h = ury - lly -- or: box.height
local n = cat.Pages.size
local m = cat.Metadata.stream

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ", llx, lly, urx, ury)
```

If we write code this way we are less dependent on the exact API, especially because the `epdf` library uses methods to access the data and we cannot easily overload method names in there. When you look at the `box`, you will see that the natural way to access entries is using a number. As a bonus we also provide the `width` and `height` entries.

3 Merging links

It has always been on my agenda to add the possibility to carry the (link) annotations with an included page from a document. This is not that much needed in regular documents, but it can be handy when you use ConTEXt to assemble documents. In any case, such a merge has to happen in a way that does not interfere with other links in the parent document. Supporting this in the engine is not an option as each macro package follows its own approach to referencing and interactivity. Also, demands might differ and one would end up with a lot of (error prone) configurability. Of course we want scaled pages to behave well too.

Implementing the merge took about a day and most of that time was spent on experimenting with the `epdf` library and making the first version of the wrapper. I definitely had expected to waste more time on it. So, this is yet another example of an extension that is quite doable in the Lua–TEX mix. Of course it helps that the ConTEXt graphic inclusion code provides enough information to integrate such a feature. The merge is controlled by the interaction key, as shown here:

```
\externalfigure[somefile.pdf][page=1,scale=700,
                              interaction=yes]
\externalfigure[somefile.pdf][page=2,scale=600,
                              interaction=yes]
```

You can fine-tune the merge by providing a list of options to the interaction key but that's still somewhat experimental. As a start the following links are supported.

- internal references by name (often structure related)
- internal references by page (like on tables of contents)
- external references by file (optionally by name and page)
- references to URIs (normally used for web pages)

When users like this functionality (or when I really need it myself) more types of annotations can be added although support for JavaScript and widgets doesn't make much sense. On the other hand, support for destinations is currently somewhat simplified but at some point we will support the relevant zoom options.

The implementation is not that complex:

- check if the included page has annotations
- loop over the list of annotations and determine if an annotation is supported (currently links)
- analyze the annotation and overlay a button using the destination that belongs to the annotation

Now, the reason why we can keep the implementation so simple is that we just map onto existing ConTEXt functionality. And, as we have a rather integrated support for interactive actions, only a few basic commands are involved. Although we could do that all in Lua, we delegate this to TEX. We create a layer that we put on top of the image. Links are put onto this layer using the equivalent of:

```
\setlayer
 [epdflinks]
 [x=...,y=...,preset=leftbottom]
 {\button
  [width=...,height=...,offset=overlay,frame=off]
  {}% no content
  [...]}}
```

The `\button` command is one of those interaction-related commands that accepts any action-related directive. In this first implementation we see the following destinations show up:

```
somelocation
url(http://www.pragma-ade.com)
file(somefile)
somefile::somelocation
somefile::page(10)
```

References to pages become named destinations and are later resolved to page destinations again, depending on the configuration of the main document. The links within an included file get their own namespace so (hopefully) they will not clash with other links.

We could use lower-level code which is faster but we're not talking of time-critical code here. At some point I might optimize the code a bit but for the moment this variant gives us some tracing options for free. Now, the nice thing about using this approach is that the already existing cross-referencing mechanisms deal with the details. Each included page gets a unique reference so references to not-included pages are ignored simply because they cannot be resolved. We can even consider overloading certain types of links or ignoring named destinations that match a specific pattern. Nothing is hard coded in the engine so we have complete freedom in doing that.

4 Merging layers

When including graphics from other applications it might be that they have their content organized in layers (that can then be turned on or off). So it will be no surprise that merging layer information is on the agenda: first a straightforward inclusion of optional content dictionaries, but it might make sense to parse the content stream and replace references to layers by those that are relevant in the main document. Especially when graphics come from different sources and layer names are inconsistent some manipulation might be needed, so maybe we need more detailed control. Implementing this is no big deal and mostly a matter of figuring out a clean and simple user interface.

◇ Hans Hagen
Pragma ADE
The Netherlands
http://luatex.org

The TEX paragraph builder in Lua

Hans Hagen

Abstract

In this article I will summarize some experiences with converting the TEX parbuilder to Lua. In due time there will be a plugin mechanism in ConTEXt, and this is a prelude to that.

1 Introduction

You enter the den of the Lion when you start messing around with the paragraph builder. Actually, as TEX does a pretty good job with breaking paragraphs into lines I never really looked into the code that does it all. However, the Oriental TEX project kind of forced it upon me. In a separate discussion of font goodies an optimizer is described that works per line. This method is somewhat similar to font expansion level one support, in the sense that it acts independent of the parbuilder: the split off (best) lines are postprocessed. Where expansion involves horizontal scaling, the goodies approach does with (Arabic) words what the original HZ approach does with glyphs.

It would be quite some challenge (at least for me) to come up with solutions that look at the whole paragraph and as the per-line approach works quite well, there is no real need for an alternative. However, in September 2008, when we were exploring solutions for Arabic paragraph building, Taco converted the parbuilder into Lua code and stripped away all code related to hyphenation, protrusion, expansion, last line fitting, and some more. As we had enough on our plate at that time, we never came back to thoroughly testing it. There was even less reason to explore this route because in the Oriental TEX project we decided to follow the “use advanced OpenType features” route which in turn led to the ‘replace words in lines by narrower or wider variants’ approach.

However, as the code was lying around and as we want to explore further, I decided to pick up the parbuilder thread. In this article some experiences will be discussed. The following story is as much Taco’s as mine.

2 Cleaning up

In retrospect, we should not have been too surprised that the first approximation was broken in many places, and for good reason. The first version of the code was a conversion of the C code that in turn was a conversion from the original interwoven Pascal code. That first conversion still looked quite C-ish and carried interesting bits and pieces of C macros, C-like pointer tests, interesting magic constants and more.

When I took the code and Lua-fied it nearly every line was changed and it took Taco and me a bit of reverse engineering to sort out all problems (thank you Skype). Why was it not an easy task? There were good reasons.

- The parbuilder (and related hpacking) code is derived from traditional TEX and has bits of pdfTEX, Aleph (Omega), and of course LuaTEX.
- The advocated approach to extending TEX has been to use change files which means that a coder does not see the whole picture.
- Originally the code is programmed in the literate way which means that the resulting functions are built stepwise. However, the final functions can (and have) become quite large. Because LuaTEX uses the woven (merged) code indeed we have large functions. Of course this relates to the fact that successive TEX engines have added functionality. Eventually the source will be webbed again, but in a more sequential way.
- This is normally no big deal, but the Aleph (Omega) code has added a level of complexity due to directional processing and additional begin and end related boxes.
- Also the ε-TEX extension that deals with last line fitting is interwoven and uses goto’s for the control flow. Fortunately the extensions are driven by parameters which makes the related code sections easy to recognize.
- The pdfTEX protrusion extension adds code to glyph handling and discretionary handling. The expansion feature does that too and in addition also messes around with kerns. Extra parameters are introduced (and adapted) that influence the decisions for breaking lines. There is also code originating in pdfTEX which deals with poor man’s grid snapping although that is quite isolated and not interwoven.
- Because it uses a slightly different way of dealing with hyphenation, LuaTEX itself also adds some code.
- Tracing is sort of interwoven in the code. As it uses goto’s to share code instead of functions, one needs to keep a good eye on what gets skipped or not.

I’m pretty sure that the code that we started with looks quite different from the original TEX code if it had been translated into C. Actually in modern TEX, compiling involves a translation into C code first but the intermediate form is not meant for human eyes. As the LuaTEX project started from that merged code, Taco and Hartmut already spent quite some time on making it more readable. Of course the original comments are still there.

Cleaning up such code takes a while. Because both languages are similar and yet quite different, it took some time to get compatible output. Because the C code uses macros, careful checking was needed. Of course Lua's table model and local variables brought some work as well. And still the code looks a bit C-ish. We could not diverge too much from the original model simply because it's well documented.

When moving around code, redundant tests and orphan code have been removed. Future versions (or variants) might as well look much different as I want more hooks, clearly split stages, and to convert some linked-list-based mechanism to Lua tables. On the other hand, as much code has been written already for ConTEXt MkIV, making it all reasonably fast was no big deal.

3 Expansion

The original C code related to protrusion and expansion is not that efficient as many (redundant) function calls take place in the linebreaker and packer. As most work related to fonts is done in the backend, we can simply stick to width calculations here. Also, it is no problem at all that we use floating point calculations (as Lua has only floats). The final result will look okay as the original hpack routine will nicely compensate for rounding errors as it will normally distribute the content well enough. We are currently compatible with the regular parbuilder and protrusion code, but expansion gives different results (not worse).

The Lua hpacker follows a different approach. And let's admit it: most TEXies won't see the difference anyway. As long as we're cross-platform compatible it's fine.

It is a well-known fact that character expansion slows down the parbuilder. There are good reasons for this in the pdfTEX approach. Each glyph and intercharacter kern is checked a few times for stretch or shrink using a function call. Also each font reference is checked. This is a side effect of the way the pdfTEX backend works as there each variant has its own font. However, in LuaTEX, we scale inline and therefore don't really need the fonts. Even better, we can get rid of all that testing and only need to pass the eventual `expansion_ratio` so that the backend can do the right scaling. We will prototype this in the Lua version[1] and when we feel confident about this approach it will be backported into the C code base. So eventually the C code might become a bit more readable and efficient.

[1] For this Hartmut has adapted the backend code to honour this field in the glyph and kern nodes.

Intercharacter kerning is dealt with somewhat strangely. When a kern of subtype zero is seen, and when its neighbours are glyphs from the same font, the kern gets replaced by a scaled one looked up in the font's kerning table. In the parbuilder no real replacement takes place but as each line ends up in the hpack routine (where all work is simply duplicated and done again) it really gets replaced there. When discussing the current approach we decided that manipulating intercharacter kerns while leaving regular spacing untouched is not really a good idea so there will be an extra level of configuration added to LuaTEX:[2]

0	no character and kern expansion
1	character and kern expansion applied to complete lines
2	character and kern expansion as part of the parbuilder
3	only character expansion as part of the parbuilder (new)

You might wonder what happens when you unbox such a list: the original font references have been replaced as are the kerns. However, when repackaged again, the kerns are replaced again. In traditional TEX, indeed rekerning might happen when a paragraph is repackaged (as different hyphenation points might be chosen and ligature rebuilding etc. has taken place) but in LuaTEX we have clearly separated stages. An interesting side effect of the conversion is that we sometimes wonder what certain code does and if it's still needed.

4 Performance

We had already noticed that the Lua variant was not that slow, so after the first cleanup it was time to do some tests. We used our regular `tufte.tex` test file. This happens to be a worst case example because each broken line ends with a comma or hyphen and these will hang into the margin when protruding is enabled. So the solution space is rather large (an example will be shown later).

Here are some timings of the March 26, 2010, version. The test is typeset in a box so no shipout takes place. We're talking of 1000 typeset paragraphs.

[2] As I more and more often run into books typeset (not by TEX) with a combination of character expansion and additional intercharacter kerning I've been seriously thinking of removing support for expansion from ConTEXt MkIV. Not all is progress especially if it can be abused.

The times are in seconds and in parentheses the speed relative to the regular parbuilder is given.

	native	**lua**	**lua + hpack**
normal	1.6	8.4 (5.3)	9.8 (6.1)
protruding	1.7	14.2 (8.4)	15.6 (9.2)
expansion	2.3	11.4 (5.0)	13.3 (5.8)
both	2.9	19.1 (6.6)	21.5 (7.4)

For a regular paragraph the Lua variant (currently) is 5 times as slow and about 6 times when we use the Lua hpacker, which is not that bad given that it's interpreted code, and each access to a field in a node involves a function call. Actually, we can make a dedicated hpacker as some code can be omitted. The reason why the protruding is relatively slow is that we have quite a few protruding characters in the test text (many commas and potential hyphens) and therefore we have quite a few lookups and calculations. In the C code implementation much of that is inlined by macros.

Will things get faster? I'm sure that I can boost the protrusion code and probably the rest as well but it will always be slower than the built-in function. This is no problem as we will only use the Lua variant for experiments and special purposes. For that reason more MkIV-like tracing will be added (some is already present) and more hooks will be provides once the builder is more compartmentalized. Also, future versions of LuaTEX will pass around paragraph-related parameters differently so that will have impact on the code as well.

5 Usage

The basic parbuilder is enabled and disabled as follows:[3]

```
\definefontfeature[example]
  [default][protrusion=pure]
\definedfont[Serif*example]
\setupalign[hanging]

\startparbuilder[basic]
    \startcolor[blue]
        \input tufte
    \stopcolor
\stopparbuilder
```

There are a few tracing options in the `parbuilders` namespace but these are not stable yet.

6 Conclusion

The module started working quite well around the time that Peter Gabriel's "Scratch My Back" ended up in my Squeezecenter: modern classical interpretations of some of his favourite songs. I must admit that I scratched the back of my head a couple of times when looking at the code. It made me realize that a new implementation of a known problem indeed can come out quite different but at the same time has much in common. As with music it's a matter of taste which variant a user likes most.

At the time of this writing there is still work to do. For instance, the large functions need to be broken into smaller steps. And of course more testing is needed.

◇ Hans Hagen
Pragma ADE
The Netherlands
http://luatex.org

[3] I'm not sure yet if the parbuilder has to do automatic grouping.

Thirty years of literate programming and more?

Bart Childs

Abstract

Don Knuth created Literate Programming about thirty years ago. It could be called a methodology, discipline, paradigm, … Bentley's "Programming Pearls" article about Knuth's book, *TEX: The Program*, caused a huge stir in the computing professions. Soon there was announcement of a Literate Programming section for the *CACM*. Several "Literate Programming systems" quickly appeared. This was followed by a few years of mild interest, cancelling the Literate Programming section in the *CACM*, and an apparent lack of public interest in the subject.

Really, what is literate programming?
What is the state of literate programming?

1 Introduction

It is commonly accepted in software engineering circles that one of the greatest needs in computing is the reduction of the cost of maintenance of codes. Maintenance programmers spend at least half of their time trying to understand what code does and maintenance is accepted to be 60% to 80% of a code's cost. In *The Mythical Man-Month: Essays on Software Engineering*, Frederick Brooks stated:

> **Self-Documenting Programs**
>
> A basic principle of data processing teaches the folly of trying to maintain independent files in synchronism. It is far better to combine them into one file with each record containing all the information both files held concerning a given key.
>
> Yet our practice in programming documentation violates our own teaching. …
>
> The results in fact confirm our teachings about the folly of separate files. Program documentation is notoriously poor, and its maintenance is worse…
>
> The solution … is to merge the files, to incorporate the documentation in the source program. This is at once a powerful incentive toward proper maintenance, and an insurance that the documentation will always be handy to the program user. Such programs are called *self-documenting* …

2 Definition of literate programming

Literate programming is a methodology/process/system for creation of codes in the form of a work of literature as well as executable programs.

The *characteristics of literate programs*, based on Knuth's `WEB` [2] and `CWEB` [3] (with Silvio Levy):

1. The section is the basic unit in the source of a literate program which is analogous to paragraphs in usual textual documents. The section's source should generally be about a single screen.
2. Sections can contain documentation, definitions (macros), and/or code.
3. The order of presentation of sections should maximize the readability of the literate program. There are elements of the structure of the code parts of sections to ensure the correct placement of the code in the resulting program source.
4. Documentation elements of a section should be presented in a format consistent with book quality documents.
5. Code parts and code fragments in documentation parts of sections should be presented in a format consistent with book quality documents. This imposes a requirement of the system parsing the computer language(s) used.

These characteristics should be in a literate programming system for programming in most high level languages. There are examples where a restricted form of literate programming is quite helpful — often the last item in the above list is omitted. These systems still meet Brooks' call for *self-documenting* programs.

My initial experience with literate programs was porting the TEX system to several different systems. I found the formatting of the code to be a great help, especially font selection for keywords, variables, and literals as well as consideration of grouping, loops, etc. Obviously, this requires parsing the code.

2.1 Knuth's Pascal `WEB` and descendants

There were a number of features in the original `WEB` that were needed to make up for problems doing systems programming in Pascal. Levy did not include these in `CWEB` because of the nature of the C language. This literate programming system has endured for nearly three decades. There is no apparent need to change it because of the evolving nature of Pascal. Pascal is not a sufficiently prominent language for systems programming and `web2c` has enabled TEX and its components and friends to be widely ported. Knuth's original `WEB` was done at an early time — relative to most of today's understanding of systems and programming languages — and many features were due to Pascal limitations. The selection of Pascal was done before C would have been a reasonable choice — by only a few years. I strongly recommend reading the original documentation and the documents produced by processing the original literate programs in the suite of tools to support the TEX system.

There are at least two systems still in use that are quite faithful to the philosophy that Knuth elucidated in his original Pascal-based WEB system and are consistent with the definition and the list of characteristics given: CWEB and FWEB. Each of these support more than one language.

2.2 CWEB — Levy and Knuth

Levy's original CWEB was an adaption of WEB to the use of C. Several features of WEB that were needed for Pascal were removed. Knuth joined Levy in the support and evolution of CWEB and others contributed the addition of support of C++ and Java.

2.3 FWEB — Krommes

John Krommes' FWEB is based on CWEB. FWEB supports C, C++, Fortran (77) and Ratfor. Krommes' research dictated the need for this multilingual nature of FWEB because his research was based on both long running Fortran programs and programs to interpret the data that were done in C++ or C.

Fortran had many vagaries that exceeded those of Pascal, including its ancient card orientation. The acronym was changed to a noun in the '90s. Fortran is central on many parallel systems and each seems to have a unique system of compiler directives that are often required to be in-line with the code. Krommes handled these with admirable foresight. The Fortran standards committee has been active in trying to bring that community into the twenty-first century. For example, semicolons are now allowed to end statements.

Krommes included multiple output and input files as well as the option of being language independent that enabled the logical next step of including scripts as part of the literate program. This language-independent mode is called verbatim.

All in all, I would have liked a much smaller version of FWEB. That sounds like a common whine about TEX: "It is too big!"

We found that a small part of TEX and a web can be taught to beginning students (see section 4). It simply requires some work.

2.4 WEB-like systems

It is my opinion that the formatted code that the above literate programming systems give for their high level languages is of great benefit. Others obviously do not share my enthusiasm.

Several systems that have been called literate programming by their creators are language independent and therefore do not meet the characteristic in item 5, section 4. This feature of *not parsing and formatting* the programming language allows the use of many different languages. This simplicity and flexibility is desirable but I believe the benefits of the formatting are crucial.

Creators of two of these systems, Williams (FunnelWEB [8]) and Ramsey (NoWEB [6]) obviously have a different opinion and focus on the benefits of being able to order sections for expository reasons rather than compiler requirements, and include effective documentation as making this form of literate programming worthwhile.

I recall a reply by Williams in the literate programming discussion group in response to a user complaining about the tangled output not looking like the user wanted: "Crikey, will they ever learn? If the web is well written you will not want to look at that version of the code." Well, something like that.

2.5 docstrip and doc.sty — LATEX tools

Frank Mittelbach created the doc.sty package to combine the TEX code and documentation for LATEX. He then created docstrip to complete a literate programming system for LATEX [4]. This might be the most used literate programming system on a regular basis because it is the common format for LATEX distributions. It should also be noted that there have been several contributors to the evolution of these tools, as is typical of the TEX community.

The advantage of having the documentation and code in one file has been discussed earlier. Since docstrip is written in TEX, which is an interpreter, minimizing comments in the output code was important to execution speeds.

The code part is not parsed when the document is processed, thus the system can be used for code other than (LA)TEX. I have found references to the use of docstrip with other coding systems, notably statistical packages. Perhaps a future article will explore docstrip in more detail.

2.6 Literate programming-like usage

Nelson Beebe created a system he described as "like literate programming" to document the many scripts (each a code fragment) for his book *Shell Scripting* [7] (with Arnold Robbins).

Like the previous subsection, a future paper is needed for a survey of many such uses of the ideas of literate programming.

3 web-mode — An Emacs-based tool

Mark Motl finished his dissertation under my direction by developing and testing a tool to adapt Emacs to literate programming for `WEB` and `CWEB` [5].

The selection of Emacs was a bit like Knuth's selection of Pascal for the second writing of TEX. Emacs was/is a large, stable system and relatively platform independent. The emergence of workstations with a tightly coupled graphics screen was also a great benefit to the Emacs philosophy. Much of the early development of `web-mode` was done on shared resource systems and the final work was done on workstations.

Finishing touches were added by several students and some by me. Some characteristics of `web-mode`:

- Emacs is cross platform. Most of my use in the last few years has been on Macs, PCs, Sun, and Linux workstations.
- It is open in the same sense as TEX and Emacs.
- The user specifies he/she is using `WEB`, `CWEB`, or `FWEB`.
- If the web is a new file, the appropriate header files are inserted with customized user-specific information.
- For existing webs, navigation information is developed unless files (like `.aux` in LATEX and similar `WEB` files) are newer than the web source.
- As the user enters source, the source is parsed to ensure that section elements are complete. For example, the meta-ness of code section names have proper balance and the trailing `=` sign, if appropriate.
- Knuth included a feature to allow the user not to type the entire code section name, but use an ellipsis for completion. In `web-mode` the Emacs completion feature makes this unnecessary.
- Navigation of a web can be done by chapter/ section name/number, by sections referencing variables, etc. These actions can be invoked by function keys or pull down menus.
- The user can view the web and change file by source, or preview of the DVI or PDF output.
- Execution of the `TANGLE`, `WEAVE`, TEX, LATEX, can be invoked by function key or pull-down menus.
- Outline editing of the source is especially useful. The first line of sections (and chapters) and the lines defining code sections are displayed. It gives a new meaning to scrolling through source.

There is much more but that detail is not needed here.

The distribution of `web-mode` will be available soon from my home page after I perform some consistency checks on current Windows and Apple systems.

4 My CS/1 experience

This section is adapted from our paper at the 16th TUG meeting in St. Petersburg [1].

We embarked on a project to teach the first computer science course using literate programming while covering all the topics covered in the usual sections. We differed from the environment used by the other sections by using Emacs and the literate programming environment `web-mode` and GNU Pascal as opposed to Turbo Pascal.

Our CS/1 course was entitled "Programming I" and although the catalog did not specify Pascal, it was understood that all sections of the course would use the same language and that problem solving would be central.

An inherent part of these CS/1 courses is to develop the student's skills in *problem solving*. Indeed, in many course outlines, that is part of the title and the main emphasis in the description of the course contents. A problem solving methodology is often stated in CS/1 courses which generally has steps like:

1. State the problem completely!
2. Develop all necessary assumptions.
3. Develop an algorithm and test data set(s).
4. Code the problem.
5. Analyze the results (and iterate?).

Literate programming is a style in which the design of the code reflects that the human reader is as important as the machine reader. The human reader is often associated with the expensive process of maintenance and the machine reader is the compiler/interpreter. Literate programming is a process which should lead to more carefully constructed programs with better, relevant 'systems' documentation. We think that the first sentence in this paragraph should be particularly relevant to students because the human reader (the one who assigns grades) is obviously the most important reader.

The features of literate programming that gave us the confidence to expect positive results are:

1. Top-down and bottom-up programming since it is structured pseudo-code.
2. Programming in small sections where most sections of code and documentation (section in this use is similar to a paragraph in prose) are approximately a screen or less of source.
3. Typeset documentation (after all, Knuth was rewriting TEX).
4. Pretty-printed code where the keywords are in bold, user supplied names in italics, etc.
5. Extensive reading aids are automatically generated including a table of contents and index.

The readability is improved by the programmer's liberal use of the tools furnished.

Each item in the following list could be added to the corresponding item in the previous list. We think there is merit to this split.

1. These topics are usual in CS/1 books but they generally lack the integration to make them really effective for the student.
2. Divide and conquer is also espoused but the larger examples given in many books forsake the principle.
3. It may be argued that this is 'feeding pearls to the swine' but we like the cognitive emphasis that comes from logical substitution of words for key-words, etc.
4. The fact that `weave` breaks lines based on its parsing is another cognitive reinforcement.
5. Encouraging/requiring students to review their programs as documents makes them *think* about readability.

I am still firmly convinced that literate programming is the way to set budding programmers and systems developers on the right track to writing beautiful and excellent programs. There was a section entitled *Problems with 'Problem Solving'* in the TUG 16 paper which gave many of the arguments for this that we gleaned from the literature.

"Are You Crazy!?" This was frequently shouted at us because, as *everybody in the world knows:*

- Emacs is impossible to learn and use,
- TEX is impossible to learn,
- the addition of `WEB` makes for too many steps to learn, and
- there is a reason for all those Aggie jokes.

and *therefore* our project was doomed!

Sometimes items from that list were suggested gently instead of being yelled or blurted out while the correspondent was writhing on the floor. We exercised a little judgement and did it on the smallest sections of the course, namely the honors sections.

There were several important items that we considered in the design of the course and how we executed the course.

Testing We intended the course to be more *problem-oriented* rather than *program-oriented.* The tests included a pre-test that was no part of the other sections. All tests were slanted away from Pascal details.

Emacs and `web-mode` We felt that in spite of this being a new editor to nearly all, the `web-mode` literate programing tool was our only choice. We modified the Emacs reference card and gave the students a five page memo based on Knuth's `WEB` introduction.

Knuth's `WEB` We were restrained to the use of Pascal and therefore Knuth's original `WEB` was appropriate. Some of the necessary minutiae was easily omitted by use of `web-mode`.

How the course was taught The focus of the semester was on problem solving. Pascal syntax was brought along as a means of presenting and then implementing a solution. The lecturer presented the week's information and handled questions on a daily basis. The TA handled the labs. Total enrollment in the class was about 40 and about half in each lab section.

Do all labs twice The labs in our courses are usually twice a week rather than one long period per week. We took advantage of that and each lab was done twice. The first time was used to have the documentation parts of sections being somewhat complete and the code parts sketchy. `WEB` used in this manner can be a documented pseudo-code system. This draft was marked quickly and returned for the more complete version to be finished before moving on to the next lab.

4.1 Results — informal summary

Three pages of detailed results were presented in the TUG 16 paper. I will present a high level view of those I think that are most important and relevant.

The initial design would have led to failure if we had analyzed the results promptly as planned. Irrelevant personal problems delayed that analysis for more than a year. The one result from the immediate analysis was

> The pre-test showed that the non-majors did not have the problem solving skills of the majors. The change was steady and by the second regular test the non-majors were superior.

Most of the students had completed more computer science courses by the time the analysis was started in earnest. The evaluation process was modified to include tracking those students who took these additional courses. Also, data was extracted for students taking the same class the previous year, this was taught by a more experienced teacher.

The additional courses were a CS/2 course which was dominated by learning the C language and then a data structures class. The performance in the CS/2 courses were not significantly different for those with and without the literate programming exposure.

The literate programming exposure apparently made a significant difference in the data structures courses. With hindsight, that seems logical because a data structures course is much more of a problem solving experience and as implied above CS/2 was too much a memorization of C syntax. The CS/2 and data structures courses mentioned were not taught by those involved in this study.

We feel the background of the students was not atypical of many CS/1 type courses. The majority of the class were majoring in computer science, but a significant number were using the course as a minor elective, a basis for deciding if they want CS as a major, or other reasons. There was not an unusual change of majors for the students in the study.

4.2 Student comments and evaluation

Upon nearing completion of the CS/1 course, the students were asked to submit a paper reflecting their feelings and attitudes towards the `WEB` programming methodology. This was to be written as a typical one-page technical note. Some expected comments were made in the evaluation process at the end of the CS/1 course taught using literate programming.

- TeX is not easy to learn,
- learning `WEB` was OK.
- Emacs was difficult (the replacement of function keys by pull-down menus was not complete in `web-mode` at the time).

4.3 Conclusions about teaching

We taught an honors section of a CS/1 course in a different manner than usual, namely using literate programming. The students used an editor, a formatting system, and a coding style that was new to all. The students' performance in subsequent courses was not hurt and may have been helped with the different methodology. The results of using the program development methodology in the CS/1 course indicate that the methodology is successful in teaching novice programmers good problem solving skills.

These are the results of the experiment:

- The students showed an increase in their problem solving skills.
- Those students unfamiliar with the Pascal programming language, or any other programming language, were more successful then those familiar with Pascal at using the literate programming paradigm to capture and document their problem solving process.
- The students were able to learn the `WEB` rules, the `web-mode` environment, GNU Emacs, and TeX rules, as well as the Pascal syntax and constructs.
- Those students exposed to the program development methodology utilizing the literate programming paradigm were as successful in the subsequent CS/2 course as those not exposed to the methodology.
- Those students exposed to literate programming were significantly more successful in the data structures course than those not exposed to the methodology.
- The subject program development methodology may lead to an improved software development process; however, more tests should be conducted.

5 Tools

This list is used to describe some of the tools I know of that exist to aid in literate programming. Some have not been widely published, much to my shame.

CWEB There are several tools referenced on Knuth's home page; see his `CWEB` area.

Leo There are several references to this "outline" editor. I have not had time to seriously look at this yet.

`web-mode` I have corresponded with many users over the years but did not realize that `web-mode` was invisible to the usual literature reviews. This will be corrected after a detailed review and making sure of Emacs updates. The only problems I have encountered are AUCTeX's implementation of `description` environments and conscription of some function keys.

TAMU We did several tools and need to organize and publish them as a set. A tool that was frequently used when applied to a `WEB` or a TeX source would give a statistical analysis of commands (I used more "features" than the students).

6 Examples of literate programs

The long term success of literate programming may depend on the number and quality of published literate programs. A partial listing of literate programs that are available openly is offered here as a start.

WEB The sources yield `TANGLE` and `WEAVE` at a minimum.

CWEB As above. The manual was last updated eight years ago and is now out of print.

TeX and Metafont, the programs The printed books are available from Addison-Wesley or you can exercise your printer and paper budget.

Stanford GraphBase Knuth presents 31 `WEB`s in his platform for combinatorial computing.

Don Knuth's home page Nearly innumerable interesting `CWEB`s for a wide range of topics.

CACM The few that were contributed to the Literate Programming column.

BC While preparing this I encountered a reference to a code of mine, `PS_Quasi`, which related to experimental, theoretical numerical solutions of ordinary differential equations. The link failed but I need to reestablish that.

I also have a few dozen that were done by our team that should be termed tools for analyzing literate programs. These need to be cleaned up, cataloged and published (on the web).

I have emphasized literate programs based on the three systems that meet the definition I stated earlier. The intended functions of those systems center on programming in specific high level languages.

7 The state of literate programming

This is a difficult topic to treat with authority and a straight face. Most of the systems have had only minor changes, if any, in the last twenty years. However, stability is frequently a good thing. It is common knowledge that adding functions to an interface will make it more difficult to use.

Identification of and counting the number of users of literate programming and "literate programming like" systems may be impossible. Some visibility includes:

original style Knuth's home page and books point to many excellent examples. I am remiss in not making a number of examples and tools available, but I will.

literateprogramming.com Some elements point to example literate programs, but only a few. It includes a fair number of references to many items that someone has called literate programming. This includes a note entitled "POD is not literate programming." Most of the entries are from the last century, but that is not so long ago.

literate programming *like* Robbins and Beebe's *Shell Scripting* book. This perhaps could have been done using `docstrip`. How many similar projects are there that could be helped by a good survey?

literate programming — other There is a community of users of the packages FunnelWEB and NoWEB. I particularly like Ramsey's philosophy of piping small tools, following the Unix philosophy. Application of these systems (and `docstrip`) is also available with the statistical system R (see Uwe Ziegenhagen's article in these proceedings, pp. 189–192).

8 Conclusions

I believe that this style of program development is a great contribution to the goal of creating excellent and maintainable programs, if it is used diligently. I have often wondered how many of the errors that Knuth has rewarded us for would have even been found if the program had been in the style of Unix "pretty printing." In spite of this, it is referenced too little.

We have observed first year students are already like the professionals: "No, I do not want to learn anything new if I already have some knowledge in the area." We think it should appear early and repeatedly in the curriculum. The design process that is called for in most software engineering treatises is a natural fit for literate programming, in my opinion.

References

[1] Bart Childs, Deborah Dunn, and William Lively. Teaching CS/1 courses in a literate manner. *TUGboat*, 16(3):300–309, September 1995.

[2] Donald E. Knuth. The `WEB` system of structured documentation. Stanford Computer Science Report CS980, Stanford University, Stanford, CA, September 1983.

[3] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0.* Addison-Wesley, Reading, MA, USA, 1993.

[4] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński, and Mark Wooding. The `docstrip` program. Technical report, Universität Mainz, Mainz, Germany, 2005.

[5] Mark Bentley Motl. *A Literate Programming Environment Based on an Extensible Editor.* PhD thesis, Texas A&M University, College Station, TX, December 1990.

[6] Norman Ramsey. Weaving a language-independent `WEB`. *Communications of the ACM*, 32(9):1051–1055, September 1989.

[7] Arnold Robbins and Nelson H. F. Beebe. *Classic Shell Scripting.* O'Reilly, Sebastopol, CA, 2005.

[8] Ross N. Williams. Funnelweb User's Manual. `http://www.ross.net/funnelweb/`, May 1992. V1.0 for FunnelWeb V3.0.

◇ Bart Childs
Texas A&M University
College Station, TeXas 77843-3112
USA
`bart (at) tamu dot edu`
`http://faculty.cse.tamu.edu/bart/`

Dynamic reporting with R/Sweave and LaTeX

Uwe Ziegenhagen

Abstract

R is a sophisticated statistical programming language available for various platforms. Since its initial development in 1992 it has become the major open source tool for statistical data analysis. For the integration with LaTeX it provides tools which allow the convenient dynamic creation of reports. In this article I will give a very brief introduction to R and show how R integrates in the LaTeX workflow.

1 Introduction to R

The history of R dates back to 1969 when John M. Chambers from Bell Labs published the outlines of S, a programming language for statistics and data analysis that was first implemented in 1975 for Honeywell computers. Starting in 1992 Ross Ihaka and Robert Gentleman from the University of Auckland in New Zealand took up the concepts underlying S to develop R, a free implementation of the language. Today R is for many statisticians the tool of choice for visualization and data analysis, covering all aspects of modern computer-based statistics. The R project team has more than 500 members; more than 1,000 packages are available on CRAN, the Comprehensive R Archive Network.

1.1 R as a calculator

Since the focus of this paper lies more on the interaction with LaTeX this is not the place for a thorough introduction to R . Interested readers may have a closer look in the bibliography of this article for suitable materials. Nevertheless I would like to point out some of the main features of the language. Listing 1.1 shows some of the operators for basic calculations.

```
1+2
1*2
1/2
1-2
2^2
sqrt(2)
sin(pi)    # cos, tan
trunc(-pi) # -3
round(pi)  # 3
```

Listing 1.1: Basic calculations with R

The basic objects to store variables in R are vectors, matrices and dataframes. Vectors and matrices may contain only a single data type; complex data structures are stored in so-called dataframes, which are in fact lists of objects that may have different data types. Various ways of creating and assigning vectors to variables are shown in Listing 1.2.

```
a <- 1:3 # store vector 1..3 in a
b = 2:4  # store 2..4 in b
c(a,b)   # [1] 1 2 3 2 3 4 # cat a & b
# generate sequence
seq(1,2,by=0.1) [1] 1.1 1.2 1.3 ...
# repeat 1..4 twice
rep(1:4,2) # [1] 1 2 3 4 1 2 3 4
```

Listing 1.2: Generating vectors in R

Our final R example in Listing 1.3 shows how a simple linear model can be computed with R. The vector of independent variables x contains the numbers 1 to 10; for the vector of dependent variables y we just multiply the x-vector with a random factor taken from a normal distribution. The linear model is then calculated using the `lm` command which presents the coefficients of the linear model.

```
> x<-1:10
> y=rnorm(10)*x
> lm(y~x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
     0.1079     1.0697
```

Listing 1.3: A linear model with R

1.2 R graphics

R manages its graphical output (see the basic example code in Listing 1.4 and its output in Figure 1) through *graphics devices* which take the graphics object data and convert them into printable or viewable form. The list of available graphics devices is extensive; there are devices for PDF, PostScript, X11, Java and SVG, to mention a few. Listing 1.5 shows for example how the PDF device can be used to produce high-quality PDF files.

```
a<- c(1:10)
plot(a)
```

Listing 1.4: The most basic R plot

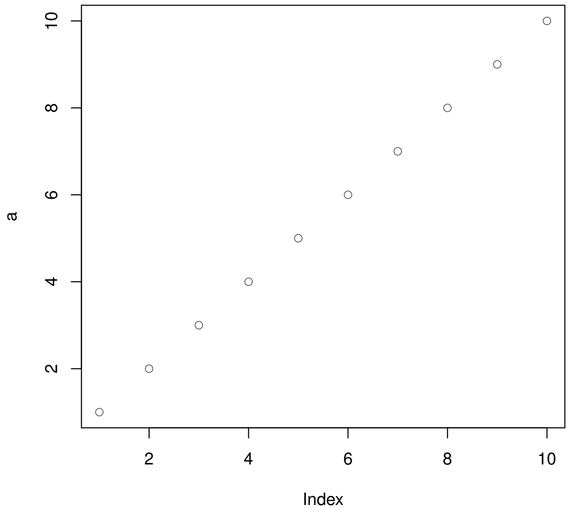


Figure 1: Graphics generated by Listing 1.4

```
pdf(file = "c:/punkte.pdf",width = 6,
  height = 6, onefile = FALSE,
  family = "Helvetica",
  title = "R Graphics Output",
  fonts = NULL, version = "1.4",
  paper = "special")

a<- c(1:10)
plot(a)

# switch back to screen device
dev.off()
```

Listing 1.5: Example code for the PDF device

1.3 The Ti*k*Z graphics device

Especially interesting for TeXnicians is the Ti*k*Z device which generates source code for the LaTeX graphics package of the same name. This device either creates files that can be compiled standalone or just the graphics code to be embedded in a LaTeX document. The advantage of this device, compared with others, is that the internal fonts of the document are used and mathematical code may be used in captions and labels as well. Listing 1.7 shows an excerpt from the file generated by the code from Listing 1.6.

```
tikz(file = "c:/test2.tex",standAlone=F)
# StandAlone=T
plot(1:10)

dev.off()
```

Listing 1.6: Example code for the Ti*k*Z device

```
% Created by tikzDevice
\begin{tikzpicture}[x=1pt,y=1pt]
\draw[color=white,opacity=0] (0,0)
rectangle (505.89,505.89);
\begin{scope}
\path[clip] ( 49.20, 61.20) rectangle (480.69,456.69);
\definecolor[named]{drawColor}{rgb}{0.56,0.96,0.51}
\definecolor[named]{fillColor}{rgb}{0.13,0.09,0.52}
\definecolor[named]{drawColor}{rgb}{0.00,0.00,0.00}
\draw[color=drawColor,line cap=round,line join=round,
fill opacity=0.00,] ( 65.18, 75.85) circle ( 2.25);
\draw[color=drawColor,line cap=round,line join=round,
fill opacity=0.00,] (109.57,116.54) circle ( 2.25);
\end{scope}
\begin{scope}
```

Listing 1.7: Excerpt from the generated Ti*k*Z code

2 Sweave and R

2.1 Introduction

In the second part of the article I want to introduce the Sweave package, developed by Friedrich Leisch. Sweave is part of the standard R installation so it requires no additional effort to install.

The package allows to include both R and LaTeX code in a single file. The R code is enclosed in "noweb" tags, `<<>>=` for the beginning, `@` for the end. Noweb is a free tool implementing Donald Knuth's approach of literate programming (for more details on this topic please see the articles in Wikipedia, et al.). The noweb-file is then processed within R using the command `Sweave("<filename>")`. To extract the R code from the file, Sweave also provides a second command, `Stangle`.

Listing 2.1 shows a very basic example, calculating $1 + 1$.

```
\documentclass{article}
\begin{document}
<<>>=
1+1
@
\end{document}
```

Listing 2.1: Basic Sweave example

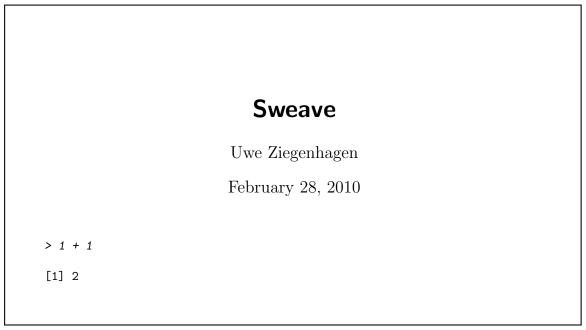
When we process the file from Listing 2.1 using the `Sweave` command in R we receive the LaTeX document shown in Listing 2.2. This document can then be compiled to PDF or DVI shown in Figure 2. As we can see in the document, Sweave requires the LaTeX package of the same name which provides commands for the input and output of Sweave code to be in the search path.

2.2 Sweave options

Sweave allows various options to be set within the `<<>>=` tag; for example, `echo=false` suppresses the

```
\documentclass{article}
\usepackage{Sweave}
\begin{document}
\begin{Schunk}
\begin{Sinput}
> 1 + 1
\end{Sinput}
\begin{Soutput}
[1] 2
\end{Soutput}
\end{Schunk}
\end{document}
```

Listing 2.2: LaTeX generated by Listing 2.1



Sweave

Uwe Ziegenhagen

February 28, 2010

> 1 + 1

[1] 2

Figure 2: PDF file generated by Listing 2.1

output of the original R source, while `results=hide` suppresses the output of results. A combination of both options may not seem to make sense, but this can be used to load data in the beginning of the analysis, set default values for variables, etc.

Since there are R packages that directly create valid LaTeX source, Sweave supports `results=tex`, a mode that passes the generated output through to LaTeX without tampering with its content.

If images are created in an R code chunk the option `fig=true` needs to be set. The default setting is to create both PostScript and PDF versions of each plot; the settings `eps=true`/`false` and `pdf=true`/`false` adjust this, respectively. Finally, the size of the plot can be set using the `width` and `height` parameter specifying the width and height of each plot in inches. Options may also be set globally by `\SweaveOpts`; see the Sweave manual for details.

Sweave also implements the noweb way of re-using code chunks: a certain piece of code can be named with `<<`⟨*name*⟩`, opt=...>>=`; the user may then use these chunks with `<<`⟨*name*⟩`>>`.

For scalar results which can be, for example, embedded in the running text, Sweave provides the `\Sexpr(`⟨*R-code*⟩`)` command. The only requirement for the code is that the return value must be either a string or an object that can be converted to a string. We will use this command in the example described next, shown in Listing 2.3.

```
\documentclass[a4paper]{scrartcl}
\begin{document}
<<echo=false,results=hide>>=
data(iris) # load iris data
@
The data set has \Sexpr{ncol(iris)} columns
and \Sexpr{nrow(iris)} rows.

<<echo=false>>=summary(iris$Petal.Length)
@
<<echo=false,results=tex>>=
xtable(lm(iris$Sepal.Width~iris$Petal.Length),
caption="Linear Model of Sepal.Width
and Petal.Length")
@
\centering
\begin{figure}[htp]
<<fig=true,echo=false>>=
pch.vec <- c(16,2,3)[iris$Species]
col.vec <- c(16,2,3)[iris$Species]
plot(iris$Sepal.Width,iris$Petal.Length,
col = col.vec,pch=pch.vec)
@
\caption{Plot of iris\$Petal.Length vs. iris\$Sepal.Width}
\end{figure}
\end{document}
```

Listing 2.3: Sweave code to generate Figure 3

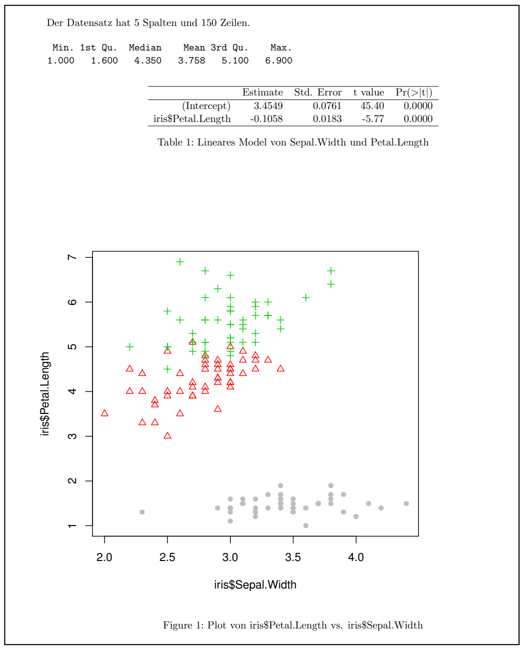
2.3 The iris example

Listing 2.3 shows a brief example for a statistical analysis of the well-known iris dataset, consisting of each 50 observations for three different species of iris flowers. In the first R code chunk the data is loaded, since this step may not be relevant for the reader we omit both the output and the R code.

To print the number of rows and columns for this dataset we use the `\Sexpr()` command in the LaTeX text before displaying a small summary of the data. Afterwards we have R compute the linear model for two variables and print the results using R's `xtable` command, which provides output in valid LaTeX syntax. Therefore we prevent Sweave from displaying the code in verbatim mode by specifying `results=tex`. Finally, the last code chunk plots a scatterplot for the variables `Petal.length` and `Sepal.width`; please note the `fig=true` statement here, specifying that the result is a picture.

2.4 Dynamic reports

The final example shows how reports with dynamic data sources can be created easily. Let's suppose we need a report on the USD/EURO exchange rates on a frequent basis. The data can be downloaded



Der Datensatz hat 5 Spalten und 150 Zeilen.

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.000   1.600   4.350   3.758   5.100   6.900
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	3.4549	0.0761	45.40	0.0000
iris$Petal.Length	-0.1058	0.0183	-5.77	0.0000

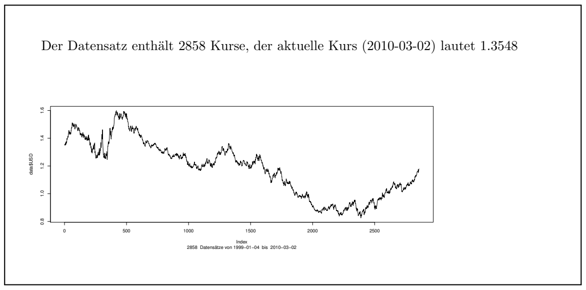
Table 1: Lineares Model von Sepal.Width und Petal.Length

Figure 1: Plot von iris$Petal.Length vs. iris$Sepal.Width

Figure 3: Document generated by Listing 2.3

from the home page of the European Central Bank where it is provided in XML or CSV format. The retrieval process can be controlled from R using the `system()` command calling an external `wget` (standard on Linux/Unix; Windows users may need to install it). To extract the data from the resulting zip file R's internal zip tool is used and the data set stored in the variable *data*. The code is shown in Listing 2.4.

We use the `\Sexpr()` command to print the dimensions of the dataset in our document before plotting a chart (see Figure 4) with the development of the Euro/dollar exchange rate.



Der Datensatz enthält 2858 Kurse, der aktuelle Kurs (2010-03-02) lautet 1.3548

Figure 4: Output of exchange rate example, generated by Listing 2.4

3 Conclusion

R provides a vast set of functions for all aspects of data analysis. Together with Sweave, a user can easily generate dynamic reports holding the results and methods leading to them in one document. With this article I want to encourage everybody to give R a try when facing a data analysis challenge. If any questions or remarks, please feel free to contact me.

```
\documentclass{scrartcl}
\begin{document}
<<echo=f,results=hide>>=
# define width and height of the plot window
windows(width = 8, height = 4)
# use external wget to download the file from
# the ECV and to store it in d.zip
system("wget -O d.zip http://www.ecb.int/
  stats/eurofxref/eurofxref-hist.zip")
# use R's internal zip to extract the file to
# the current directory
zip.file.extract(file="eurofxref-hist.csv",
  zip="d.zip",unzip="",dir=getwd())
# read the data
data= read.csv("eurofxref-hist.csv",sep=",",
  header=TRUE)
@
The data contains \Sexpr{nrow(data)} rates,
  the latest rate (\Sexpr{data$Date[1]}) was
  \Sexpr{data$USD[1]}
\centering
\begin{figure}[ht]
<<fig=true,echo=false,width=15,height=6>>=
# plot the data
plot(data$USD,t="l", sub=paste(nrow(data),"
  datasets from ",data$Date[nrow(data)],"
  until ",data$Date[1]),asp=)
@
\end{figure}
\end{document}
```

Listing 2.4: Sweave code with dynamic data source

References

[1] Michael J. Crawley, *Statistics — An Introduction using R.* Wiley, 2005.

[2] Peter Dalgaard, *Introductory Statistics with R.* Springer-Verlag, 2004.

[3] Uwe Ligges, *Programmieren mit R.* Springer-Verlag, 2008.

[4] John Maindonald and John Brown, *Data Analysis and Graphics Using R.* Cambridge University Press, 2010.

[5] R Core Team, *An Introduction to R.* `http://cran.r-project.org/doc/manuals/R-intro.pdf`

◇ Uwe Ziegenhagen
Lokomotivstr. 9
50733 Cologne, Germany
`ziegenhagen (at) gmail dot com`
`http://www.uweziegenhagen.de`

A web-based TeX previewer: The ecstasy and the agony

Michael Doob

Abstract

The appeal of a web-based combined TeX editor and previewer is instantaneous. It allows not only the easy testing of snippets of code, the writing of short abstracts and even of short papers, but also allows sharing of the results over the web. Unfortunately, even a benign program like TeX presents serious security risks, and care must be used when exposing such an application.

This article describes a web-based viewer of this type. We will:

- Illustrate how remarkably easy it is, using tools readily available, to construct a previewer,
- give examples of potential security problems, and
- indicate some solutions to these problems.

The context of this talk is a LAMP (Linux, Apache, MySQL, PHP) environment, but the basic ideas can be applied to any of the common operating systems.

1 Ecstasy

The appeal of a web-based TeX previewer is immediate. There are many possible reasons for this. We start with a few of the them.

1.1 Motivation

1.1.1 Remote access

We at the Publications Office of the Canadian Mathematical Society receive papers accepted for publication (sometimes called a sow's ear) at many different levels of quality of TeX. They must all be made to conform to our publication standards (sometimes called a silk purse), and significant manpower is used for this purpose. We have a number of editors who work both at our office and at home. There is no problem putting TeX on a home computer. We have our own style file, and that can be put on the home computers too (although it does change from time to time). However, there is a significant problem with our fonts. We have a number of proprietary (Adobe) fonts, and the license restricts their distribution. The TFM files are no problem and can be put on the home computers; the only problem is with the previewing since that uses the proprietary information. Hence a web page previewer with a one-button upload of the TeX file followed by running LaTeX with our class file and then displaying the resulting pages is just what we need.

1.1.2 Abstract submissions

The Canadian Mathematical Society has semiannual meetings in June and December. There are several hundred abstracts for each meeting which need to be in LaTeX format compatible with the style of our proceedings. Our traditional method was to allow presenters to submit their (purported) LaTeX files by email. Changing these sows' ears into silk purses consumes significant resources. With a web page the author can edit the LaTeX file until it works properly with our style file.

We now provide a window into which the abstract may be loaded. It can be run though the appropriate version of LaTeX and, if needed, can be further edited and rerun within the same window. This transfers the editing efforts from our personnel to the author. There is, of course, a resulting decrease in quality due to author inability to use LaTeX optimally. The abstracts are ephemeral (they are used for the one meeting only), and so this is an acceptable cost.

1.1.3 Snippet testing

Sometimes it's desirable to try out a new definition that may take a few tries to get right. If the web server is on a local machine, the turnaround time is instantaneous. It's easy to incrementally improve the code until it is perfect.

Similarly, it is useful to use the picture environment incrementally to create figures that will be usable with any implementation of LaTeX.

If you subscribe to `texhax` (`http://lists.tug.org/texhax`) then lots of little problems that arise from that list can be checked and/or debugged on the spot.

1.1.4 Because we can

The improvements in the speed of software applications used with web browsers over the past few years have been breathtaking. We have long been able to run TeX on a local machine and view the output immediately on a previewer. It is interesting that we can now replicate that experience using even a modest web connection.

1.2 Some nice implementations

There are already several web-based TeX previewers available. Here are some particularly nice examples:

- Troy Henderson's `http://www.tlhiv.org/ltxpreview`
- Jan Přichystal's `http://tex.mendelu.cz/en`
- Jonathan Fine's `http://www.mathtran.org/toys/jfine/editor2.html`

All of these have the same general pattern: A window for typing input, some method of output display, and options that may be chosen using radio buttons or pulldown menus.

1.3 LAMP Implementation

1.3.1 Environment

Our environment used for this application is sometimes called LAMP: the Linux kernel for the operating system, the Apache web server, the MySQL database management system (unused in this application) and the PHP scripting language (sometimes the "P" is Perl or Python; indeed, either could be used instead of PHP). No extra modules are used with Apache, and no additional packages are loaded into PHP. In addition, no JavaScript is used.

1.3.2 Desired elements

The minimum implementation would usually display input (an input window using direct typing, cut-and-paste or file upload), as well as output that is dependent on the success or failure of the TeX job. In addition, it's also easy to have file uploads only and to display (portions of) the log file.

It's also possible to preload TeX input or specific packages. For example, it might be more convenient to have the material in the input window automatically inserted within:

```
\documentclass{article}
\begin{document}

\end{document}
```

if all of the TeX files will be using `article.cls` and no other packages are needed. Similarly, it's also easy to preload either document classes or packages using pulldown menus. Examples are given in the documentation.

1.3.3 Browser peculiarities

Ideally the output should be rendered identically by different browsers. This ideal, unfortunately, is not met. For example, the output from rerunning TeX should reflect the content in the current input window. In fact, there is an HTML metacommand for exactly this purpose:

```
<META HTTP-EQUIV="CACHE-CONTROL"
      CONTENT="NO-CACHE">
```

Alas, some browsers will ignore this, but these shortcomings can be overcome in a LAMP environment. It's always possible to generate unique names with each call to TeX to avoid the cache problem. It's also possible to use freely available software to generate output (`png`, `jpg`, `pdf` or `svg`) whose renderings will be (more or less) browser independent.

2 Agony

As can be seen in the accompanying documentation, it's easy to set up a web-based TeX previewer within a LAMP environment. Alas, as with any web application that may be accessed widely, there are certain concerns and possible exploits that must be addressed. At first blush, TeX is pretty robust and locks out the most dangerous threats. For example, there are no direct system calls available. Nonetheless, there are precautions that must be taken. Examples follow to illustrate these problems, roughly in increasing order of vulnerability.

2.1 The need to know principle

Clearly, the more widespread the audience is for a web application, the less is the information that should be disclosed about the operating environment. There are two options: control the access to the web pages to reduce the risk or control the amount of information disclosed. In a LAMP environment both are easy.

It is a standard configuration command for the Apache server to restrict access to some (or even all) directories to clients with specific Internet addresses, so the access, if desired, may be localized. Greater restriction of access may (or may not!) reduce the risk of system compromise.

On the other hand, if there is widespread access, then the log file, even when there is only one line of TeX input, will reveal information about the operating system:

```
This is TeX, Version 3.14159 (Web2C 7.4.5)
/usr/share/texmf/tex/latex/base/size10.clo
```

In this case, the structure of the file system is revealed; it has files in a position (under `/usr/share`) that indicates an installation via a package manager on a Unix system rather than a `texlive` installation or some other operating system, and as such it gives hints to the location of the vulnerabilities that any operating environment possesses. Loading more packages and fonts generates similar messages concerning the versions running and the structure of the file system. These may and should be filtered out when the log file is requested. This same is true for error messages.

2.2 Denial of service

A more serious problem is that of Denial of Service (DoS) attacks. These are designed to utilize all of the resources available on a particular computer and thus deny access by others. There are several methods by which this may be done.

2.2.1 CPU hogging

Consider what happens with the following LaTeX input:

```
\newcounter{cnt}
\loop
   \stepcounter{cnt}
   \ifnum \value{cnt}<500000
\repeat
```

There could hardly be a simpler loop construct. Running it will do nothing but increment the counter from 0 to 500000 and then quit. This takes a few seconds. If you use a utility (like `top`) to check CPU usage while this is running, you will find it maxed out. If the `\stepcounter{cnt}` is deleted, TeX will run indefinitely, eating up all available CPU resources. As a further insult, the PHP call will freeze the browser, so no termination is possible, even if the program were run by innocent error. Ouch!

Here is another example:

```
\newcounter{cnt}
\loop
   \thecnt\newpage \stepcounter{cnt}
   \ifnum \value{cnt}<10000
\repeat
```

This produces a 10,000 page document with one integer on each page (actually two if you include the page number). Suppose the `\stepcounter{cnt}` is left out. Then the loop is infinite, and TeX happily runs until it reaches its memory limit and then halts. This indicates the following: as long as the loop is doing anything that uses memory there will be a graceful failure in an accidental infinite loop.

What can be done to keep infinite loops from eating up inappropriate resources? There are at least two remedies for this:

- Any standard implementations of Linux comes with the `pam` (pluggable authentication module) software. This module uses a file called `limits.conf` to control, among other things, the amount of CPU time any process can use.
- For operating systems without `pam` there is a program called `cpulimit` which may be used to control the percentage of available CPU resources that may be allocated to a given process.

2.2.2 Disk hogging

Now consider the following LaTeX input:

```
\newcounter{cnt}
\loop
   \leavevmode\newpage \stepcounter{cnt}
   \ifnum \value{cnt}<10000
\repeat
```

This produces a 10,000 page document with only the page numbers on each page (of course, the use of `\pagestyle{empty}` will make the page completely blank). If we delete the `\stepcounter{cnt}` from the input, then TeX runs indefinitely using no memory, but the DVI file will (apparently) grow without limit.

This problem is easy to address. The file mentioned above, `limits.conf`, can also control disk usage. Alternatively, disk quotas, turned off by default, may be enabled.

2.2.3 Server hogging

Any web application is subject to attack through the server. A distributed DoS attack, that is, one from a botnet of many clients, is really impossible to stop. Even with web pages, the mouse clicks can be spoofed, so it is important to keep the web applications isolated from the rest of the computer environment. One possibility is to have users register and log into the environment that runs the web-based browser software.

2.3 PHP attacks

In a recent paper [1], Stephen Checkoway, Hovav Shacham, and Eric Rescorla have pointed out a significant vulnerability in the writing and subsequent rereading of PHP scripts. Consider the following code:

```
\newwrite\bummerfile
\openout\bummerfile=badfile.php
\write\bummerfile{<?php}
\write\bummerfile{echo passthru("date");}
\write\bummerfile{phpinfo();}
\write\bummerfile{echo
          passthru("cat /etc/passwd");}
\write\bummerfile{?>}
```

This opens a file called `badfile.php` in the same directory where the DVI file is written and writes in it five lines of PHP code. These implement three commands: a listing of the current time (a typical system call), a listing of all the PHP parameters on the system (a clear violation of need-to-know), and finally a listing of the password file. It should be noted that the subdirectory from where the reading of files is done by the Apache server is easily obtained from the listing of the page source. It typically will have something like `<img src="jail/temp128053010.png">` within it, indicating in this case that the subdirectory being read is called `jail`. Thus by adding `jail/badfile.php` to the original `http` address, the file `badfile.php` is executed.

This vulnerability may be addressed in several ways: the directory from which the image files are read can be separated from the one where the TEX program writes. Alternatively, if a directory contains an `.htaccess` file which in turn contains a line `php_flag engine off`, then PHP files will not be run from that directory (note that this feature is disabled by default and must be enabled in the Apache configuration file).

2.4 Isolation

Putting any application on the web, as we have seen, has inherent dangers. While these can not be eliminated, they can be somewhat mitigated by isolating the web application, inasmuch as possible, from the rest of the computer environment. There are several possible approaches.

2.4.1 Single computer

The Apache server has a configuration file that is read when the server starts (often called `httpd.conf`). It allows the server to start at different locations in the file system depending on the calling IP address. In particular the address `127.0.0.1` (also known as `localhost`) is always reserved for the local computer. Setting up a virtual host for that address can ensure that the files are not accessible from any outside address. If you need a web-based TEX previewer to be used by many people on one computer, this is a safe method of implementation.

2.4.2 Small sets of users

The configuration file for the Apache server can also be used to restrict the server to predefined IP addresses. Alternatively, pages can be password protected.

2.4.3 Chroot jail

The `chroot` command is available on all Unix implementations. Copies of all the software (binaries and libraries) needed for the application are put in one directory, and the `chroot` command then limits the operating system access to that directory (and its subdirectories) only. We say that the operating system is in a chroot jail. This makes the rest of the computer environment safe even if the application is broken.

Running the Apache server in a chroot jail will protect the rest of the operating system. In fact a script may be set up to create the jail automatically. If the software in the jail seems questionable, a new copy can be reconstructed.

2.4.4 Software isolation of the operating system

An even stronger form of isolation is to run the Apache server under its own operating system. It is now fairly easy to set up virtual computers within a Unix environment. It's then possible to take a snapshot of the original implementation of the operating system and then refresh the installation regularly. This means that any damage can be easily repaired.

2.4.5 Hardware isolation

The most extreme measure is to put the application on its own platform. This is in effect running the web application as an embedded device. Since a web browser can be run headless, the costs are actually quite modest. It is possible, for example, to set up a mini-ITX board with an enclosure, RAM and storage for about $150.

3 Documentation

Finally, we want the actual PHP code that implements the web-based previewer. This is included in the TEX file [2]. Running the file through LATEX prints the documentation along with instructions for extracting the PHP code.

This code has worked properly with all browsers tested (Firefox, Safari, Internet Explorer, Chrome, Opera). Nonetheless, it should be considered as a starting point. It is hoped that it may be improved by making it more robust and, hopefully, not be compromised by the types of attacks given in this paper.

References

[1] Stephen Checkoway, Hovav Shacham, Eric Rescorla. Are Text-Only Data Formats Safe? `http://cseweb.ucsd.edu/~hovav/dist/texhack.pdf`

[2] Michael Doob. A web-based TEX previewer — Sources. `http://tug.org/TUGboat/31-2/doob-texwebviewer.tex`

◇ Michael Doob
Department of Mathematics
The University of Manitoba
Winnipeg, Manitoba R3N 2T2
Canada
mdoob (at) ccu dot umanitoba dot ca

Qurʾānic typography comes of age: Æsthetics, layering, and paragraph optimization in ConTEXt

Idris Samawi Hamid

1 The background of Oriental TEX

Attempts to integrate scripts beyond the Latin into the TEX universe are nearly as old as TEX itself. In the case of Right-to-Left (RTL) scripts as Arabic script, the inadequacy of the original TEX to the task was pointed out bluntly by Knuth himself back in 1987.[1] Since then, the heroic efforts of the ArabTEX and Omega projects took large strides in the way of extending TEX to support Arabic-script typesetting. On the other hand, by the early 2000s the realization of a paradigm capable of capturing the fullness of the Arabic script and its sophistication still seemed a long ways away. Combined with other challenges, e.g., critical-edition typesetting, so much work remained to be done.

In the winter of 2005–6, this author, along with Hans Hagen and Taco Hoekwater, initiated a very ambitious attempt to address the challenges of Arabic-script and critical-edition typesetting in the context of a radical overhaul and extension of TEX that would affect and potentially benefit virtually every corner of the TEX paradigm. This overarching context constitutes the ongoing LuaTEX project. By virtue of a major grant from Colorado State University in the spring of 2006, since extended by the generosity of US TUG, DANTE, and private donors, Oriental TEX has both served as the midwife of LuaTEX as well as having reached major mileposts in its particular goals pertaining to Arabic-script and critical-edition typesetting and typography.

2 Mileposts in the Oriental TEX project

2.1 The Qurʾān test

The most relevant torture test of Arabic-script typesetting and typography involves capturing the nuances of the Arabic used in traditional Qurʾānic script, a task that involves much by way of multiple layers of diacritics, paragraph optimizations using stretching and shape alternates, as well as multilayered coloring.

The Oriental TEX project is proud to announce that it has reached the milepost of being able to represent these aspects of Qurʾānic typography, marking a major outward milepost in the forward movement of Oriental TEX. There is still a ways to go, but in the current visual results we can confidently say that we are "over the hump" so to speak.

[1] See p. 157 of Knuth's *Digital Typography*, CSLI Publications, Stanford, 1987.

2.2 Infrastructure

The visuals achieved by Oriental TEX build on an extensive infrastructure, involving the following mileposts, both achieved and in progress:

- Aleph + pdfε-TEX + native UTF-8 + Lua = LuaTEX (done);
- OpenType + language processing in Lua ⇒ ConTEXt MkIV (beta)
- Development of Husayni font family (flagship font nearly complete);
- Paragraph optimization and justification model (initial implementation);
- Bidirectional model (under development — no more naive mirroring primitives but a full typographical framework for bidi);
- Structural-element/critical-apparatus control ⇒ CriTEXt (under development)

2.3 Documentation

In addition to the numerous papers, presentations, and other documentation by Hans Hagen, I am working on the following documentation tasks for a wider audience:

- *An Ontology of Arabic-script Typography* (in development);
- *CriTEXt: The Critical Edition Module for ConTEXt* (white paper available, unpublished)
- *Typographical Æsthetics and Engineering: Structured and Automated Authoring in ConTEXt* (in development: this will be perhaps the first book on ConTEXt for a general audience)

3 Qurʾānic typography

Qurʾānic typography involves getting the following tasks done.

3.1 Control of æsthetics

Given an Arabic string, there is often more than one way to æsthetically represent it. Which way one chooses depends on the context. This goes beyond the mere availability of glyph alternates as in Latin. Put another way, just as one must choose the right font for a given typographical task in Latin script, so also must one choose the right *set of æsthetic features* for a given Arabic-script task. This principle can be applied to Latin, especially in the form of calligraphy fonts, but it's nowhere near as important an issue.

Available Arabic fonts generally mimic Latin fonts in that they have little-to-no flexibility in this regard. That is, given a font, its æsthetic style is generally fixed. OpenType allows for far more flexibility of æsthetic sets within Arabic-script, providing more culturally authentic possibilities.

In ConTEXt MkIV, we have something called *fontfeatures*. Using fontfeatures, we can define, add, and subtract *sets* of OpenType lookups — called *features* to create myriads of output possibilities.

In the case of Qurʾānic typography, we will illustrate a default feature set, and see what happens when we

- subtract and add features,
- stretch glyphs using alternates.

3.2 Layering

Arabic script is mostly consonantal. The vowels are generally not letters.[2] And the consonants are characterized by ambiguity: given a letter shape, it may represent two or more actual letters. One disambiguates by means of *identity marks*, mostly in the form of dots.

In historical texts, including the Qurʾān, sometimes only the consonantal layer is represented in the text, sometimes only either the vowel or the identity-marks layer. Sometimes the two layers clash and we have to adjust the shape of the consonant or the mutual positions of characters belonging to the two diacritics layers.

3.3 Paragraph optimization

Given that Arabic script in general does not accept hyphenation, getting even color in a paragraph will use alternate glyph substitution, or stretched glyph substitution. The first involves an entirely different shape, e.g.,

ك versus ك

whereas the latter involves a stretch of the existing character, e.g.,

الحمد versus الحمــد

One can also combine this with changes in æsthetics, e.g.,

الحمد versus الحـمـد

Getting all of this under control is still in the experimental stages of implementation. The plethora of possibilities for optimization, plus the good job that the current paragraph builder does, leads us to focus on a line-by-line optimization after the initial paragraph is optimized by TEX. We illustrate this by showing some real-life samples from the Qurʾān: from Sūraḧ's Fātiḥaḧ and Baqaraḧ.

[2] There are three consonants that sometimes function as vowels as well, analogous to the letter 'y' in English.

4 Æsthetics and layering

4.1 Fontfeatures: default

Given a font, it should have a default æsthetic behavior. To that end, we define a default fontfeature set:

```
\definefontfeature[husayni-default]
 [analyze=yes, mode=node, language=dflt,
  script=arab, ccmp=yes, init=yes, medi=yes,
  fina=yes, rlig=yes, calt=yes, salt=yes,
  anum=yes, ss01=yes, ..., ss60=yes, ...,
  js16=yes, kern=yes, curs=yes, mark=yes,
  mkmk=yes, tlig=yes,
  colorscheme=husayni:default]
```

For illustration we have shown excerpts from the list of dozens of possible features. Features to be disabled can be commented out, or 'yes' changed to 'no'.

Also note the `colorscheme` key at the end. This refers to a mechanism for the coloring of the various layers of the text as we mentioned earlier.

4.2 Subtracting æsthetics

Let's get a barebones, minimalist Arabic-script implementation by subtracting features. Thus the key command here is `\subff`:

```
\switchtobodyfont[husayni-default,40pt]
\definefontfeature [first_order]
[script=arab,dlig=yes,ss01=yes,
 ss03=yes,ss07=yes,ss10=yes,ss12=yes,
 ss15=yes,ss16=yes,ss19=yes,ss24=yes,
 ss25=yes,ss26=yes,ss27=yes,ss31=yes,
 ss34=yes,ss35=yes,ss36=yes,ss37=yes,
 ss38=yes,ss41=yes,ss42=yes,ss43=yes]
\subff{first_order}
```

Our ConTEXt input text, typed in Arabic (omitted in other examples):

```
العالمين اَلْعَالَمِيْنَ \crlf
مُتَهَجِّدٌ سَيَنْفَجِرُ لُواتيخ
```

And the result:

العالمين اَلْعَالَمِيْنَ

مُتَهَجِّدٌ سَيَنْفَجِرُ لُواتيخ

4.3 Default æsthetics

Let's show the default æsthetics:

```
\switchtobodyfont[husayni-default,40pt]
```

The result:

العالمين اَلْعَالَمِيْنَ

مُتَهَجِّدٌ سَيَنْفَجِرُ

لُواتيخ

Study the differences between the first two words once we introduce vowels: one of them has a mild stretch to accommodate a lower vowel.

4.4 Advanced æsthetics

For more fancy text, let's add some features. Note the stacking of characters, giving in some cases a slanted feel to the text. The key command here is `\addff`:

```
\switchtobodyfont[husayni-default,40pt]
\definefontfeature [stack:haa:multi-level]
[script=arab,ss05=yes,ss06=yes,ss09=yes,
 ss13=yes,ss17=yes,ss40=yes]
\addff{stack:haa:multi-level}
```

اَلْعَلَمِيْنَ مُتَهَجِّدٌ سَيَنْفَجِرُ

لُواتيخ

4.5 Stretched and alternate æsthetics

Again, we are adding, this time adding stretched-alternate and glyph-alternate features:

```
\switchtobodyfont[husayni-default,40pt]
\definefontfeature [maximal_stretching]
 [script=arab,ss05=yes,ss09=yes,ss06=yes,
 ss13=yes,ss17=yes,ss40=yes,js13=yes,js14=yes,
 js16=yes,js05=yes]
\addff{maximal_stretching}
```

اَلْعَلَمِيْنَ مُتَهَجِّدٌ سَيَنْفَجِرُ

لُواتيخ

This example in part illustrates that TEX is informed about where it is legal to stretch and where it is illegal. This word, third from the right in the above example, shows three stretches; in real life we would use only one of those three for any given stretched instantiation of the word.

سينفجر

4.6 The base consonant layer

Here we get the base consonants only:

```
\switchtobodyfont[husayni-default,40pt]
\definefontfeature [consonant]
 [script=arab,ss61=yes,ss49=yes,ss52=yes]
\addff{consonant}
```

العلمٮں مٮهحد سٮٮڡحر

لواٮٮح

4.7 The identity marks layer

Let's disambiguate the characters, and use colors to illustrate the difference:

```
\switchtobodyfont[husayni-default,40pt]
\setfontcolorscheme[1]
\definefontfeature [identity]
 [script=arab,ss49=yes,ss52=yes]
\addff{identity}
```

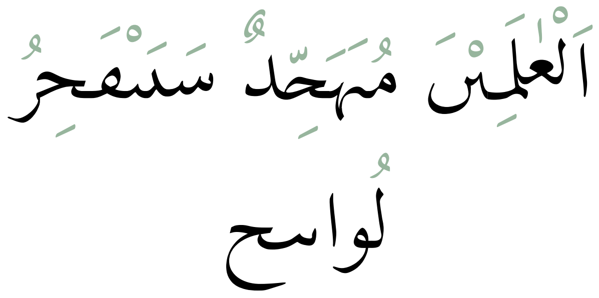
العلمين متهجد سينفجر

لواتيخ

4.8 The vowels layer

In the history of the Qurʾānic text, the vowels were developed before the identity marks, so vowels without identity marks is not an abstract example:

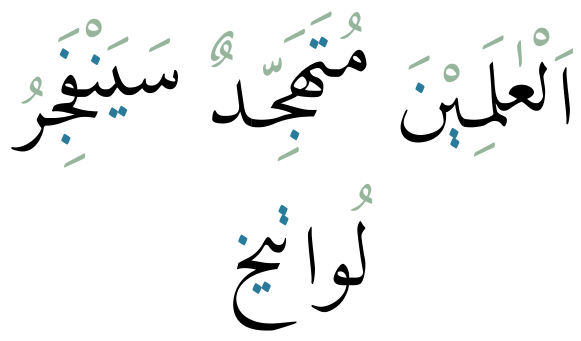
```
\switchtobodyfont[husayni-default,40pt]
\setfontcolorscheme[1]
\definefontfeature [vowel]
 [script=arab,ss61=yes]
\addff{vowel}
```



4.9 Full layering

Let's put it all together, with full layering and color distinctions:

```
\switchtobodyfont[husayni-default,40pt]
\setfontcolorscheme[1]
```



5 Optimization and sample Qurʾānic typography

5.1 Sūraḧ Fātiḥaḧ

Let's look at the first page of the handwritten frontispiece of the standard Egyptian edition of the Qurʾān. The entire edition is typeset, except for the two first pages that constitute the frontispiece (see Figure 1).

As simple as the handwritten version seems, it contains numerous subtleties that make imitation using traditional typesetting technology virtually impossible. Figure 2 presents it in Oriental TEX, with full layering.

But we can be creative and imitate other styles as well. An Iranian style is shown in Figure 3, and an Indo-Pakistani style in Figure 4.

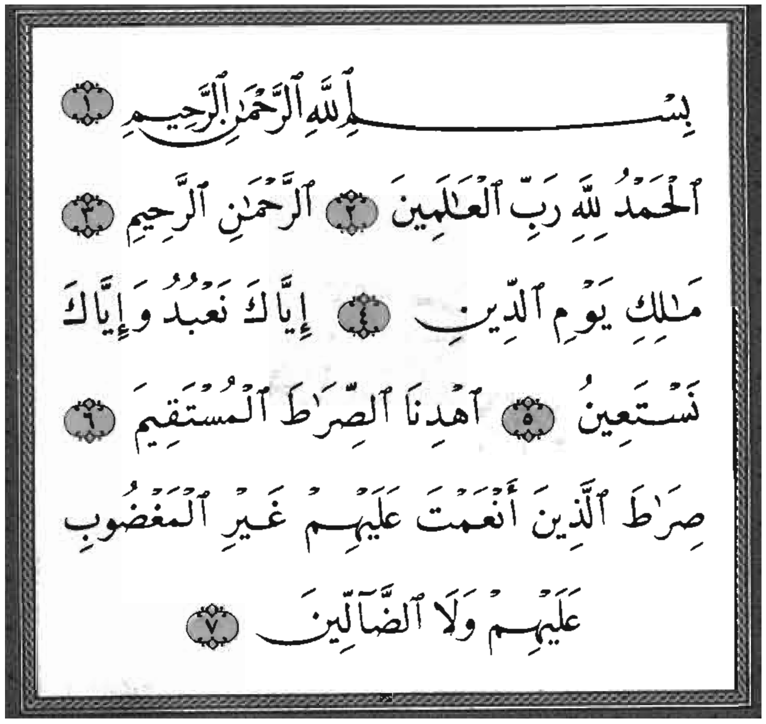
5.2 Sūraḧ Baqaraḧ

From the frontispiece, we move to more straightforward paragraph typography. We illustrate this with two æsthetic sets from Sūraḧ Baqaraḧ, with optimization applied to get square paragraphs that fill the page, as well as full layering (Figures 5 and 6).

6 Conclusion

As you can see, Oriental TEX has come a long way. There is much work ahead, including further refinements to the æsthetics and the optimization routines, but it is certainly gratifying to see such sophisticated real-world results. Qurʾānic typography has finally come of age!

◇ Idris Samawi Hamid
Colorado State University
ishamid (at) colostate dot edu



بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ ١
ٱلۡحَمۡدُ لِلَّهِ رَبِّ ٱلۡعَٰلَمِينَ ٢ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ ٣
مَٰلِكِ يَوۡمِ ٱلدِّينِ ٤ إِيَّاكَ نَعۡبُدُ وَإِيَّاكَ
نَسۡتَعِينُ ٥ ٱهۡدِنَا ٱلصِّرَٰطَ ٱلۡمُسۡتَقِيمَ ٦
صِرَٰطَ ٱلَّذِينَ أَنۡعَمۡتَ عَلَيۡهِمۡ غَيۡرِ ٱلۡمَغۡضُوبِ
عَلَيۡهِمۡ وَلَا ٱلضَّآلِّينَ ٧

Figure 1: Sūraḧ Fātiḥaḧ, from the 1924 Egyptian Edition of the Qurʾān.

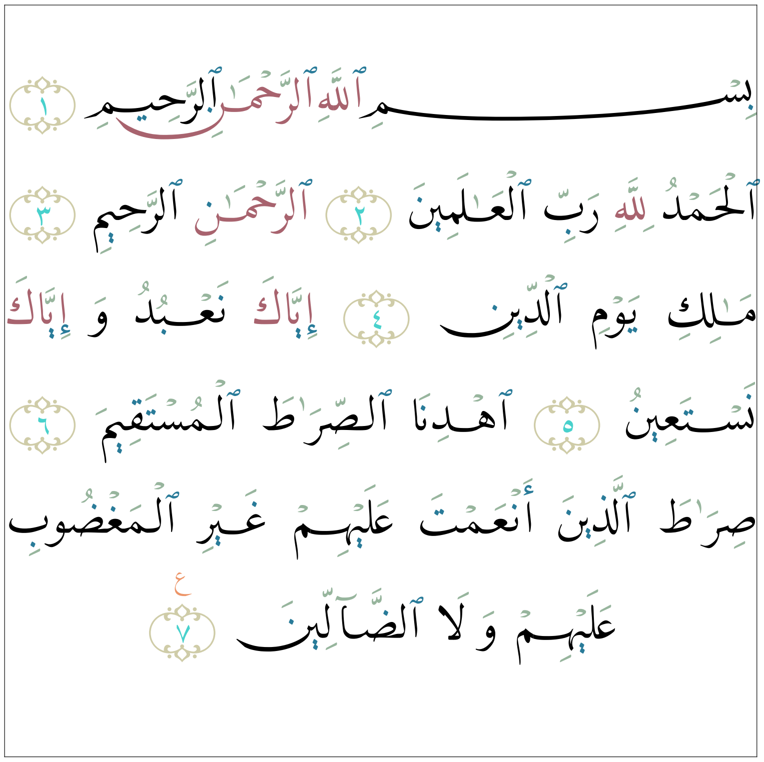

بسم ٱللّه ٱلرّحمن ٱلرّحيم ١
ٱلحمد لله ربّ ٱلعلمين ٢ ٱلرّحمن ٱلرّحيم ٣
ملك يوم ٱلدّين ٤ إيّاك نعبد و إيّاك
نستعين ٥ ٱهدنا ٱلصّرط ٱلمستقيم ٦
صرط ٱلّذين أنعمت عليهم غير ٱلمغضوب
ع
عليهم و لا ٱلضّآلّين ٧

Figure 2: Sūraȟ Fātiḥaȟ, based on the 1924 Egyptian Edition of the Qurʾān, done in Oriental TEX.

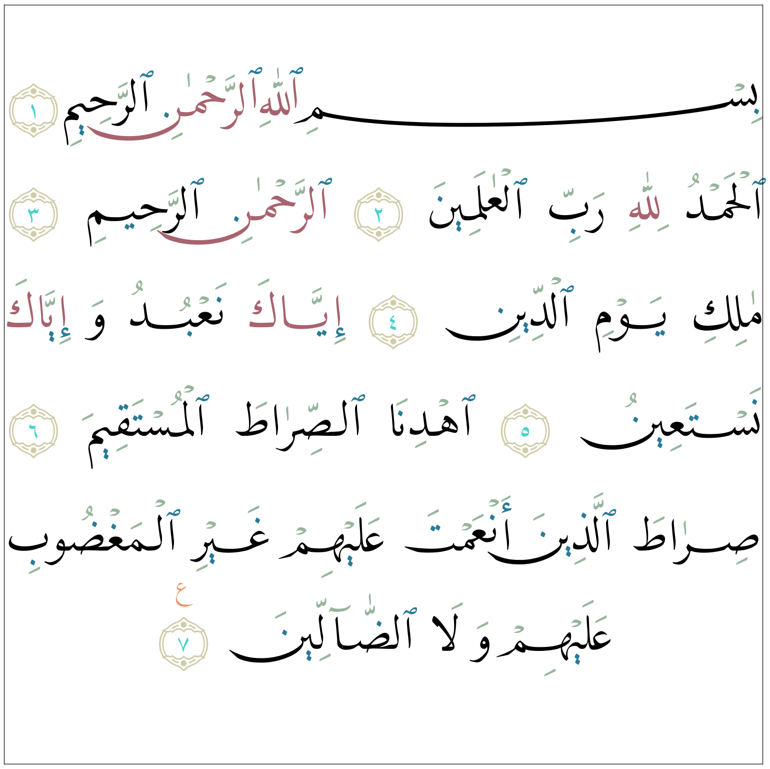
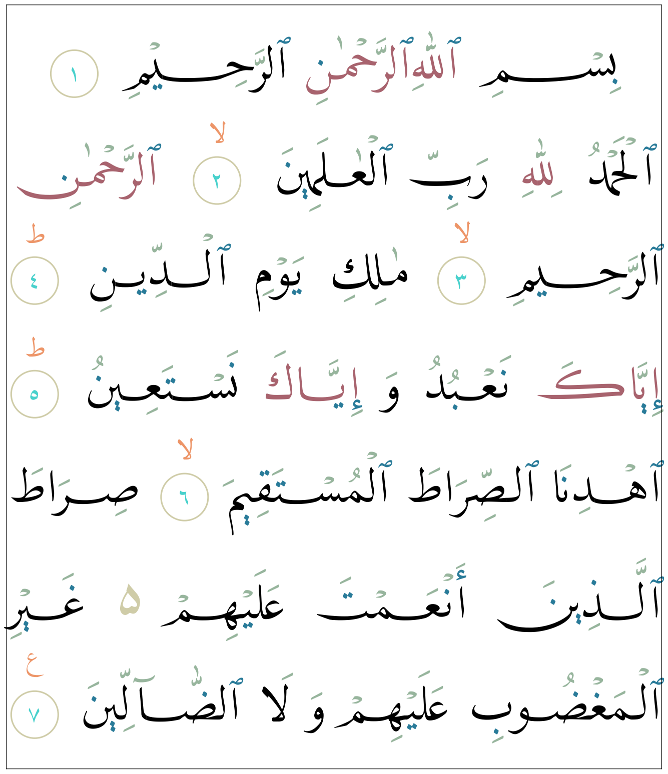
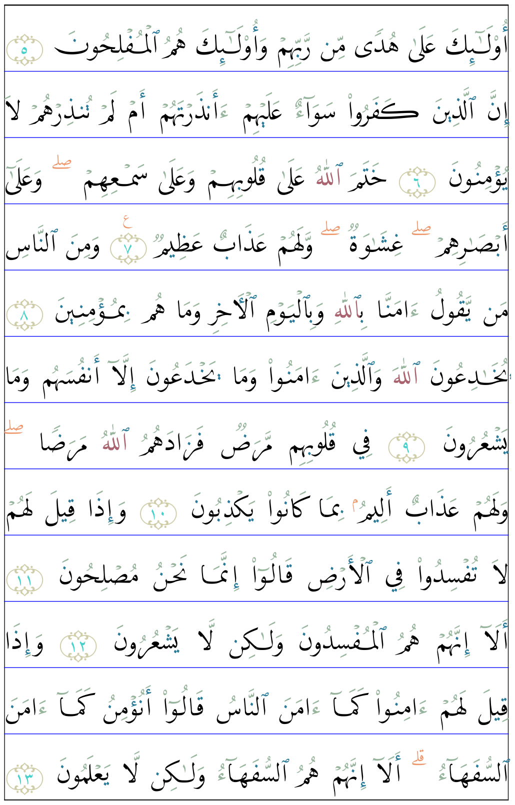

بسم ٱللّه ٱلرّحمن ٱلرّحيم ١
ٱلحمد لله ربّ ٱلعلمين ٢ ٱلرّحمن ٱلرّحيم ٣
ملك يوم ٱلدّين ٤ إيّاك نعبد و إيّاك
نستعين ٥ ٱهدنا ٱلصّراط ٱلمستقيم ٦
صراط ٱلّذين أنعمت عليهم غير ٱلمغضوب
ع
عليهم و لا ٱلضّآلّين ٧

Figure 3: Sūraȟ Fātiḥaȟ, in a more Iranian style.



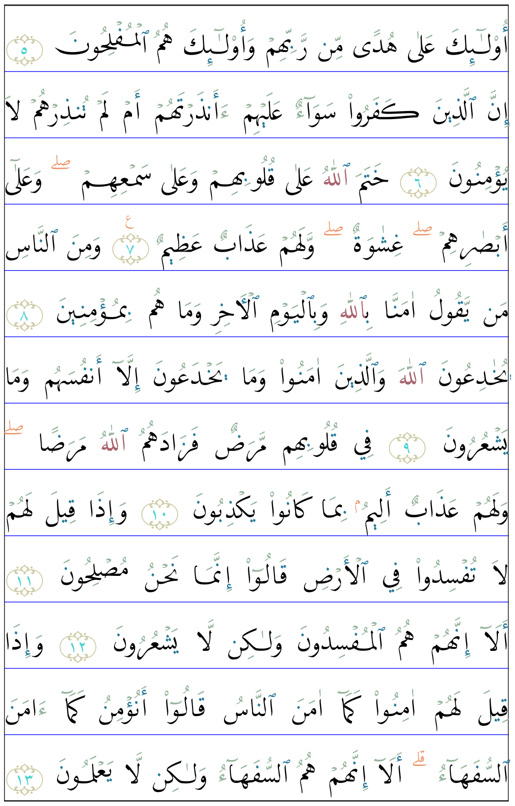
بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ ١
ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَٰلَمِينَ ٢ لا ٱلرَّحْمَٰنِ
ٱلرَّحِيمِ ٣ لا مَٰلِكِ يَوْمِ ٱلدِّينِ ٤ ط
إِيَّاكَ نَعْبُدُ وَ إِيَّاكَ نَسْتَعِينُ ٥ ط
ٱهْدِنَا ٱلصِّرَٰطَ ٱلْمُسْتَقِيمَ ٦ لا صِرَٰطَ
ٱلَّذِينَ أَنْعَمْتَ عَلَيْهِمْ ٥ غَيْرِ
ٱلْمَغْضُوبِ عَلَيْهِمْ وَ لَا ٱلضَّآلِّينَ ٧ ع

Figure 4: Sūrah Fātiḥah, in a more Indo-Pakistani style.



أُولَٰٓئِكَ عَلَىٰ هُدًى مِّن رَّبِّهِمْ وَأُولَٰٓئِكَ هُمُ ٱلْمُفْلِحُونَ ٥
إِنَّ ٱلَّذِينَ كَفَرُوا۟ سَوَآءٌ عَلَيْهِمْ ءَأَنذَرْتَهُمْ أَمْ لَمْ تُنذِرْهُمْ لَا
يُؤْمِنُونَ ٦ خَتَمَ ٱللَّهُ عَلَىٰ قُلُوبِهِمْ وَعَلَىٰ سَمْعِهِمْ ۖ وَعَلَىٰٓ
أَبْصَٰرِهِمْ ۖ غِشَٰوَةٌ ۖ وَلَهُمْ عَذَابٌ عَظِيمٌ ٧ ع وَمِنَ ٱلنَّاسِ
مَن يَقُولُ ءَامَنَّا بِٱللَّهِ وَبِٱلْيَوْمِ ٱلْءَاخِرِ وَمَا هُم بِمُؤْمِنِينَ ٨
يُخَٰدِعُونَ ٱللَّهَ وَٱلَّذِينَ ءَامَنُوا۟ وَمَا يَخْدَعُونَ إِلَّآ أَنفُسَهُمْ وَمَا
يَشْعُرُونَ ٩ فِي قُلُوبِهِم مَّرَضٌ فَزَادَهُمُ ٱللَّهُ مَرَضًا ۖ
وَلَهُمْ عَذَابٌ أَلِيمٌۢ بِمَا كَانُوا۟ يَكْذِبُونَ ١٠ وَإِذَا قِيلَ لَهُمْ
لَا تُفْسِدُوا۟ فِي ٱلْأَرْضِ قَالُوٓا۟ إِنَّمَا نَحْنُ مُصْلِحُونَ ١١
أَلَآ إِنَّهُمْ هُمُ ٱلْمُفْسِدُونَ وَلَٰكِن لَّا يَشْعُرُونَ ١٢ وَإِذَا
قِيلَ لَهُمْ ءَامِنُوا۟ كَمَآ ءَامَنَ ٱلنَّاسُ قَالُوٓا۟ أَنُؤْمِنُ كَمَآ ءَامَنَ
ٱلسُّفَهَآءُ ۗ أَلَآ إِنَّهُمْ هُمُ ٱلسُّفَهَآءُ وَلَٰكِن لَّا يَعْلَمُونَ ١٣

Figure 5: Sūrah Baqarah, in a basic Egyptian style.



أُولَٰٓئِكَ عَلَىٰ هُدًى مِّن رَّبِّهِمْ وَأُولَٰٓئِكَ هُمُ ٱلْمُفْلِحُونَ ٥
إِنَّ ٱلَّذِينَ كَفَرُوا۟ سَوَآءٌ عَلَيْهِمْ ءَأَنذَرْتَهُمْ أَمْ لَمْ تُنذِرْهُمْ لَا
يُؤْمِنُونَ ٦ خَتَمَ ٱللَّهُ عَلَىٰ قُلُوبِهِمْ وَعَلَىٰ سَمْعِهِمْ ۖ وَعَلَىٰٓ
أَبْصَٰرِهِمْ ۖ غِشَٰوَةٌ ۖ وَلَهُمْ عَذَابٌ عَظِيمٌ ٧ ع وَمِنَ ٱلنَّاسِ
مَن يَقُولُ ٰامَنَّا بِٱللَّهِ وَبِٱلْيَوْمِ ٱلْءَاخِرِ وَمَا هُم بِمُؤْمِنِينَ ٨
يُخَٰدِعُونَ ٱللَّهَ وَٱلَّذِينَ ٰامَنُوا۟ وَمَا يَخْدَعُونَ إِلَّآ أَنفُسَهُمْ وَمَا
يَشْعُرُونَ ٩ فِي قُلُوبِهِم مَّرَضٌ فَزَادَهُمُ ٱللَّهُ مَرَضًا ۖ
وَلَهُمْ عَذَابٌ أَلِيمٌۢ بِمَا كَانُوا۟ يَكْذِبُونَ ١٠ وَإِذَا قِيلَ لَهُمْ
لَا تُفْسِدُوا۟ فِي ٱلْأَرْضِ قَالُوٓا۟ إِنَّمَا نَحْنُ مُصْلِحُونَ ١١
أَلَآ إِنَّهُمْ هُمُ ٱلْمُفْسِدُونَ وَلَٰكِن لَّا يَشْعُرُونَ ١٢ وَإِذَا
قِيلَ لَهُمْ ٰامِنُوا۟ كَمَآ ٰامَنَ ٱلنَّاسُ قَالُوٓا۟ أَنُؤْمِنُ كَمَآ ءَامَنَ
ٱلسُّفَهَآءُ ۗ أَلَآ إِنَّهُمْ هُمُ ٱلسُّفَهَآءُ وَلَٰكِن لَّا يَعْلَمُونَ ١٣

Figure 6: Sūrah Baqarah, in a slightly different vowelization style, with more character stacking.

Asymptote: Interactive TeX-aware 3D vector graphics

John C. Bowman

Abstract

Asymptote is a powerful descriptive vector graphics language for technical drawing recently developed at the University of Alberta. It attempts to do for figures what (LA)TeX does for equations. In contrast to METAPOST, Asymptote features robust floating-point numerics, high-order functions, and a C++/Java-like syntax. It uses the simplex linear programming method to resolve overall size constraints for fixed-sized and scalable objects. Asymptote understands affine transformations and uses complex multiplication to rotate vectors. Labels and equations are typeset with TeX, for professional quality and overall document consistency.

The feature of Asymptote that has caused the greatest excitement in the mathematical typesetting community is the ability to generate and embed inline interactive 3D vector illustrations within PDF files, using Adobe's highly compressed PRC format, which can describe smooth surfaces and curves without polygonal tessellation. Three-dimensional output can also be viewed directly with Asymptote's native OpenGL-based renderer. Asymptote thus provides the scientific community with a self-contained and powerful TeX-aware facility for generating portable interactive three-dimensional vector graphics.

1 Introduction

Notable enhancements have recently been made in the TeX-aware vector graphics language Asymptote.[1] This article provides an overview of those advances made since the publication of articles in *TUGboat* that describe Asymptote's 2D [1] and 3D [2] typographic capabilities. Some of these advances were developed in preparation for and during TeX's 2^5 anniversary workshop in San Francisco. These improvements are contained in the current release (2.03) of Asymptote.

2 Batching of 3D TeX

A significant improvement was made in the processing of 3D TeX labels: their conversion into surfaces is now batched, resulting in much faster execution.

In two dimensions, Asymptote uses a two-stage system to position TeX labels within a figure. First, a bidirectional TeX pipe is used to query the width, height, and depth of a TeX string. This information is used to align the label within a TeX layer on top of a PostScript background. The PostScript background and TeX layer are linked together within a file that is then fed to TeX for final processing. In other words, in two dimensions all that Asymptote really does is prepare a TeX file, deferring typesetting issues to the external TeX engine.

Since TeX is inherently a two-dimensional program, the above scheme will clearly not work in three dimensions. As described in [2], Asymptote uses a PostScript interpreter to extract Bézier paths from the output of TeX+Dvips (or PDFTeX+Ghostscript). Previously, typesetting each three-dimensional label therefore required executing three external processes, drastically slowing down the processing of three-dimensional figures (particularly under the Microsoft Windows operating system).

In most instances, however, the deferred drawing routines [1, 2] do not need detailed Bézier path data in order to size figures, but only the three-dimensional bounding boxes of each label. The only exception is the case where a three-dimensional label needs to be manipulated (e.g. extruded or transformed), a case that in practice arises infrequently. In all other cases, the bounding box may be computed simply by transforming into three dimensions the two-dimensional bounding box reported *via* the bidirectional TeX pipe (which can process many thousands of TeX strings per second).

This allows the conversion of three-dimensional labels into Bézier paths to be deferred until the final conversion of a three-dimensional picture into a fixed-size frame, from which OpenGL calls or PRC code can then be generated. To distinguish the individual path arrays associated with each TeX string within the generated PostScript code, each string is typeset on a separate page. Batching TeX labels in this manner yields remarkable performance gains (typically a factor of two to five faster, depending on the number of TeX labels and the underlying operating system).

3 Billboard labels

By default, three-dimensional labels now behave like "billboards" that interactively rotate to face the camera (fig. 1). This default can be changed locally or globally:

```
import three;
settings.autobillboard=true; // default
currentprojection=perspective(1,-2,1);
draw(unitbox);
label("Billboard",X,red,Billboard);
label("Embedded",Y,blue,Embedded);
```

[1] Andy Hammerlindl, John Bowman, and Tom Prince, available under the GNU Lesser General Public License from http://asymptote.sourceforge.net/

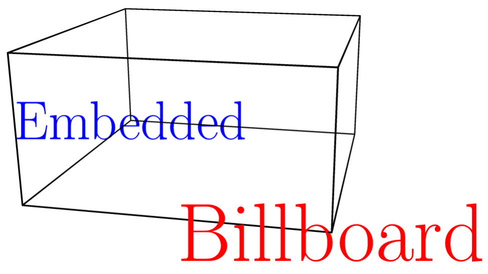


Figure 1: Billboard labels interactively rotate to face the camera, while embedded labels rotate with the picture.

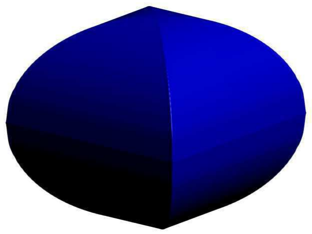
4 Rendering options

Using the latest PRC specification [3], Michail Vidiassov recently overhauled Asymptote's PRC driver, which was originally written by Orest Shardt. The most significant new feature, lossy PRC compression, allows one to produce much more compact 3D PDF files (typically smaller by a factor of two or more). Such specialized rendering options can be specified *via* the structure `render` defined at the beginning of module `three`. The real member `compression` of this structure can be used to set the desired compression value. The real variables `Zero=0.0`, `Low=0.0001`, `Medium=0.001`, and `High=0.01` represent convenient predefined compression values. The default setting, `High`, normally leads to no visible differences in rendering quality. However, when drawing the Bézier approximation to a unit sphere described in [6], PRC compression may create rendering artifacts at the poles and should be disabled:

```
import three;
draw(unitsphere,
     render(compression=Zero,merge=true));
```

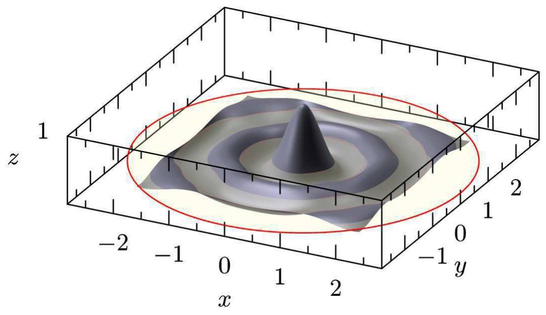
The `merge` argument here is a tri-state boolean variable; the value `true` causes nodes to be merged into a group before Adobe's fixed-resolution rendering mesh is generated. The choice `merge=default` causes only opaque patches to be merged, while the default setting `merge=false` completely disables merging. Patch merging results in faster but lower-quality rendering. It is particularly useful for rendering parametrized surfaces like the volume bounded by two perpendicular unit cylinders centered on the origin:

```
import graph3;
currentprojection=orthographic(5,4,2);
real f(pair z) {
  return min(sqrt(1-z.x^2),sqrt(1-z.y^2));
}
surface s=surface(f,(0,0),(1,1),40,Spline);
transform3 t=rotate(90,O,Z),
           t2=t*t, t3=t2*t;
transform3 i=xscale3(-1)*zscale3(-1);
draw(surface(s,t*s, t2*s,t3*s, i*s, i*t*s,
             i*t2*s, i*t3*s),blue,
     render(compression=Low,closed=true,
            merge=true));
```



This example also illustrates another PRC rendering option, the boolean member `closed`; specifying `closed=true` requests one-sided rendering, whereas the default value `closed=false` requests two-sided rendering.

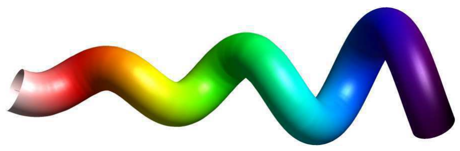
Asymptote now automatically generates a PRC model tree that reflects the object hierarchy, grouping patches together logically, as illustrated in the model tree for the PDF version of the graph below:



One can also manually begin and end a new group called `name` in the 3D model tree for a picture `pic`, using the render options in the structure `render`:

```
void begingroup3(picture pic=currentpicture,
                 string name="",
                 render render=defaultrender);
void endgroup3(picture pic=currentpicture);
```

Various geometric PRC primitives have also been implemented in the current PRC driver. For example, the primitives `drawPRCsphere`, `drawPRCcylinder`, and `drawPRCtube` are useful for drawing and capping compact representations of thick curves (tubes) on a picture. The algorithm for constructing circular tubes was first rewritten to use Oliver Guibé's splined representation of parametrized surfaces, using in the angular direction `splinetype periodic` scaled by $2a$, where the parameter $a = \frac{4}{3}(\sqrt{2}-1)$ is determined by requiring that the third-order midpoint of a cubic Bézier spline lie on the unit circle. This Bézier surface representation is used directly for OpenGL output. For PRC output, it is more efficient to extract from this representation a path that describes the tube center and another path lying on the surface of the tube. These two paths are then passed as arguments to the `drawPRCtube` primitive, which Adobe Reader then renders into a smooth tube, as shown below:



When drawing a three-dimensional dot, the rendering setting `sphere` allows the user to choose between the built-in PRC representation of a sphere (`PRCsphere`) or an efficient NURBS approximation (`NURBSsphere`) to a sphere using 10 distinct control points [4] (the 8-point version discussed in [5] leads to rendering artifacts at the poles). The default, `NURBSsphere`, generates slightly larger files but renders faster than the built-in PRC primitive.

Another new rendering option, which applies to both OpenGL and PRC output, is `labelfill`. Enabled by default, this option allows one to fill subdivision cracks in opaque unlighted (purely emissive) labels, thereby working around artifacts due to the suboptimal algorithms used in Adobe Reader.

5 SVG output

To support web usage, Asymptote now uses Martin Gieseking's excellent `dvisvgm` utility to generate SVG natively (as well as PostScript, PDF, and 3D PRC) vector graphics output. The setting `svgemulation` may be enabled to emulate unimplemented SVG features like Gouraud and tensor-patch shading; otherwise such elements will be replaced by PNG images.

6 Latexmk support

The latest version (1.18) of the `asymptote.sty` package, which allows one to embed Asymptote commands within a LaTeX file, supports both global and local values for the `inline` and `attach` options. It supports John Collins' excellent `latexmk` Perl script for updating only those figures that have changed since the last compilation. One may also specify an `\asydir` subdirectory for Asymptote figures.

7 Conclusion

The Asymptote enhancements described in this article have greatly increased the speed and usability of Asymptote, especially for large documents that contain many three-dimensional figures. They are the result of collaborations among many Asymptote users. In particular, I would like to acknowledge Andy Hammerlindl for designing and implementing much of the underlying Asymptote language, Orest Shardt and Michail Vidiassov for their exceptional work on the PRC driver, Olivier Guibé for his implementation of splined parametric surfaces, Philippe Ivaldi for his implementation of rotation-minimizing frames [7], and Will Robertson and Herbert Schulz for discussions at the TUG 2010 workshop regarding `asymptote.sty`. Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada.

References

[1] John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. *TUGboat: The Communications of the TeX Users Group*, 29(2):288–294, 2008.

[2] John C. Bowman and Orest Shardt. Asymptote: Lifting TeX to three dimensions. *TUGboat: The Communications of the TeX Users Group*, 30(1):58–63, 2009.

[3] ISO/TC171/SC2. 3D use of product representation compact (PRC) format, 2009. `http://pdf.editme.com/files/PDFE/SC2N570-PRC-WD.pdf`.

[4] Kaihuai Qin. Representing quadric surfaces using NURBS surfaces. *Journal of Computer Science and Technology*, 12(3):210–216, 1997.

[5] Kaihuai Qin, Zesheng Tang, and Wenping Wang. Representing spheres and ellipsoids using periodic NURBS surfaces with fewer control vertices. *Computer Graphics and Applications, Pacific Conference on*, 0:210, 1998.

[6] Orest Shardt and John C. Bowman. Surface parametrization of nonsimply connected planar Bézier regions. *Submitted to Computer-Aided Design*, 2010.

[7] Wenping Wang, Bert Jüttler, Dayue Zheng, and Yang Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27(1):1–18, 2008.

◇ John C. Bowman
Dept. of Mathematical and Statistical Sciences
University of Alberta
Edmonton, Alberta
Canada T6G 2G1
`bowman (at) math dot ualberta dot ca`
`http://www.math.ualberta.ca/~bowman/`

Drawing structured diagrams with SDDL

Mathieu Bourgeois and Roger Villemaire

Abstract

We present SDDL, a Structured Diagram Description Language aimed at producing graphical representations for discrete mathematics and computer science. SDDL allows combining graphical objects (circles, lines, arrows, ...) and LaTeX boxes to produce diagrams representing discrete structures such as graphs, trees, etc. with an easy-to-use domain specific language.

1 What is SDDL?

SDDL, *Structured Diagram Description Language*, is a high-level domain specific language tailored for diagrams that have an inherent structure. Examples of these might be trees, graphs and automata. Any diagram that is naturally structured can be described in SDDL. However, it was designed especially for data structures appearing in computer science and discrete mathematics.

The main objective of this language is to realize drawings in a natural and structured way. Mainly, one describes a drawing in SDDL in a way that is similar to drawing on a blackboard. For example, specific shapes (like circles, ellipses, texts, boxes) are drawn at positions that can be either absolute or relative to specific points on already placed shapes. Since we aim at mathematical drawings, text is handled through LaTeX.

Since exhibiting the structure of a diagram is essential to our purpose, SDDL uses an object-oriented hierarchy, completely written in Java, under our high-level language. Having shapes as objects makes structuring of diagrams easier and more intuitive since most shapes are explicitly represented by a class. Furthermore, extension by the end user is quite feasible since Java is a well-known language.

At the lower level, SDDL uses another graphical description language for LaTeX, namely Asymptote [1]. Our tool produces Asymptote code, which is finally converted to an encapsulated PostScript (eps) vector file.

2 Canvas and shapes

A diagram in SDDL is defined by a main `Canvas`. It is just like a standard painting canvas, in the sense that we can place (or paint) different things on it as much as we like. The things we can place inside a `Canvas` are `Shape`s. Thus, for creating a "Hello, world!" diagram, we would use the following:

```
put Text with [text = "Hello, world!"] in main;
```

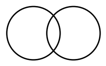
resulting in:

Hello, world!

In this example, `Text` is a `Shape`, and it possesses a property `text`, which is the LaTeX string used to render the text. Each `Shape` defines a certain number of such properties.

SDDL permits the definition of variables. It is a dynamically typed language, so variables don't have to be declared before being used. Variables make it possible to reuse the same `Shape` at multiple locations. For example,

```
a = Circle with [radius = 10.0];
put a at (-7.5, 0.0) in main;
put a at (7.5, 0.0) in main;
```

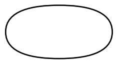


In this example, we specify the location where we want our `Shape` to be. If a location is not specified with the `at` clause, the `Shape` will be placed at the point of origin of the current `Canvas`, which is its center.

One important thing to note is that once a `Shape` has been put in a `Canvas`, it is immutable. It cannot be modified or removed. A `Shape` can be modified *after* it has been drawn on a `Canvas`, but this will have an effect only on later use of this object and will not modify the actual `Canvas` in any way; a `Shape` always appears in a `Canvas` as it was at the moment it was added.

When a `Shape` is created, every property specified by the user is set, in the given order. It is not necessary to specify all properties at once, nor to set all of them. Most of the properties have appropriate default values. However, some properties, if not set, will yield strange results. For example, a circle with no radius property set will have a default radius of 0. Usually, property order is irrelevant. For instance, giving for an ellipse the radius along the x-axis or the y-axis first will yield exactly the same result.

```
a = Ellipse;
a = a with [xradius = 20.0];
a = a with [yradius = 10.0];
put a in main;
```



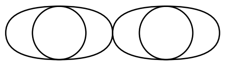
3 Structuring multiple canvases

Every diagram consists of a *main* `Canvas`. However, we can define other `Canvas` objects if we wish. An additional `Canvas` can be introduced using a typical variable assignment. However, when a `put` command is used, SDDL checks to see if the canvas variable is already defined. If it isn't, it automatically creates an empty `Canvas` for use.

One of the main reasons to use other `Canvas`es is to create an explicit structure in our diagram. Since a `Canvas` is a `Shape`, once a sub-`Canvas` has been created, it can be placed inside the `main` `Canvas`, or any other one for that matter. This will ensure that everything that is defined inside our sub-`Canvas` will be placed at the proper position in the final diagram. However, any `Canvas` that has been defined, but is not linked directly or indirectly with the *main* `Canvas`, will not be drawn in the final diagram.

As an example, let's say we want to make a diagram that consists of two identical "eyes". Instead of defining two identical objects, we will create one in a `Canvas` named *form* and put it at different positions inside our *main* `Canvas`. Our "eye" consists of a circle and an ellipse, both with the same center. All we need to do for this is place them at the default position of the `Canvas`. Finally, we can take this sub-`Canvas` and place it at the two positions required.

```
a = Circle with [radius=10.0];
b = Ellipse with [xradius=20.0, yradius=10.0];
put a in form;
put b in form;
put form at (-20.0, 0.0) in main;
put form at (20.0, 0.0) in main;
```



4 Paths

SDDL offers support for the description of paths. The way in which they are described is similar in syntax with MetaPost and Asymptote, though with some variations. A `Path` shape is described by linking points together with specific linking symbols. To draw a line between two points, the line symbol `--` can be used. To link some points using a curve, use the curved line symbol `~`, which will use Asymptote's positioning algorithms to create a nice-looking curve which passes through those points. For the moment, user control over the curve is limited to beginning and ending tangents, but could be expanded.

```
p1 = a--b--c;
p2 = a~b~c;
```

One of the other things you may want to do with a `Path` is to create an `Arrow` out of it. SDDL defines symbols for forward, backward and bidirectional arrows for both linear and curved lines. Right now, however, support is restricted to forward arrows.

```
p1 = a<-b--c->d;
p2 = a<~b~c~>d;
p3 = a<->b;
p4 = a<~>b;
```

Once a `Path` is defined, it can be used as any other `Shape`. However, one of the main things that you want to do is to put your `Path` somewhere. For this, you can write something like this:

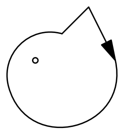
```
p = (0.0, 0.0) -- (10.0, 10.0);
put p in main;
```

This is the standard way to add `Shape`s to a `Canvas`. However, since `Path`s are usually used to link different `Shape`s together in the *main* `Canvas`, special handling is done in SDDL. Namely, a `Path` that is not explicitly assigned to a variable will be automatically placed in the *main* `Canvas`. Therefore the `Path` will be directly placed in *main* without any explicit `put` expression. Thus, linking has its own special syntax, which is more natural and convenient. As an example, this program

```
(0.0, 0.0) -- (10.0, 10.0);
```

yields exactly the same result as the one just above. Furthermore, points can be added and multiplied by a scalar (as vectors). Using a dot to locate the origin of the drawing, we can use the SDDL syntax to create this example:

```
a = (10.0, 10.0);
a--2.0*a->(30.0, 0.0);
a~(-1.0)*a~(30.0, 0.0);
```



Finally, as we will now see, `Shape`s also usually define specific points.

5 Drawing and linking shapes together

To draw a diagram, usually some shapes with a specific structure are first laid down. Once that is done, those objects are linked together with lines, curves and arrows. However, these links must be made between specific positions, usually derived from one or more specific `Shape`s. For example, we may want to link the northwestern point of a rectangle with the point on a circle at an angle of 45 degrees. These points could obviously be computed in advance. However, this becomes quite problematic when points are at peculiar angles or more complex relationships between points and `Shape`s are needed. Worst of all, if the position or any other property of a `Shape` is changed, all computations will have to be redone.

To ease object linking, SDDL provides so-called *reference points*. A reference point is a point that has no static value, unlike points defined by a pair of two real numbers. Instead, a reference point is defined by its relationship to a specific `Shape` appearing at a particular location. As a matter of fact, all reference points are defined along with the Shape because they are a natural part of it. This ensures that we always have nice-looking lines at exactly the positions we want them to be.

However, a reference point's exact position in a Canvas will depend on the Shape's position. Worse, since the same Shape can appears at multiple locations in the same Canvas, we have to know which occurrence we are talking about!

Therefore, SDDL introduces a feature called a Drawing. A Drawing is an object that represents a Shape at a particular position, i.e. a specific occurrence of a Shape in a Canvas. Whenever a Shape is put inside a Canvas, a Drawing is returned and can be assigned to a variable. This gives a way to uniquely identify every occurrence of a Shape appearing in a Canvas. If the same Shape is placed twice, two different Drawings will be returned, each referring to a different occurrence of the same Shape.

```
a = Circle with [radius = 10.0];
d1 = put a at (-10.0, 0.0) in main;
d2 = put a at (10.0, 0.0) in main;
```

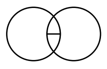
Once a drawing has been defined, a way to access its points is needed. SDDL defines a syntax for extracting points from drawings:

coord of ⟨*coord*⟩ **(** ⟨*args*⟩ **) in** ⟨*drawing*⟩

Here one specifies the kind of point to use (⟨*coord*⟩) and any particular arguments required to obtain it. The available points are Shape specific, so only those kinds of points which make sense for the particular Shape can be used. For instance one can get a point at a particular angle on a Circle. This is also an example where an additional argument is needed. For instance, anglePoint(45.0) will give the point at 45 degrees.

Finally, to reference the drawing from which we take the point, its drawing Path (a dot separated path) from the main diagram must be given.

```
a = Circle with [radius = 10.0];
d1 = put a at (-7.5, 0.0) in main;
d2 = put a at (7.5, 0.0) in main;
(coord of anglePoint(0.0) in main.d1) --
   (coord of anglePoint(180.0) in main.d2);
```

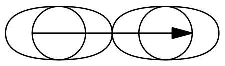


In this example, a line between two identical circles is drawn. The drawing Paths are simple here, since both drawings are made directly in *main*. Thus, only *main* followed by the drawing name is needed.

Simply giving a Drawing is not sufficient in order to make reference points non-ambiguous; complete drawing Paths are required, as we will now show. Let's go back to the two eyes we drew previously. The eye was defined by a circle and an ellipse, both put inside a sub-Canvas. Now, let's say we want to draw a line between the two circles. Each of those circles is defined in the sub-Canvas and they are, as a matter of fact, the same Drawing *a*. Adding a link between *a* and itself would add a link between the sub-Canvas' circle and itself (what could this mean?). Adding the sub-Canvas to the main Canvas twice, as we did before, would just duplicate this structure inside the main Canvas.

But since a Canvas is a Shape, we get a drawing when we put the sub-Canvas inside *main*. We can therefore identify each inner circle by listing the drawings required to access them, as shown in the following SDDL example. Thus, we can always link together Shapes that are deeply nested inside a sub-Canvas.

```
a = Circle with [radius=10.0];
b = Ellipse with [xradius=20.0, yradius=10.0];
c = put a in form;
put b in form;
f1 = put form at (-20.0, 0.0) in main;
f2 = put form at (20.0, 0.0) in main;
(coord of anglePoint(180.0) in main.f1.c) ->
  (coord of anglePoint(0.0) in main.f2.c);
```



6 Programming constructs and lists

SDDL also defines typical programming constructs. For instance lists are created and elements accessed in a syntax similar to most other dynamic languages. Since the content of a list is a general Java Object, you can create a list of objects that are not of the same type. Thus, an assignment like

```
l = [2.0, (0.0, 0.0), [], a--b];
```

is perfectly legal, even if not typically very useful! A more typical use of lists would be

```
list = [(0.0, 0.0), (10.0, 10.0)];
l = list[0];
l2 = list[1];
l--l2;
```

Lists are nice, but to use them properly, adequate programming constructs are needed, such as for and while loops. SDDL defines those, permitting us to draw arrays of Shapes or define points through a simple list.

The ability to use loops significantly simplifies many diagrams in computer science. In this example, we first define a list of points. Afterward, using a for loop, we link those points into a linear Path.

```
a = [(10.0, 10.0), (20.0, 10.0),
     (20.0, 20.0), (10.0, 20.0)];
for i from 0 to 2 do
```

```
  a[i]--a[i+1];
end
```

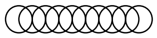
Alternatively, we could get the same result using the following while loop.

```
i = 0;
while i != 3 do
  a[i]--a[i+1];
  i = i + 1;
end
```

7 Functions

SDDL also permits the creation of functions inside the code. Those must be written before the diagram description. A SDDL function is defined with the keyword `fun`. The list of arguments does not need to be typed, only named. The return type (if it exists) does not need to be specified either. To call the function in the resulting code, a typical C-like call is used.

```
fun dropMany(s, i, v, n, c){
  for j from 1 to n do
    put s at i + j*v in c;
  end
}
circle = Circle with [radius = 5.0];
dropMany(circle, (0.0, 0.0),
         (5.0, 0.0), 10.0, main);
```



In this case, we define a function that drops many instances of a `Shape` *s* at different positions on a `Canvas` *c*. The positions of the `Shape` are determined by an initial point *i* and a translation vector *v*. Finally, the number of `Shape`s placed is also given as a parameter *n*. Each `Shape` is then placed at the initial point translated by the translation vector a certain number of times. This permits us to create many circles along one line.

One important point is the parameter for the `Canvas` in which we place the `Shape`s. However, we pass *main* as our parameter. The reason for this is that SDDL does not possess global variables. A function can only access its parameters and variables that have been defined inside the function. Thus, trying to place something in *main* from inside a function will yield an error message saying the variable *main* is unknown.

8 Primitives

SDDL, as mentioned before, is written in Java. One of the nice features of Java is its libraries. Many classes have been written for any number of different tasks and the number of functions and algorithms implemented is astonishing. To rewrite libraries that are already defined in Java or to link them one by one in SDDL seemed pointless. It is much nicer to directly use those functions, since they're already there.

Since Java is a reflexive language, in the sense that the program knows about itself and can modify itself at will, functions can be dynamically accessed at runtime. We already use this feature to simplify the definition of the SDDL interpreter: the interpreter doesn't have to know every possible type of `Shape` in order to work. Using these same mechanisms permits loading classes at runtime and gives the user the power to call from inside SDDL any Java function.

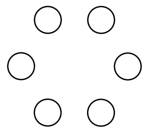
This is done through so-called primitives, which are defined using the keyword `primitive` followed by the name of the function, as it will appear in SDDL. Following that, two strings are given: one for the package in which the function is defined and one for the name of the function as it was defined in Java. Any primitive declaration must be made prior to any function definition.

Let's look at an example. One of the main things a diagram description language would require is some basic mathematics and geometry libraries. Java possesses a nice `java.lang.math` package which we would like to use, especially the `sin` and `cos` functions. For this, we will need to define two primitives.

```
primitive static sin "java.lang.Math" "sin";
primitive static cos "java.lang.Math" "cos";
```

Once they are defined, they can be called just like any other SDDL functions. In this case, we will use those functions to place circles around an invisible circle.

```
for i from 1 to 6 do
  put Circle with [radius = 5.0] at
             (20.0 * cos(i * 3.14 / 3.0),
              20.0 * sin(i * 3.14 / 3.0))
             in main;
end
```



9 A concrete example

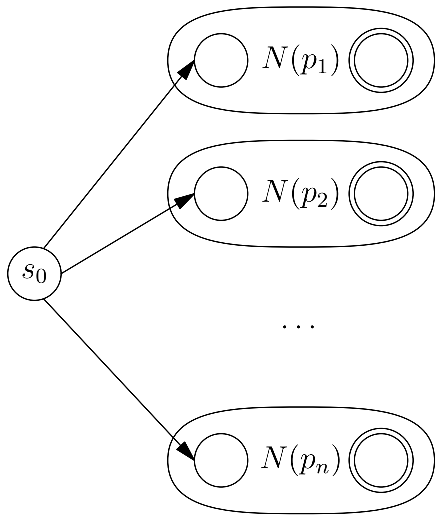
Now that we have all our tools, let's use SDDL to describe a simple diagram of an automaton. This is the complete code, along with the resulting image.

```
circle1 = Circle with [radius = 10.0];
circle2 = Circle with [radius = 12.0];
ellipse = Ellipse with [xradius = 50.0,
                        yradius = 20.0];
circle = put circle1 at (-30.0, 0.0) in form;
put circle1 at (30.0, 0.0) in form;
put circle2 at (30.0, 0.0) in form;
put ellipse in form;

d1 = put form at (100.0, 100.0) in main;
d2 = put form at (100.0, 50.0) in main;
d3 = put form at (100.0, -50.0) in main;
put Text with [text="$N(p_1)$"]
         at (100.0, 100.0) in main;
put Text with [text="$N(p_2)$"]
         at (100.0, 50.0) in main;
put Text with [text="\dots"]
         at (100.0,0.0) in main;
put Text with [text="$N(p_n)$"]
         at (100.0, -50.0) in main;

put Text with [text = "$s_0$"]
         at (0.0, 20.0) in main;
d4 = put circle1 at (0.0, 20.0) in main;

(coord of anglePoint(70.0) in main.d4)->
(coord of anglePoint(180.0) in main.d1.circle);
(coord of anglePoint(0.0) in main.d4)->
(coord of anglePoint(180.0) in main.d2.circle);
(coord of anglePoint(-70.0) in main.d4)->
(coord of anglePoint(180.0) in main.d3.circle);
```



10 Modifying the hierarchy by adding shapes

Using what has been described above, most diagrams can be created. However, many simplifications can be made and some domains may find it relevant to add classes specific to their trade. SDDL eases the modification of the Java hierarchy underneath the language by using reflexivity. Any user who respects some basic guidelines will be able to have its class work automatically in SDDL without any specific linking required.

To do this, every class must possess a certain number of properties. Those properties are defined through setters and getters. For instance, for the radius of a circle, these are called `setRadius` and `getRadius`. Any property that follows this rule will be accessible.

Furthermore, an empty constructor must also be defined that sets, as much as possible, default values for all properties of the defined `Shape`. There are also some small functions to be defined, like the `draw` and `obtainPath` functions. Any reference points must also be defined with a function to obtain them.

As an example, let's say we would like to add a class to represent a cloud. A cloud will have a number of "spikes" and a size. The class definition (without function definition) that we would require would be the following:

```
public class Cloud extends Shape{
  double size;
  int numberOfSpikes;
  public Cloud(){...}
  public void setSize(double size){...}
  public double getSize(){...}
  public void setNumberOfSpikes(int n){...}
  public int getNumberOfSpikes(){...}
  public void draw(AbstractPoint a,
                   PrintWriter w){...}
  public Path obtainPath(){...}
}
```

11 Future additions and availability

SDDL is available now at `http://www.info2.uqam.ca/~villemaire_r/Recherche/SDDL/`. It is functional, though not by any means complete. Many additional `Shape`s could be added (in particular tree and graph classes) and options could be added to existing classes. Development of the application continues for the time being and a more thorough version will be made available.

References

[1] John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. *TUGboat: The Communications of the TEX Users Group*, 29:288–294, 2008.

◇ Mathieu Bourgeois
Université du Québec à Montréal
Montréal, Canada
`bourgeois dot mathieu dot 2 (at) courrier dot uqam dot ca`

◇ Roger Villemaire
Université du Québec à Montréal
Montréal, Canada
`villemaire dot roger (at) uqam dot ca`
`http://intra.info.uqam.ca/personnels/Members/villemaire_r`

Unicode mathematics in LaTeX: Advantages and challenges

Will Robertson

Abstract

Over the last few years I've been tinkering with Unicode mathematics in XeTeX. In this paper, I discuss Unicode mathematics in the context of LaTeX with the unicode-math package.

1 Introduction

XeTeX was the first widely-used Unicode extension to TeX. Several years ago Jonathan Kew added OpenType maths support to XeTeX [12] following Microsoft's addition of mathematics to the OpenType specification as they were preparing Microsoft Word 2007. Around that time I built a prototype implementation of a Unicode maths layer for LaTeX, called unicode-math, but with very few OpenType maths fonts available, and other projects consuming my time, the project lost momentum and I never managed to finish the package and upload it to CTAN.

That has now changed. In the leadup to the TUG 2010 conference I thoroughly revisited the code, re-writing most of it in the LaTeX3 programming environment 'expl3'. (A brief introduction to expl3 is given by Joseph Wright in [24].) Long-standing issues were resolved and support for LuaTeX was begun. It is now ready for greater distribution with TeX Live 2010.

In a happy twist of fate, the STIX fonts have recently been released and can be used by this package. Details to follow.

1.1 Outline

In Sections 2 and 3, I cover the origins and nature of Unicode mathematics, and what fonts are currently available which use it. In Sections 4 and 5, I address specific details of how to use Unicode maths in LaTeX, and comment on some challenges faced when doing so; in Section 6, I present my thoughts for the possible future of this work. Finally, in Sections 7 and 8 I discuss some technical aspects of the package and its development process.

2 What is Unicode maths?

Before we talk about Unicode maths, it is necessary to discuss the computer typesetting of mathematics from the very beginning, or at least since TeX was first created.

2.1 Origins

TeX was designed alongside a set of text and maths fonts, the 'Computer Modern' family. The original Computer Modern maths fonts were limited by restrictions of the time, consisting of three separate fonts with 128 glyphs each (for each design size).

Later, the well-known amsmath package provided a complement of glyphs designed to match Computer Modern; these extra maths fonts extended the repertoire of standard symbols that could be expected to be used by most mathematicians.

As well as the amsmath fonts, a (small) number of other maths fonts were also created for TeX systems, including Lucida[1] and MathTime Pro.[2] Each maths font developed generally contained a different set of glyphs, and as a consequence of this the developers who had to write the TeX support layer for each font generally had to start from scratch to implement the font encoding that bound symbols to glyph slot numbers. This tedious process is one factor in explaining the general dearth of maths fonts for TeX-based systems.

2.2 The newmath encoding

In the 1990s, the Math Font Group[3] was created to design an 8-bit math font encoding [7] to alleviate this problem of having to invent *ad hoc* encodings for each new maths font. This 'newmath' encoding was carefully designed to include as many maths symbols as possible, and each symbol was assigned a standard glyph slot. New fonts could just follow this system, and switching maths fonts would be as easy as switching text fonts since the newmath font encoding would automatically know where all the symbols were located.

The project produced a LaTeX implementation to support the 'newmath' encoding, but it was never completed for a variety of reasons. While XeLaTeX and LuaLaTeX are now available to access OpenType fonts that use Unicode maths, there may be still some interest in retaining (and finally releasing) newmath for future large-scale maths font encoding support — perhaps in order to support the STIX fonts and/or the proposed Latin Modern Math font in eight-bit LaTeX [13].

2.3 Unicode maths and the STIX fonts

After newmath the attention of the Math Font Group turned to Unicode, namely to answer the question: 'What maths symbols have actually been used and invented in published technical writing?' The particulars of this phase of history have been covered by Barbara Beeton's report of the project at the time [2]. To sum it up very briefly, members of this project,

[1] `http://tug.org/lucida`
[2] `http://www.pctex.com/mtpro2.html`
[3] `http://www.tug.org/twg/mfg/`

$$\cos^2 \phi + \sin^2 \phi = -e^{i\pi}$$

```
\setmathfont{Cambria Math}
$\cos^2 \varphi + \sin^2 \varphi = -e^{i\pi}$
```

Example 1: A minimal example of the unicode-math package.

now known as the STIX Project, gathered together a comprehensive list of symbols used in mathematics from as many sources as they could find and submitted these symbols to the Unicode consortium for addition to the Unicode specification. From their labours, we now have a formal description of thousands of glyphs that a maths font should contain and the particulars of how those glyphs should look and behave [4].

Having defined the symbols to appear in Unicode mathematics, a group of scientific publishers commissioned a new font family to be the reference implementation for the newly specified Unicode mathematics [3]. These STIX fonts were designed to blend with Times New Roman which was, I believe, (and perhaps still is) the most commonly used font in technical publishing.

2.4 OpenType maths and the modern era

But mathematics typesetting needs more than just glyphs. TEX itself uses a number of parameters built into the maths fonts it uses in order to place mathematics on the page in a form suitable for high-quality typesetting, such as where superscripts should be placed, whether delimiters should grow to encompass the material they surround, what alternative glyph to use for 'big operators' when in displaystyle rather than textstyle, and so on. The details have been elucidated and illustrated splendidly by Bogusław Jackowski [11]. A system to utilise Unicode maths must contain analogous information and use similar algorithms to produce acceptable results.

For this purpose, Microsoft extended the OpenType specification to include tables of structured information for mathematics typesetting, generalising and extending the original algorithms within TEX.

OpenType maths has been described in more detail by Ulrik Vieth both in the context of its historical development [21] and with a particular emphasis on how the OpenType parameters correspond to TEX's own [23]. He has also discussed some of the deficiencies of TEX's mathematics engine [20], most of which are now addressed with OpenType maths.

3 The unicode-math package

With Unicode mathematics able to encode the maths glyphs we need, and the OpenType font format able to store the required parameters to use the new maths fonts, the only thing missing is the typesetting engine to put the pieces together. Microsoft Word 2007 and 2010 contains one, and so does XeTEX and LuaTEX. It is important to recognise that a Unicode maths font is suitable for both Word and TEX-based systems, which I believe will aid the adoption of the Unicode maths approach.

The unicode-math package is an initial attempt to write a high-level interface to Unicode maths for LATEX documents. After loading the package, users can write

```
\setmathfont{Cambria Math}
```

as shown in Example 1 to select Cambria Math or any other Unicode maths font.

Readers may be familiar with the fontspec package, which is a high-level interface for loading fonts (usually OpenType fonts) in XeLATEX and now also LuaLATEX [18]. Where fontspec is designed for loading fonts to change the text font of the document, unicode-math allows a similar interface to select the maths font.

Previous work in this area has been performed by Andrew Moschou with his mathspec package for XeLATEX. With mathspec, a text font can be loaded to substitute the alphabetic symbols of the mathematics setup — say to use Minion Pro Italic for the Latin symbols and Porson for the Greek symbols — but all other maths symbols are left untouched. A similar process has been shown previously for maths fonts in eight-bit LATEX by Thierry Bouche [5]. The unicode-math package, by contrast, is designed to use OpenType maths fonts that contain all glyphs and associated information necessary to replace the existing LATEX maths setup.

The two packages are therefore designed for different purposes; use mathspec if most of your maths needs are fulfilled by a pre-existing maths package (such as mathpazo) but you would like your maths alphabets to be taken from the text font; alternatively, use unicode-math if you have an OpenType maths font that you would like to use for typesetting all aspects of the mathematics.

The unicode-math package almost completely replaces LATEX's maths setup. Control sequences are provided to access every Unicode maths symbol, and literal input of all such characters in the source is also supported. Maths can be copied from another source (such as a web page or PDF document) and pasted directly into the LATEX document and the

Cambria: $$\int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}t$$

Asana: $$\int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}t$$

STIX: $$\int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}t$$

Euler: $$\int_0^\infty \mathrm{e}^{-\mathrm{st}} \mathrm{f}(\mathrm{t})\,\mathrm{dt}$$

```
\def\laplace{\hfill$\displaystyle
  \int_0^\infty \mathup e^{-st}f(t)\,\mathup dt
                $\\[1ex]}
Cambria: \setmathfont{Cambria Math} \laplace
Asana:   \setmathfont{Asana Math}   \laplace
STIX:    \setmathfont{XITS Math}    \laplace
Euler:   \setmathfont[math-style=upright]
                    {Neo Euler}    \laplace
```

Example 2: Available OpenType maths fonts at the time of writing.

content will be retained, albeit with some loss of its presentational aspects (most notably subscripts and superscripts).

With some minor exceptions, no changes to the mathematical document source should be necessary to be able to switch fonts using Unicode maths. ConTEXt has an analogous system [14], and we have discussed future plans for coordinating our efforts to be consistent where possible and reduce duplication of work between ConTEXt and LaTeX.

3.1 What fonts are available?

This is all well and good, but the system doesn't do much good if there are no fonts to take advantage of it. *Cambria Math*, by Tiro Typeworks,[4] was the first OpenType maths font released (through Ascender Corp.), commissioned originally for Microsoft Office 2007.

There are three open source OpenType maths fonts currently available, developed using the free font editor FontForge[5] to add the OpenType maths parameters. These fonts are:

- Apostolos Syropoulos's *Asana Math*,[6] which has its origins in the 'Pazo' fonts, which are a clone of Palatino with additional maths support;
- Khaled Hosny's *XITS Math*,[7] which is a fork of the STIX fonts to include preliminary OpenType maths layout information (XITS will eventually be deprecated by an official release of the STIX fonts with the same functionality); and,
- Khaled Hosny's *Neo Euler*,[8] which is a Unicode re-working [10] of Hermann Zapf and Donald Knuth's Euler font.

[4] `http://www.tiro.com/projects.html`
[5] `http://fontforge.sourceforge.net/`
[6] `http://ctan.org/pkg/asana-math`

These four OpenType maths fonts are shown in Example 2, in which note the fact that the maths font can now change part-way through a document.

Of these, XITS Math and Asana Math will both be included in TEX Live 2010, and they can be loaded with (respectively)

```
\setmathfont{xits-math.otf}
\setmathfont{Asana-Math.otf}
```

without any font installation necessary.

Readers may be interested in Daniel Rhatigan's dissertation [17] on the history of and design comparisons between the Times-, Euler-, and Cambria-based maths fonts (recall that STIX is modelled after Times).

4 Advantages

The main advantage of using Unicode maths is that it becomes easy to switch between maths fonts. There are some more benefits than simply standardising the way maths fonts are loaded, however.

I suspect the most directly useful aspect of Unicode maths will be relieving (most of) the headache around finding *and using* a particular math font glyph. The STIX fonts are available as a fallback font for all symbols that are part of Unicode maths. After all, most maths symbols are geometrically abstract enough that they do not need to be directly matched with the text font.

4.1 Readable source

Unicode maths provides the ability for maths symbols and characters to be input in Unicode directly in the source file, as shown in Example 3. For example, you may input a literal 'α' directly into a source document rather than typing '`\alpha`'. A convenient way to achieve this input style is to use the auto-completion of text editors such as TeXShop and TEXworks, in which typing a unicode-math control sequence and then hitting the 'escape' key will produce the literal input character. Since the original control sequence still must be typed letter-by-letter, this technique doesn't improve input speed, but makes source documents far more readable and amenable

[7] `http://github.com/khaledhosny/xits-math`
[8] `http://github.com/khaledhosny/euler-otf`

```
\[ 𝐄 = − ∇ 𝜙 − \frac{∂𝐀}{∂t} \]
\[ 𝐁 = ∇ × 𝐀 \]
\[ ∇ ⋅ 𝐃 = 𝜌 \]
\[ ∇ × 𝐇 − \frac{∂𝐃}{∂t} = 𝐉 \]
```

Example 3: Example of LaTeX source using Unicode math input with literal maths characters. Such input can be pasted from another source or typed with the aid of 'smart completion' in a text editor.

to casual editing. (Completion files for unicode-math will be distributed with the package.)

With direct Unicode input for symbols in a LaTeX document, only small changes to the regular syntax are required to approach the simplicity of Murray Sargent's 'nearly plain-text encoding of mathematics' [19], which can be used in Microsoft Office to achieve a TeX-like efficiency at writing maths while obtaining a WYSIWYG view of the document. (I personally still prefer the TeX way, however, since you can use macros and so on to retain consistency and give your symbols meaning.)

4.2 Mathematical alphabets

Unicode maths contains glyph slots to contain all styled alphabetic symbols used in mathematics, including bold, blackboard, script, etc., styles. The complete listing is shown in Example 4. Each style contains variations on some or all of the lowercase and uppercase Latin and Greek characters and Arabic numerals. The commands shown for switching alphabets force each particular shape, hence their explicit names such as `bfit` for 'bold italic'; general `\mathbf` and `\mathsf` commands are also provided to switch to the correct upright or italic shape depending on the context (see Section 4.3 and Example 6). Note that `\mathbf` is used to access bold symbols in both Latin *and* Greek; this is a great useability improvement over traditional LaTeX that requires either `\boldsymbol` or the bm package (or a specific maths font package) to access bold Greek letters.

As an aside, note that the command `\mathrm` from LaTeX is renamed in unicode-math to `\mathup` to emphasise the fact that it can be used for upright *Greek* symbols as well. The old name is still provided for backwards compatibility, of course.

As authors wish to use fonts with alphabet styles that are not currently present in Unicode, the system must be able to cope with the addition of new alphabets and new alphabet styles. The most relevant example here is the existence in the STIX fonts of a variety of these non-Unicode ranges, most notably the 'calligraphic' style in contradistinction to the 'script' style that *is* included in Unicode. Example 5 shows the differences between these two styles; some mathematicians are used to using these two alphabet styles separately (with script letters accessed through the mathrsfs package, for example). Here, the XITS fonts have encoded the calligraphic shapes in the position of the script glyphs under the OpenType font feature `ss01`, which is accessed through fontspec font features as `StylisticSet=1`.

In time, I believe that the calligraphic alphabet will be incorporated into the Unicode standard, but until then the unicode-math package must be able to use it explicitly as an exceptional case. The system in unicode-math for creating new alphabet styles in this way is not completely generalised yet, but work in this area is planned for the future (including the addition of alphabets neither Latin nor Greek that might be also used in a mathematical context, such as Russian).

`\mathit`	𝑎𝑏𝑐	𝑋𝑌𝑍	𝛼𝜉𝜃	𝛹𝛯𝛺	
`\mathbfit`	𝒂𝒃𝒄	𝑿𝒀𝒁	𝜶𝝃𝜽	𝜳𝜩𝜴	
`\mathup`	abc	XYZ	αξθ	ΨΞΩ	123
`\mathbfup`	𝐚𝐛𝐜	𝐗𝐘𝐙	𝛂𝛏𝛉	𝚿𝚵𝛀	𝟏𝟐𝟑
`\mathbb`	𝕒𝕓𝕔	𝕏𝕐ℤ			𝟙𝟚𝟛
`\mathtt`	𝚊𝚋𝚌	𝚇𝚈𝚉			𝟷𝟸𝟹
`\mathsfit`	𝘢𝘣𝘤	𝘟𝘠𝘡			
`\mathbfsfit`	𝙖𝙗𝙘	𝙓𝙔𝙕	𝞪𝞷𝞱	𝞧𝞝𝞨	
`\mathsf`	𝖺𝖻𝖼	𝖷𝖸𝖹			𝟣𝟤𝟥
`\mathbfsfup`	𝗮𝗯𝗰	𝗫𝗬𝗭	𝝰𝝽𝝷	𝝭𝝣𝝮	𝟭𝟮𝟯
`\mathscr`	𝒶𝒷𝒸	𝒳𝒴𝒵			
`\mathbfscr`	𝓪𝓫𝓬	𝓧𝓨𝓩			
`\mathfrak`	𝔞𝔟𝔠	𝔛𝔜ℨ			
`\mathbffrak`	𝖆𝖇𝖈	𝖃𝖄𝖅			

Example 4: Mathematical alphabets in Unicode from the STIX fonts.

Script style: 𝒜ℬ𝒞𝒳𝒴𝒵
Calligraphic: 𝒜ℬ𝒞𝒳𝒴𝒵

```
\setmathfont
  [range=\mathscr]{XITS Math}
\setmathfont
  [range=\mathcal,StylisticSet=1]{XITS Math}
Script style: $\mathscr{ABCXYZ}$\\
Calligraphic: $\mathcal{ABCXYZ}$
```

Example 5: Accessing the non-Unicode calligraphic style in the STIX fonts.

```
math-style=
```

`ISO`	a	z	B	X	α	β	Γ	Ξ
	$\boldsymbol{a}$	$\boldsymbol{z}$	$\boldsymbol{B}$	$\boldsymbol{X}$	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\Gamma}$	$\boldsymbol{\Xi}$
`TeX`	a	z	B	X	α	β	$\mathrm{\Gamma}$	$\mathrm{\Xi}$
	$\mathbf{a}$	$\mathbf{z}$	$\mathbf{B}$	$\mathbf{X}$	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\mathbf{\Gamma}$	$\mathbf{\Xi}$
`upright`	a	z	B	X	$\upalpha$	$\upbeta$	$\mathrm{\Gamma}$	$\mathrm{\Xi}$
	$\mathbf{a}$	$\mathbf{z}$	$\mathbf{B}$	$\mathbf{X}$	$\boldsymbol{\upalpha}$	$\boldsymbol{\upbeta}$	$\mathbf{\Gamma}$	$\mathbf{\Xi}$
`french`	a	z	B	X	$\upalpha$	$\upbeta$	$\mathrm{\Gamma}$	$\mathrm{\Xi}$
	$\mathbf{a}$	$\mathbf{z}$	$\mathbf{B}$	$\mathbf{X}$	$\boldsymbol{\upalpha}$	$\boldsymbol{\upbeta}$	$\mathbf{\Gamma}$	$\mathbf{\Xi}$

Example 6: Different output styles without changing the input source according to the `math-style` option.

4.3 Flexible output

The unicode-math package does not assume a one-to-one mapping between the Unicode characters in the source and the Unicode glyphs in the output. In fact, the design of the maths setup, by default, is such that there is no semantic difference between upright and italic letters in the input source; consistent output is achieved regardless of the style of the input source.

Claudio Beccari [1] has detailed the requirements of typesetting mathematics according to the ISO standard (ISO31/XI), which requirements differ in important ways from the typical output of LaTeX mathematics. More recently, Ulrik Vieth [22] discussed many of the details of mathematical typesetting in the context of mathematical physics; the features offered by the unicode-math package help to provide the flexibility required to achieve these ideas for any maths font available. (Packages to perform this in classical LaTeX, such as the isomath package, require maths fonts set up with a particular encoding.) As an example of the different approaches to mathematical typesetting, Example 6 shows how documents are able to be typeset per ISO standards or in a more classical TeX-like format without changing the source text of the mathematics. Similarly, the output style of bold characters can also be adjusted.

As the package can load fonts for maths glyphs dynamically, multiple fonts and multiple styles can be used between various characters or families or alphabets of characters. Example 7 shows an example in which the maths was typed 'as usual', but different glyphs and glyph ranges were assigned fonts with different colours (grayscaled for *TUGboat*). This particular example may not be very practical, but it illustrates that the system is flexible enough to accommodate a wide range of effects. Even single characters within an alphabet may be chosen, such as in Example 8 where the 'π' symbol alone is chosen to be typeset upright.

$$F(s)=\mathscr{L}\biggl\{f(t)\biggr\}=\int_0^\infty \mathrm{e}^{-st}f(t)\,\mathrm{d}t$$

```
\setmathfont{Cambria Math}
\def\SET#1{\setmathfont[#1]{Cambria Math}}
\SET{range={\mathop,\mathscr}, Colour=red}
\SET{range={\equal}, Colour=00BB22}
\SET{range={\mathopen,\mathclose}, Colour=blue}

\[ F(s)=\mathscr{L}\,\biggl\{f(t)\biggr\}
   = \int_0^\infty \mathup e^{-st}f(t)
   \, \mathup d t \]
```

Example 7: Hooks make it possible to use a variety of fonts or styles — in this case, colours — for different maths characters or families/alphabets of maths characters.

1: $\{\alpha,\dots,\pi,\dots,\omega\}$

2: $\{\alpha,\dots,\mathrm{\pi},\dots,\omega\}$

```
\setmathfont{Cambria Math}
1: $\{\alpha,\dots,\pi,\dots,\omega\}$

\setmathfont
  [range={"1D70B},math-style=upright]
  {Cambria Math}
2: $\{\alpha,\dots,\pi,\dots,\omega\}$
```

Example 8: An example of selecting a different font for a single alphabetic glyph. The glyph slot `"1D70B` corresponds to the pi symbol in the mathematical Greek Unicode range.

5 Challenges

The biggest problem I can see with the advent of Unicode maths, besides more fonts — I believe they'll slowly start to appear now that there are tools and programs to support them — is educating people into using them well.

5.1 Using the correct characters

Example 9 shows five different maths glyphs that are all triangular, while Example 10 shows the eight different slash-like glyphs; four in each direction. Consider whether it's clear, only from the description in the tables, which ones to use in different contexts.

Without careful documentation and good education, it may be hard for users to know which is the 'correct' glyph to use in many occasions. The markup in TeX and LaTeX has generally steered towards presentational aspects. But, as an example, with five different choices for which triangle to choose,

Slot	Command	Glyph	Class
U+25B5	`\vartriangle`	$x \vartriangle y$	relation
U+25B3	`\bigtriangleup`	$x \bigtriangleup y$	binary
U+25B3	`\triangle`	$x\triangle y$	ordinary
U+2206	`\increment`	$x\increment y$	ordinary
U+0394	`\mathup\Delta`	$x\mathup{\Delta}y$	ordinary

Example 9: Four triangular glyphs (from the STIX fonts) with five different uses but all with similar shapes.

Slot	Name	Glyph	Command
U+002F	Solidus	x/y	`\slash`
U+2044	Fraction slash	$x\fracslash y$	`\fracslash`
U+2215	Division slash	$x \divslash y$	`\divslash`
U+29F8	Big solidus	$x \xsol y$	`\xsol`
U+005C	Reverse solidus	$x\backslash y$	`\backslash`
U+2216	Set minus	$x \smallsetminus y$	`\smallsetminus`
U+29F5	Reverse solidus operator	$x \setminus y$	`\setminus`
U+29F9	Big reverse solidus	$x \xbsol y$	`\xbsol`

Example 10: A multitude of symbols for different purposes. Glyphs taken from the STIX fonts.

different authors may inadvertently choose different (but visually similar) glyphs for the same purpose in their mathematics. Furthermore, font designers are going to need to carefully design these glyphs to be consistent with the STIX fonts, which have been designed as 'reference material' against which all aspects of Unicode maths can be compared.

My feelings are that new tools will be needed to write LaTeX mathematics more semantically (which I will talk about later in Section 6.2). But such tools will need to be specific for each scientific field that uses different notation. This is an open problem.

5.2 LaTeX vs MathML

Mathematics represented in TeX and MathML are really quite separate beasts, although TeX can (perhaps obviously) be used as an engine to typeset MathML [9, 16]. While LaTeX input is designed to be hand-written and has visual output as the primary goal, MathML is a machine-friendly (human-unfriendly!) language to represent mathematics far more unambiguously and verbosely. There is not much overlap between how LaTeX looks at Unicode maths and how MathML is used, although packages such as stex ('semantic TeX') wed the ideas of 'Content MathML' to LaTeX (I briefly discuss semantic input of maths later in Section 6.2).

MathML and LaTeX often use different names for the symbols in Unicode maths. For example, the infinity symbol ∞ (U+221E) is `\infty` in TeX and `&infin;` in MathML. There are very few naming conflicts, but do bear in mind that the W3C names for maths symbols can occasionally be incompatible with the names used in unicode-math. As an example, consider the two 'set minus' characters in Example 10, which inherit their names from Plain TeX and the amssymb package, respectively. U+2216 is `\smallsetminus` and U+29F5 is `\setminus`. However, MathML does it differently due to a historical accident: U+2216 is referred to by either `&setminus;` or `&smallsetminus;` or a number of other synonyms; U+29F5 is as-yet unnamed [6]. The general mismatch between these two Unicode maths glyph naming schemes might make it difficult to move between MathML and LaTeX if one is used to writing symbol names in MathML and starts writing LaTeX mathematics, or vice versa.

Despite the semantic advantages of Content MathML, however, it is still not supposed to be used as an input language for mathematics; MathML and the language of LaTeX maths are simply designed for different things. Therefore, in practise I don't believe there will be any problems resulting from the differences in glyph naming between the two.

6 Thoughts for the future

Unicode is clearly here to stay, and we are entering a time where, for the first time, fonts for mathematics can be built with standard OpenType font tools, and they can be used in a variety of cross-platform environments — from XeTeX and LuaTeX to Microsoft Office to MathML on the web. I hope and believe that this will herald the more profuse production of maths fonts than we've seen in the past.

The unicode-math package is only the first step for modernising the maths support in LaTeX. I consider the future of maths in LaTeX to be supported by three main pillars of functionality: font support; structural improvements to the input language supported by advanced layout algorithms; and 'semantic'-style input. Font support is broadly covered by the unicode-math package, which leaves two topics to discuss below.

6.1 Layout of mathematics

For 'structural improvements to the input language', I really mean improvements for writing the kinds of things that the amsmath package has typically been used for; namely, it provides high(er)-level tools to describe the layout of mathematical expressions. While the amsmath package has been extremely popular for many years, it is not perfect. The best candidate to extend it is the breqn package [8], which is now maintained by Morten Høgholm. (breqn is completely compatible with amsmath, thus transitioning from one to the other is very easy.)

The breqn package's primary features are to simplify the input necessary over what is required for more complex structures in amsmath; the way that it does this is by incorporating complex algorithms to perform automatic breaking of mathematics over lines. This has long been regarded as impossible to perform correctly all of the time — and while no-one is arguing that breqn is always correct, it *usually* is. When it is not, the task is done manually as is presently the case anyway.

6.2 Semantic input of mathematics

If you look over the list of 'TeX names' used by unicode-math for the Unicode maths symbols, it is clear that the names chosen have often been chosen to be descriptive rather than semantic. For example, `\doteq`, `\bigwedge`, `\smwhtsquare` ('small white square'), and so on. This is not unique to unicode-math; this follows the general naming scheme for LaTeX math font symbols where the name of a symbol shouldn't be too specific for one general use.

However, when there are clear semantics for symbols it is generally more useful to use a semantic input style for that piece of mathematics. For example, with '$a \to b$' (and this is from regular LaTeX), it is clearly more sensible to write `$a \to b$` rather than `$a \rightarrow b$` when we're writing what would be said aloud as 'for/from a to b'. Similarly, (and more hypothetically), writing `\intersection` and `\union` is probably better than `\cap` and `\cup`, respectively, in that their meaning in the former is immediately obvious from the source document.

I am aware of two macro packages that attempt to provide a general semantic input style for mathematics in LaTeX: the cool ('content-oriented LaTeX') package and the aforementioned stex package. As an argument for using them, and by way of comparison between them, consider writing an integral

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x.$$

In pure LaTeX, we must write this in a purely presentational manner, explicitly writing subscripts and superscripts on the integral symbol, and inserting a manual space and upright font switch to write the '$\mathrm{d}x$'. The LaTeX source is:

```
\int_{x_0}^{x_1} f(x) \,\mathrm{d}x
```

By contrast, consider what this mathematical statement actually *means*: a direct integral of a function $f(x)$ from x_0 to x_1. There is more detail in the typesetting of the statement than in the mathematics of it! In the cool package, this is written

```
\Integral{f(x)}{x,x_0,x_1}
```

In stex (the package name is cmathml for just the mathematics component of stex), it is

```
\CintLimits{x}{x_0}{x_1}{f(x)}
```

Another pertinent example is for representing derivatives. To write $\frac{\mathrm{d}f}{\mathrm{d}x}$ in LaTeX requires using an explicit fraction with more markup for the upright 'd': `\frac{\mathrm d f}{\mathrm d x}`. The packages cool and cmathml respectively use `\D{f}{x}` and `\Cddiff{x}{f}`. For multiple derivatives the benefits are even more obvious; `\D{f(x,y)}{x,y,z}` or `\Cpartialdiff{3}{x,y,z}{f(x,y)}` instead of

```
\frac{\mathrm d^3}
  {\mathrm d x\,\mathrm d y\,\mathrm d z}
    f(x,y)
```

to obtain

$$\frac{\mathrm{d}^3}{\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z} f(x,y).$$

I haven't spent much time with the more recent cmathml package, but my experiences with writing mathematics using the cool package have been very positive. The additional semantics using this notation isn't helpful from an academic sense of adding more meaning to the document source (although that's also a good thing). The real benefit is that it makes these maths constructions easier to type.

Based on the work of cool and cmathml, I believe that standardising some of the ideas for semantic markup of mathematics will benefit document authors (many of whom, after all, use similar macros in their own texts, albeit in an *ad hoc* way) and help in the automatic translation of mathematics written in LaTeX to other markup systems like MathML, and vice versa.

The unicode-math package will not address such ideas directly; it is purely a system to use mathematics with OpenType fonts. But as this package becomes more mature and can be used as a solid foundation for Unicode mathematics, then it will be time to start thinking seriously about formalising ideas behind 'semantic mathematics'.

7 A technical note on alphabet remapping

In TeX and LaTeX, using different fonts for alphabets such as `\mathbf` and `\mathscr` involved setting the 'math code' of the ASCII Latin letters to 'variable' and simply switching the math font. This meant that, internally, `\mathbf` and friends simply resulted in a font switch, which is efficient and straightforward (although sometimes tricky to juggle with only sixteen maths fonts in eight-bit TEX).

Unfortunately, things are not so simple with unicode-math. Within Unicode, each alphabet style (for bold and script and so on) is encoded in a distinct Unicode range. For example, the italic mathematical 'w' accessed with `$w$` is symbol U+1D464. The bold upright mathematical '**w**' (`$\mathbf{w}$`) is U+1D430. In order to switch from one to the other using a command like `\mathbf` requires that the mathcodes for all affected letters must change locally inside its argument. This isn't *too* inefficient, since assigning `\mathcode`s is pretty fast, but it's not particularly elegant. It would be easier not to support the `\mathbf{...}`-style syntax at all and instead refer to such symbols with macros, such as `\mbfw` for the bold '**w**'. But we must support the switching-style commands for backwards compatibility.

There are some alternatives to doing things this way, but they all have trade-offs. The simplest solution would be to use XƎTEX's input mapping feature that allows letters in the source to be transformed into other letters before typesetting. Thus, the 'variable math code' approach as used in LATEX could be used for Unicode maths alphabets. However, this system is less flexible (features such as Example 8 would be more difficult) and an alternative approach would be required for LuaTEX.

Another approach would be to use (math-)active characters for all maths symbols. In this approach, ASCII letters such as 'a' would be *active* in maths mode and expand (as if it were a macro) to a construction such as

```
\csname mathchar_\mathstyle_a \endcsname
```

where `\mathstyle` would resolve to 'up' or 'bf' (etc.) depending on the context and `\mathchar_up_a` and `\mathchar_bf_a` (etc.) would be defined accordingly with the appropriate Unicode maths glyph. This is more efficient for font switching but less efficient (and perhaps more fragile) when symbol remapping is not taking place. Using active characters is the technique used by the breqn package to do its automatic line breaking of mathematics, and extending that system for unicode-math would be quite logical. One way or the other, breqn compatibility is planned for unicode-math in the future.

Using LuaTEX for alphabet remapping (which is how ConTEXt's implementation works) is probably the best way to tackle this problem, but while unicode-math is written for XƎLATEX as well (and it will continue to be for the immediate future) we must stick with TEX-based programming solutions.

8 Experiences writing the package

Some aspects of writing the unicode-math package have been more organised than other LATEX code I've written. As more and more LATEX code is being developed publicly in source code repositories such as GitHub, BitBucket, and others, I would like to discuss quickly some of the infrastructure of this package's development.

8.1 Cross-platform development

The fontspec and unicode-math packages are both now targeted towards running on both XƎLATEX and LuaLATEX. Despite small differences in how certain things are done, this generally works well for both.

Most of the code in unicode-math and fontspec has been written (or re-written) with the expl3 programming interface. This has proven to be a very useable interface to numerous high-level programming constructs; expl3 allows more complex ideas to be easily realisable within the limitations of TEX macro programming.

In this shared XƎLATEX/LuaLATEX environment, Lua code is restricted to a minimum in order to minimise separate code branches for each engine, as much as possible. Functions in Lua are 'hidden' inside TEX macros, so all of the main programming in unicode-math resembles plain old TEX programming. I personally find this much easier to read than mixing Lua code and TEX macro code together.

8.2 Version control

I use the Git version control system, and for some time I've been using GitHub repositories[9] for most of my public code (LATEX and otherwise). GitHub provides free accounts for developers of open source software, and their site includes a very functional bug tracker/issue reporter per project. Tracking bugs over several years is certainly no fun with email.

I've had many people contribute code and provide feedback through the GitHub project page, and I highly recommend such a public development environment for all package developers.

The main advantages to these systems, for me, are the ease with which others can collaborate on code or documentation writing and with how issues can be resolved. Having a public code repository also allows users to access historical versions of the code, which can be important for those on legacy systems who cannot upgrade their distributions but

[9] `http://github.com/wspr`

need old versions of some packages that are no longer available on CTAN in their original form.

8.3 Test suite

Inspired by the test suite available for the LaTeX 2ε and LaTeX3 codebase, I implemented a test suite for unicode-math based on a 'visual diff' between the output of each test file compared to a known 'reference' output that had been compiled some time beforehand. At the time of writing there are 128 tests in total, the output of which are included all together as a separate documentation file in the unicode-math distribution as a rather complete set of minimal examples showing various aspects of the package and its features.

In hindsight, using an image-based test was perhaps not the best way to approach regression testing with LaTeX. The test scripts use ImageMagick's `compare` tool, which first discretises the PDF output to a bitmap and compares the pixels between the output and the reference. Unfortunately, due to rounding errors this technique is prone to the occasional 'false negative' in which the bitmap output of a test might change by a few (very small) pixels but there's nothing wrong with the output of the test itself.

An unintended benefit of this technique, on the other hand, is that any changes in the fonts I am using are immediately detected. This makes the unicode-math test suite a useful way for me to see what's actually changing when the fonts that I use in the test suite are updated.

However, the visual diff is slow and, as mentioned above, not always accurate (although it is repeatable, at least). A more reliable and efficient approach might use `\showbox` and `\tracingoutput` to create a detailed (textual) log of the TeX boxes generated in the output. This 'box log' can be checked for differences against a normalised result produced by a prior test run, and this is the technique used successfully by the LaTeX 2ε and LaTeX3 test suites [15].

Regardless of whether it's the most efficient or the most reliable technique to use, the test suite is still essential for catching bugs before I release new versions of the package to the public. I can change code without fear of unexpected problems in the behaviour of the package.

9 Conclusion

It's early days for Unicode mathematics. The work here shows the first steps for using OpenType fonts in LaTeX for mathematics; while we have done little to change the style of the input, there are still clear advantages in more consistent commands and Unicode input in the source. Being able to support new fonts without any extra LaTeX support files will hopefully spur new efforts in building new maths fonts. I am looking forward to seeing what happens in the future.

I would like to thank the TeX Users Group for supporting my attendance of the TUG2010 conference, and extend further thanks towards some people without whom the unicode-math package couldn't exist: Barbara Beeton for all her work with the STIX project and for her thoughtful correspondence; members of the LaTeX3 project for, well, everything; Khaled Hosny and others for their work with luaotfload and LuaLaTeX in general; all of those who have collaborated with, enthusiastically commented on, and especially tested the code; Jonathan Kew for XeTeX; and Taco Hoekwater et al. for LuaTeX.

References

[1] Claudio Beccari. Typesetting mathematics for science and technology according to ISO31/XI. *TUGboat*, 18(1):39–48, 1997. `http://tug.org/TUGboat/tb18-1/tb54becc.pdf`.

[2] Barbara Beeton. Unicode and math, a combination whose time has come — Finally! *TUGboat*, 21(3):176–185, September 2000. `http://tug.org/TUGboat/tb21-3/tb68beet.pdf`.

[3] Barbara Beeton. The STIX project — From Unicode to fonts. *TUGboat*, 28(3), 2007. `http://tug.org/TUGboat/tb28-3/tb90beet.pdf`.

[4] Barbara Beeton, Asmus Freytag, and Murray Sargent III. Unicode support for mathematics. Unicode Technical Note 25 Version 9, Unicode, Inc., 2008. `http://www.unicode.org/reports/tr25`.

[5] Thierry Bouche. Diversity in math fonts. *TUGboat*, 19(2):120–134, 1998. `http://tug.org/TUGboat/tb19-2/tb59bouc.pdf`.

[6] David Carlisle and Patrick Ion. XML entity definitions for characters. Technical Report W3C Working Draft 21, W3C, 2008. `http://www.w3.org/TR/xml-entity-names/`.

[7] Matthias Clasen and Ulrik Vieth. Towards a new math font encoding for (LA)TeX. *Cahiers GUTenberg*, 28–29, 1998. `http://cahiers.gutenberg.eu.org/cg-bin/article/CG_1998___28-29_94_0.pdf`.

[8] Michael Downes. Breaking equations. *TUGboat*, 18(3):182–194, 1997. `http://tug.org/TUGboat/tb18-3/tb56down.pdf`.

[9] Hans Hagen. MathML [in ConTeXt]. *MAPS*, 27:66–119, 2002. `http://www.ntg.nl/maps/27/18.pdf`.

[10] Hans Hagen, Taco Hoekwater, and Volker R.W. Schaa. Reshaping Euler: A collaboration with Hermann Zapf. *TUGboat*, 29(2):283–287, 2008. `http://tug.org/TUGboat/tb29-2/tb92hagen-euler.pdf`.

[11] Bogusław Jackowski. Appendix G illuminated. *TUGboat*, 27(1):83–90, 2006. `http://tug.org/TUGboat/tb27-1/tb86jackowski.pdf`.

[12] Jonathan Kew. XƎTEX Live. *TUGboat*, 29(1):146–150, 2008. `http://tug.org/TUGboat/tb29-1/tb91kew.pdf`.

[13] Johannes Küster. Newmath and Unicode. *Proceedings of EuroTEX 2005*, 2005. `http://tug.org/TUGboat/tb27-0/kuster.pdf`.

[14] Aditya Mahajan. Integrating Unicode and OpenType math in ConTEXt. *TUGboat*, 30(2), 2009. `http://tug.org/TUGboat/tb30-2/tb95mahajan-cmath.pdf`.

[15] Frank Mittelbach. A regression test suite for LATEX 2$_\varepsilon$. *TUGboat*, 18(4):309–311, December 1997. `http://tug.org/TUGboat/tb18-4/tb57mitt.pdf`.

[16] Luca Padovani. MathML formatting with TEX rules, TEX fonts, and TEX quality. *TUGboat*, 24(1):53–61, 2003. `http://tug.org/TUGboat/tb24-1/padovani.pdf`.

[17] Daniel Rhatigan. Three typefaces for mathematics. Master's thesis, University of Reading, 2007. `http://www.typeculture.com/academic_resource/articles_essays/pdfs/tc_article_47.pdf`.

[18] Will Robertson. Advanced font features with XƎTEX — the fontspec package. *TUGboat*, 26(3):215–223, 2005. `http://tug.org/TUGboat/tb26-3/tb84robertson.pdf`.

[19] Murray Sargent III. Unicode nearly plain-text encoding of mathematics. Unicode technical note 28, Unicode, Inc., 2006. `http://www.unicode.org/notes/tn28/`.

[20] Ulrik Vieth. Math typesetting in TEX: The good, the bad, the ugly. *MAPS*, 26:207–216, 2001. `http://www.ntg.nl/maps/26/27.pdf`.

[21] Ulrik Vieth. Do we need a 'Cork' math font encoding? *TUGboat*, 29(3):426–434, 2008. `http://tug.org/TUGboat/tb29-3/tb93vieth.pdf`.

[22] Ulrik Vieth. Experiences typesetting mathematical physics. In *Proceedings of EuroTEX*, 2009. `http://tug.org/TUGboat/tb30-3/tb96vieth.pdf`.

[23] Ulrik Vieth. OpenType math illuminated. *TUGboat*, 30(1):22–31, 2009. `http://tug.org/TUGboat/tb30-1/tb94vieth.pdf`.

[24] Joseph Wright. LATEX3 programming: External perspectives. *TUGboat*, 30(1):107–109, 2009. `http://tug.org/TUGboat/tb30-1/tb94wright-latex3.pdf`.

◇ Will Robertson
School of Mechanical Engineering
University of Adelaide, SA, Australia
`will dot robertson (at) latex-project dot org`

Math never seen

Johannes Küster

Abstract

Why have certain mathematical symbols and notations gained general acceptance while others fell into oblivion?

To answer this question I present quality criteria for mathematical symbols. I show many unknown, little-known or little-used notations, some of which deserve much wider use.

I also show some new symbols and some ideas for new notations, especially for some well-known concepts which lack a good notation (Stirling numbers, greatest common divisor and least common multiple).

1 Introduction

For TeX's "20th birthday it seems appropriate to present some fine points of mathematical typography and some ideas for new symbols and notations. Let's start with a quotation from *The METAFONTbook* [5, p. 8]:

> "Now that authors have for the first time the power to invent new symbols with great ease, and to have those characters printed in their manuscripts on a wide variety of typesetting devices, we must face the question of how much experimentation is desirable. Will font freaks abuse this toy by overdoing it? Is it wise to introduce new symbols by the thousands?"

We all know that METAFONT didn't become widely accepted. But even with other font editors, font freaks did not create new symbols by the thousands. So while maybe METAFONT was too complicated, and its way of thinking foreign to most designers, this can't be the real reason why only very few new symbols showed up. In fact, to design a new useful symbol is by no means an easy task, which I hope will become clear in the following. Just as we all do a lot more reading than writing, it is much easier to use existing symbols (e.g. with TeX) than to create good, useful new symbols (e.g. with METAFONT). So TeX with the character set offered by Computer Modern fonts (and the AMS fonts) shaped the typography of mathematics in the past 30 years.

This situation only changed with Unicode mathematics: Unicode now offers mathematical symbols literally by the thousands. But it gives little explanation and little usage information; many symbols are described only by shape, not by meaning. For many Unicode mathematical symbols it is not clear how to use them, and in many cases it is not clear whether there are any competing or superior notations.

2 Quality criteria

What makes a notation superior to another? What makes a symbol successful (in the sense that other mathematicians accept and adopt it)? The following list gives the most important quality criteria. A mathematical symbol or notation should be:

- readable, clear and simple
- needed
- international (or derived from Latin)
- mnemonic
- writable
- pronounceable
- similar and consistent
- distinct and unambiguous
- adaptable
- available

This list is certainly not exhaustive, but these are the most important points. Not all criteria are equally important, and some may conflict with others, so few symbols really fulfill all criteria. — Let me explain each point in turn.

Above all, a notation should be *readable* — but what constitutes readability? Certainly it comprises *clear* and *simple*. Also a notation should be short, at least it should make an expression shorter than writing out the same statement with words. Some of the other criteria contribute to readability as well.

When a good, widely accepted notation already exists, there is no need to invent a new one. So a new notation should be *needed* or *necessary*.

Most mathematical symbols are *international* (even if they are given different names in different languages and although there are different traditions in mathematical notation, e.g. the use of a dot or a comma as decimal separator). Of course a new notation should be international. In the case of an abbreviation (like "sin", "log", etc.), it should be derived from Latin, as most scientific terminology stems from Latin (and Greek), and so does the international vocabulary of mathematics.

A notation should be easy to learn, and its meaning should be easy to remember, at least after one has heard or read an explanation once; i.e. a notation should be *mnemonic*.

A lot of mathematics is still (and will be) written by hand (e.g. in a mathematician's research as the fastest way to denote his thoughts, on the blackboard, etc.). So a notation should be *writable*. In fact mathematical typography shows its close relation to handwriting in many places. But while written mathematics could always be explained by the writer (e.g. by the teacher at the blackboard), printed mathematics has to speak for itself. So in some cases it is desirable to go for greater differentiation in print than what is possible in handwriting.

A notation should also be *pronounceable*. Usually this is not a problem: for most notations there is a manner of speaking, although often language-specific and often not closely related to the notation (e.g. we call "$|a|$" the "absolute value of a", and we would do so whatever the notation would be). But we'll see an example below where a missing manner of speaking was a problem.

A new notation should be *consistent* with the general system of mathematical notation and *similar* to existing notations (e.g. for a symmetric relation one should choose a symmetric symbol, for a new kind of mapping one should choose some kind of arrow). In print, we can differentiate more than in handwriting, but still it is often preferable to stay close to existing notations.

As a special case of similarity, there are many concepts in mathematics which are *dual* or complementary to each other, and such dual concepts should be given dual notations (e.g. $<$ and $>$; $\wedge$ and $\vee$; $\cup$ and $\cap$; $\subset$ and $\supset$). Conversely, dual symbols should denote dual concepts.

In some cases dual symbols work against mnemonics. For many students it is difficult to remember which is which, so one has to use an additional memory aid (e.g. to remember which one of the the logic symbols $\wedge$ or $\vee$ denotes the "logical or", one might learn that $\vee$ reminds of Latin "vel", which means "or").

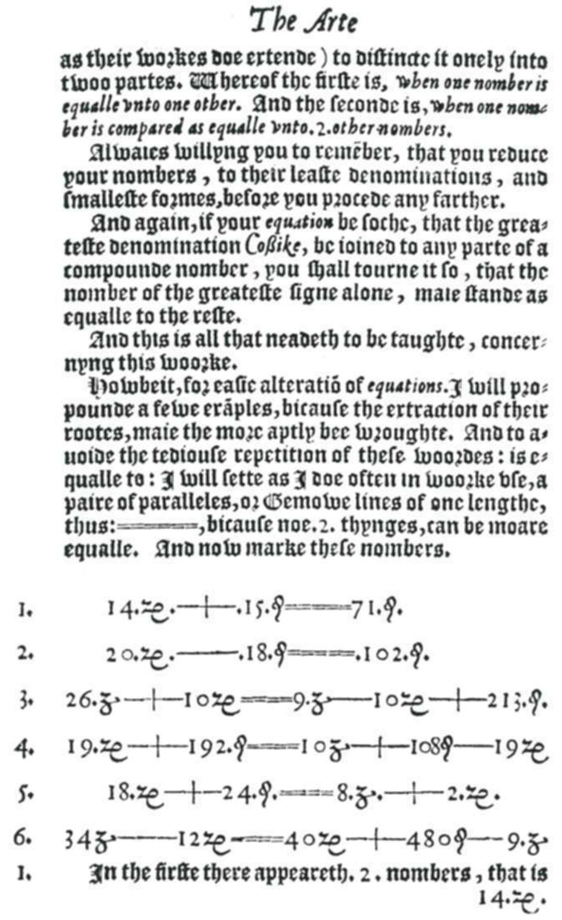
Of course a new notation should be *distinct* and *unambiguous*. Otherwise it will not be an improvement upon existing notations.

A notation should be *adaptable,* it should allow for manipulation. Also mathematical concepts are often generalized, and thus notation is often stretched to more general cases. A good notation allows for that.

A historical example is given by the competing notations $\dot{x}$ of Newton and dx of Leibniz. While Newton's notation was similar to existing notations and better fitted into the general system, the novel notation of Leibniz was superior, as it was more versatile and allowed for manipulation and generalization.

To give another example, the greatest common divisor of two integers a and b could be denoted as $\gcd(a, b)$; alternatively one might think of an infix notation, e.g. $a \top b$. When applied to three arguments both notations still work: $\gcd(a, b, c)$ and $a \top b \top c$. But one could also take the gcd of all elements of a set S. With the first notation, we can write this as $\gcd(S)$. Yet the alternative notation fails, it is not adaptable enough.

And last on our list, a symbol should be *available*. This is not really a criterion for quality, but rather for acceptance. The best notation does not help much if other people are not able to use it. In former times, this mainly meant availability at the printer's office — nowadays it means availability in a font, then a clear and simple shape which can be added to other fonts with ease, and of course inclusion in Unicode mathematics.


The Arte

as their woꝛkes doe extende) to diſtincte it onely into twoo partes. Whereof the firſte is, *when one nomber is equalle vnto one other.* And the ſeconde is, *when one nomber is compared as equalle vnto.2.other nombers.*

Alwaies willyng you to remēber, that you reduce your nombers, to their leaſte denominations, and ſmalleſte foꝛmes, befoꝛe you pꝛocede any farther.

And again, if your *equation* be ſoche, that the greateſte denomination *Coßike*, be ioined to any parte of a compounde nomber, you ſhall tourne it ſo, that the nomber of the greateſte ſigne alone, maie ſtande as equalle to the reſte.

And this is all that neadeth to be taughte, concernyng this wooꝛke.

Howbeit, foꝛ eaſie alteratiō of *equations*. I will pꝛopounde a fewe exāples, bicauſe the extraction of their rootes, maie the moꝛe aptly bee wꝛoughte. And to auoide the tediouſe repetition of theſe wooꝛdes : is equalle to : I will ſette as I doe often in wooꝛke vſe, a paire of paralleles, oꝛ Gemowe lines of one lengthe, thus: $=\!=\!=\!=$, bicauſe noe.2. thynges, can be moare equalle. And now marke theſe nombers.

1. 14.ze.—+—.15.ϙ=====71.ϙ.
2. 20.ze.——.18.ϙ=====.102.ϙ.
3. 26.ʒ—+—10ze====9.ʒ——10ze—+—213.ϙ.
4. 19.ze—+—192.ϙ====10ʒ—+—108ϙ——19ze
5. 18.ze—+—24.ϙ.====8.ʒ.—+—2.ze.
6. 34ʒ——12ze====40ze—+—480ϙ——9.ʒ

1. In the firſte there appeareth. 2. nombers, that is 14.ze.

Figure 1: Robert Recorde, *The Whetstone of Witte* (London, 1557). Recorde's explanation for his symbol "=" is given in the lines just above the display formulae.

3 Historical examples

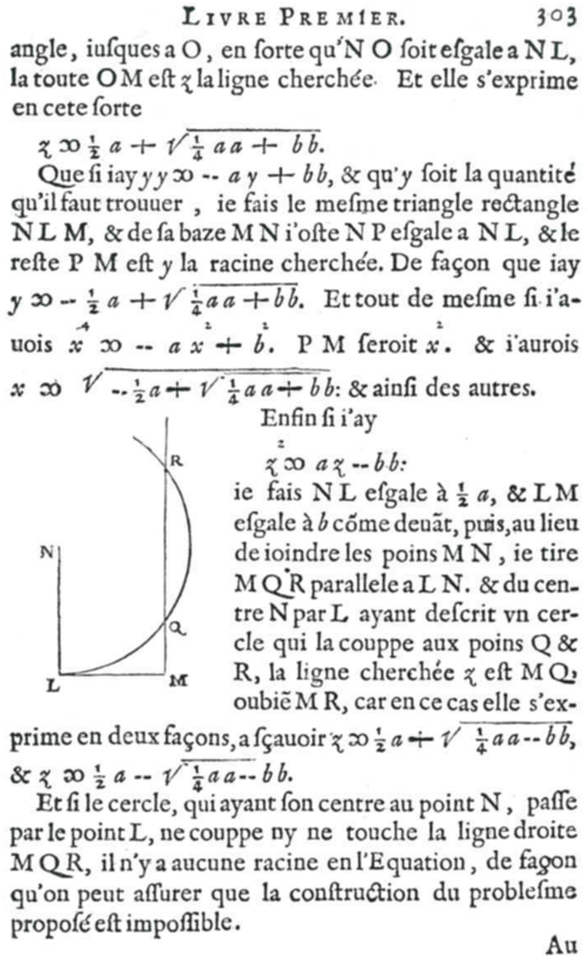
To illustrate these quality criteria, I will give a few historical examples, some unsuccessful, some successful. The historical information is mainly taken from [1].

3.1 Symbols for equality

Our modern symbol for equality "=" was introduced by Robert Recorde in 1557 in his book "*The Whetstone of Witte*" (see figure 1). Recorde explained his choice thus:

> "And to avoide the tediouse repetition of these woordes : is equalle to : I will sette as I doe often in woorke use, a paire of paralleles, or Gemowe lines of one lengthe, thus: $=\!=$, bicause noe .2. thynges, can be moare equalle."

("Gemowe" means "twin"). This is quite a famous example, as it is one of the very few cases where an author not only introduced a new symbol, but explained why he chose its particular form.


LIVRE PREMIER. 303

angle, iuſques a O, en ſorte qu'N O ſoit eſgale a N L, la toute O M eſt z la ligne cherchée. Et elle s'exprime en cete ſorte

$$z \text{ ꝏ } \tfrac{1}{2} a + \sqrt{\tfrac{1}{4} aa + bb}.$$

Que ſi iay $yy \text{ ꝏ } -- ay + bb$, & qu'y ſoit la quantité qu'il faut trouuer, ie fais le meſme triangle rectangle N L M, & de ſa baze M N i'oſte N P eſgale a N L, & le reſte P M eſt y la racine cherchée. De façon que iay $y \text{ ꝏ } -- \tfrac{1}{2} a + \sqrt{\tfrac{1}{4} aa + bb}$. Et tout de meſme ſi i'auois $\overset{4}{x} \text{ ꝏ } -- a\overset{2}{x} + \overset{2}{b}$. P M ſeroit $\overset{2}{x}$. & i'aurois $x \text{ ꝏ } \sqrt{-- \tfrac{1}{2} a + \sqrt{\tfrac{1}{4} aa + bb}}$: & ainſi des autres.

Enfin ſi i'ay

$$\overset{2}{z} \text{ ꝏ } az -- bb:$$

ie fais N L eſgale à $\tfrac{1}{2} a$, & L M eſgale à b cõme deuãt, puis, au lieu de ioindre les poins M N, ie tire M Q R parallele a L N. & du centre N par L ayant deſcrit vn cercle qui la couppe aux poins Q & R, la ligne cherchée z eſt M Q, oubiẽ M R, car en ce cas elle s'exprime en deux façons, a ſçauoir $z \text{ ꝏ } \tfrac{1}{2} a + \sqrt{\tfrac{1}{4} aa -- bb}$, & $z \text{ ꝏ } \tfrac{1}{2} a -- \sqrt{\tfrac{1}{4} aa -- bb}$.

Et ſi le cercle, qui ayant ſon centre au point N, paſſe par le point L, ne couppe ny ne touche la ligne droite M Q R, il n'y a aucune racine en l'Equation, de façon qu'on peut aſſurer que la conſtruction du probleſme propoſé eſt impoſſible.

Au

Figure 2: René Descartes, *La géométrie* (Leiden, 1637). The symbol "ꝏ" for equality appears throughout this page, e.g. as the second symbol in the first displayed formula.

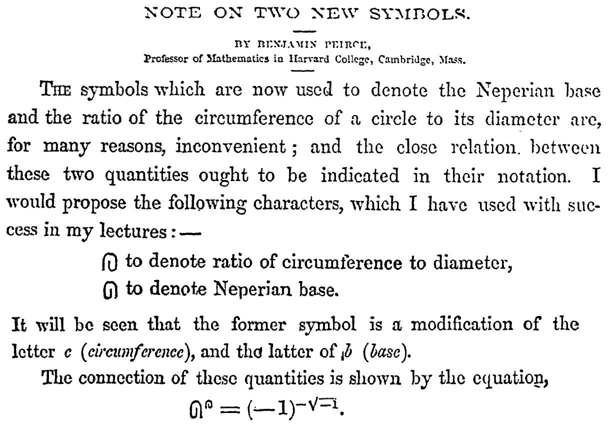
But 80 years later, René Descartes introduced a different symbol for equality, namely "ꝏ", in his book *La géométrie* (see figure 2). Descartes didn't give an explanation, so it is not clear why he invented a new symbol nor why he chose this particular form. Most likely, he was in need for a new symbol as he already used "=" for "plus or minus" (i.e. "±" in modern notation) elsewhere in his writings. The symbol of Descartes might stem from the ligature "æ", a common abbreviation for the Latin word "aequalis", but rotated 180 degrees. Typographically, it rather resembles a rotated "œ", or maybe it's even the astrological symbol for Taurus, turned sideways.

When we compare the two symbols (with our quality criteria in mind), we see that both symbols are mnemonic. Yet Recorde's symbol is simpler, and it is simpler to write. Equality is of course a symmetric relation, but the symbol of Descartes is not symmetric, and this is its main disadvantage. So it seems clear that "=" is the superior symbol.

But in fact these two symbols (and a few competing symbols as well) struggled for supremacy throughout the 17th century. Descartes was the more eminent mathematician, and with his important works his notation also spread. General adoption of "=" as the symbol for equality came only in the early 18th century, mainly because Leibniz and Newton both used it.

3.2 Symbols of Benjamin Peirce

In 1859, Benjamin Peirce introduced the symbols "ᑏ" and "ᑎ" to denote the numbers 3.14159… and 2.71828… (see figure 3). To my knowledge, these were the first significant symbols of American origin.


NOTE ON TWO NEW SYMBOLS.

BY BENJAMIN PEIRCE,
Professor of Mathematics in Harvard College, Cambridge, Mass.

THE symbols which are now used to denote the Neperian base and the ratio of the circumference of a circle to its diameter are, for many reasons, inconvenient; and the close relation between these two quantities ought to be indicated in their notation. I would propose the following characters, which I have used with success in my lectures:—

ᑏ to denote ratio of circumference to diameter,
ᑎ to denote Neperian base.

It will be seen that the former symbol is a modification of the letter *c* (*circumference*), and the latter of *b* (*base*).

The connection of these quantities is shown by the equation,

$$ᑎ^{ᑏ} = (-1)^{-\sqrt{-1}}.$$

Figure 3: Benjamin Peirce's symbols for the numbers 3.14159… and 2.71828… (from J. D. Runkle's *Mathematical Monthly*, Vol. I, No. 5 (February, 1859), p. 167–168).

Peirce's symbols were used by some of his pupils (among them his sons Charles Sanders Peirce and James Mills Peirce), but they weren't generally accepted, and they were never used in Europe. By checking our quality criteria, we can see a number of possible reasons.

First of all, the symbols were not really necessary: π and e were already widely used to denote these two numbers, and this was good enough for most mathematicians. Also they are not consistent with the general system of mathematical notation: constants and special numbers are usually denoted with letters, not with special symbols. Then these symbols were not readily available at the printer's office (of course this difficulty was often overcome with other symbols when demand was high enough). More importantly, the symbols ᑏ and ᑎ are not really mnemonic:

> "It will be seen that the former symbol is a modification of the letter *c* (*circumference*), and the latter of *b* (*base*)."

The connection between ᑏ and *c*, and between ᑎ and *b* is hard to see, and it is difficult to remember which is which. To make matters worse, James Mills Peirce used variations of his father's symbols (and also a special symbol for the imaginary unit, see figure 4), but the supposedly mnemonic connection to c and b does not get any clearer.

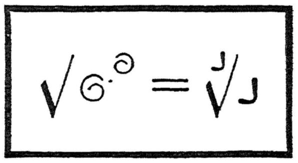


Figure 4: Variations of Benjamin Peirce's symbols (James Mills Peirce, *Three and Four Place Tables* (Boston, 1871)). In modern notation, this formula reads as $\sqrt{\mathrm{e}^{\pi}} = \sqrt[\mathrm{i}]{\mathrm{i}}$.

But on two of our criteria these symbols really fail: firstly, how should we pronounce these? The symbols do not provide a manner of speaking:

"ᑎ to denote ratio of circumference to diameter,
ᘉ to denote Neperian base."

Should we always say "ratio of circumference to diameter" and "Neperian base"? In comparison, to pronounce "π" and "e" is easy and fast.

Secondly, the symbols are dual, but the underlying concepts are not. Of course, 3.14159… and 2.71828… are connected in many interesting ways, but they are not dual to each other. So there are good reasons why these two symbols were not generally accepted.

3.3 Symbols for "floor" and "ceiling"

To denote the floor function (i.e. rounding a real number to the largest previous integer), Gauß introduced the bracket notation "$[x]$" (C. F. Gauß, *Theorematis arithmetici demonstratio nova* (1808)). This remained standard for a long time, and is sometimes even used today. But in 1962, Kenneth E. Iverson (in his book *A Programming Language*) introduced new notations

$\lfloor x \rfloor$ for the floor function, and
$\lceil x \rceil$ for the ceiling function.

These notations were readily accepted and are the standard notations today. Also they have been available in TeX and Computer Modern fonts right from the beginning, which certainly helped them to spread. Instead of the ambiguous $[x]$ (as brackets are used for many different concepts, not only for "floor"), we get a new, unambiguous notation $\lfloor x \rfloor$, and also a new, dual notation $\lceil x \rceil$ for the dual concept "ceiling" which didn't have a standard notation before.

These new notations are definitely very mnemonic, almost self-explanatory, and still they are not too far from the old notation, so they are consistent with the general system. Anyone used to the notation $[x]$ could learn and accept the new notations without difficulty.

These were very successful innovations indeed, and they meet all our quality criteria.

4 Unknown and little-known notations

Now I will discuss some important existing notations which deserve to be better known or to be used more often. All of these improve readability, but some only work in print, not in handwriting.

4.1 Usage of roman and italic letters

By careful usage of roman letters (or upright glyph shapes) one can greatly improve the readability of mathematical formulae. Instead of "roman" and "italic" I prefer to use the terms "upright" and "oblique" (or "slanted") here as these terms apply to all kind of glyphs, not only to letters. There's a little-known rule, best stated as

> Operators and constants with a fixed meaning should be set upright.

Important here is *"with a fixed meaning"*. Note that this rule only applies to operators and constants, not to functions or other concepts. Of course this only works in print, not in handwriting. This rule is seldom applied properly in TeX, probably because Computer Modern fonts did not supply upright lowercase Greek.

This rule applies at least to the following *constants with a fixed meaning:* Euler's number e, circle number π, imaginary unit i, Euler's constant γ (or C in European tradition), golden ratio ϕ; and at least to the following *operators with a fixed meaning:* differential operator d and partial differential operator ∂, difference Δ, Kronecker symbol δ_{ij}, and Christoffel symbols $\Gamma^{\kappa}_{\mu\nu}$. For consistency, all "ordinary" Greek uppercase letter must be italic then: $\mathit{\Gamma}, \mathit{\Delta}, \mathit{\Theta}, \ldots$

This list is not exhaustive, and the actual scope of this rule might depend on context. An author could extend the scope to some constants and operators which carry a fixed meaning throughout his text. In an encyclopedia of mathematics (with a wide range of topics and notations), applying this rule greatly improves readability, while e.g. in a monograph about all the fascinating properties of Euler's number, using an italic e might seem preferable, to separate it better from surrounding text — but even here I would apply this rule, with some careful spacing and kerning. Matters are more complicated when typesetting physics, as upright type is used here also for units, indices with a fixed meaning, particles, quanta, and quantum states; but even here this rule is useful.

My suggestion is to apply this rule to integral symbols as well: an upright integral symbol and an upright differential operator d serve as a kind of delimiters around the integrand:

$$\int_a^b f(x)\,\mathrm{d}x.$$

This is not the case when the integrand is a fraction: here the differential operator is often written in the numerator, but still, using an upright "d" increases readability.

When we look at a few examples, we see that this rule gives more structure and more clarity to formulae:

$$\bar{z} = a - ib = \rho\,(\cos\varphi - i\sin\varphi) = \rho e^{-i\varphi}$$

$$\bar{z} = a - \mathrm{i}b = \rho\,(\cos\varphi - \mathrm{i}\sin\varphi) = \rho \mathrm{e}^{-\mathrm{i}\varphi}$$

$$\left(\frac{d^2}{dr^2} + \frac{1}{r}\frac{d}{dr}\right)\ln\psi_0(r) = h(r)$$

$$\left(\frac{\mathrm{d}^2}{\mathrm{d}r^2} + \frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r}\right)\ln\psi_0(r) = h(r)$$

$$\int\limits_0^\infty \frac{e^{-at^2}\,dt}{t+x} = e^{-ax^2}\left(\sqrt{\pi}\int\limits_0^{\sqrt{ax}} e^{t^2}\,dt - \frac{1}{2}\,\mathrm{Ei}(ax^2)\right)$$

$$\int\limits_0^\infty \frac{\mathrm{e}^{-at^2}\,\mathrm{d}t}{t+x} = \mathrm{e}^{-ax^2}\left(\sqrt{\pi}\int\limits_0^{\sqrt{ax}} \mathrm{e}^{t^2}\,\mathrm{d}t - \frac{1}{2}\,\mathrm{Ei}(ax^2)\right)$$

For the most important constants and operators, I suggest to use the following TeX macros (somewhat analogous to the way to input these in some computer algebra systems):

`\E` for e, `\PI` for π, `\I` for i, `\df` for d.

Here `\df` could be defined as `\mathop` with an argument, which takes care of proper spacing, e.g.

```
\def\df#1{\mathop{\mathrm{d}{#1}}}
```

(proper font-specific spacing and kerning could be added to these macros with `\mspace` or `\mskip` and `\mkern`; in "newmath" encodings, the upright "d" is contained in "Math Core" to allow for kerning with math italic letters).

4.2 *O*-notation and Vinogradov symbols

For the well-known O-notation (invented by Paul Bachmann in 1894 and made popular by Edmund Landau), there is a little-known alternative with the so-called Vinogradov symbols, named after the Russian number theorist Ivan Matveevich Vinogradov (1891–1983). Unfortunately, I could not find when and where he introduced this notation.

So instead of $f(x) = O(\log n)$, equivalently we can write $f(x) \lessdot \log n$, or we could use the symmetric variant of "$\lessdot$" and reverse the order: $\log n \gtrdot f(x)$. This notation is used mainly in number theory. The two Vinogradov symbols are included in Unicode:

`uni2AA1`	⪡	"double nested less-than",
`uni2AA2`	⪢	"double nested greater-than".

In my opinion, the Unicode character names are misnomers. At least additional information is missing in Unicode that these two symbols are used as Vinogradov symbols.

The obvious TeX macro names for these symbols are `\subord` for "⪡" and `\supord` for "⪢", analogous to `\subset` and `\supset`.

The Vinogradov symbols must not be confused with

`uni226A`	$\ll$	"much less-than",
`uni226B`	$\gg$	"much greater-than".

Alas, very often "$\ll$" is used instead of "⪡", either because authors are unaware of the difference, or because Computer Modern fonts do not provide the Vinogradov symbols.

When we compare Vinogradov's notation and the O-notation we see that both have their advantages; neither is superior to the other.

Vinogradov's notation does not require additional parentheses. With its symmetric variant, it works in two ways: f ⪡ g and g ⪢ f. It better fits the general system of mathematical notation, and it better fits with other symbols, especially with Hardy's symbol "$\asymp$" for asymptotic equivalence:

$$(f ⪡ g) \wedge (g ⪡ f) \iff f \asymp g.$$

O-notation is similar to other Bachmann-Landau notations, namely o-, ω-, Ω-, and Θ-notation. Also it can be used in terms in arithmetic expressions:

$$f(x) = \frac{x}{\log x}\left(1 + O\left(\frac{1}{\log x}\right)\right),$$

with the downside that the O might be overlooked in a longer expression.

But O-notation makes strange use of "=", it is somewhat foreign to the general system. In fact, here "=" does not stand for "is equal to", but rather for "is of the order of" or "is a member of the class". So it would be more correct to use "$\in$". Of course this is well-known and has often been discussed. Still it is annoying, and so this might be a case where we should use greater differentiation in print: i.e. to keep "=" in handwriting as a short and fast notation, but to use an unambiguous special variant of "=" in print, maybe by creating a new special symbol.

4.3 Intervals

In exercise 18.14 in *The TeXbook* [4, p. 171], Knuth says *"Some perverse mathematicians use brackets backwards, to denote 'open intervals'"*, and the following formula is given as an example:

$$]{-\infty}, T[\, \times\,]{-\infty}, T[\,.$$

This notation for open intervals is taught in school at least in some countries (e.g. in Germany), and it is also recommended by a German DIN standard and an international ISO standard. So I prefer to be a perverse mathematician — but only in handwriting.

The answer to this exercise [4, p. 322] states *"Open intervals are more clearly expressed in print by using parentheses instead of reversed brackets"*, and the given formula is then written as

$$(-\infty, T) \times (-\infty, T).$$

But the notation "(a, b)" is overloaded with meanings: it is used to denote an ordered pair, coordinates, the greatest common divisor, etc. So this cannot be the best way to denote open intervals, neither in handwriting nor in print.

One simple way to improve the ambiguous notation (a, b) is to use a semicolon instead of a comma to separate the endpoints: $(a; b)$. This is especially useful when the decimal separator is a comma (which is the standard notation in some countries, e.g. in Germany): $(1{,}9; 3{,}8)$ is much more readable than $(1{,}9, 3{,}8)$.

This improvement works in handwriting as well, and it adds a lot of clarity for the reader, with minimal effort on the writer's side.

Still we can do better in print, namely by using special delimiters, already available in Unicode:

`uni2997`	⦗	"left black tortoise shell bracket",
`uni2998`	⦘	"right black tortoise shell bracket".

If there is such a thing as an "unknown standard", this certainly is one: at least one German manual of style [12] recommends these special delimiters for intervals, and one important German book [9, 10] uses these to very good effect. Of course, I also recommend these delimiters in my own writings about typography of mathematics [7, 8].

Just as brackets, these delimiters are reversed to denote open intervals:

$$⦗a; b⦘, \quad ⦗a; b⦗, \quad ⦘a; b⦘, \quad ⦘a; b⦗;$$

in an example formula, this looks like this:

$$⦘0; 1⦘ = \{\, x \in \mathbf{R} \mid 0 < x \le 1 \,\}$$

(note that we keep the semicolon as separator, as suggested above). The formula from above is now written as

$$⦘-\infty; T⦗ \times ⦘-\infty; T⦗$$

— admittedly, this is still not very readable, but it is not a nice example anyway (it might be preferable here to introduce an abbreviation for the given interval, say U, and to denote the formula as $U \times U$ or even as U^2).

For use in TeX, I recommend the following macros (with two arguments):

`\ivc{a}{b}`	for	$⦗a; b⦘$	("interval, closed"),
`\ivo{a}{b}`	for	$⦘a; b⦗$	("interval, open"),
`\ivco{a}{b}`	for	$⦗a; b⦗$,	
`\ivoc{a}{b}`	for	$⦘a; b⦘$.	

These macros can take care of proper kerning and spacing and of the semicolon as separator. We can alter these as necessary, e.g. whenever the special delimiters are not available in the used font. For larger versions, we can define macros as `\bigivc` etc. For automatic extension of delimiters (i.e. using `\left` and `\right`), we can define macros starting with an uppercase letter: `\Ivc` etc. In an similar way we can define macros for other delimiters with special meaning, e.g. `\abs` for absolute value $|a|$ or `\norm` for norm $\|a\|$.

5 New symbols and new notations

In this last section I will show some of my ideas for new symbols (even though some of these are not successful). The first few examples are rather minor points, but the last one seems quite important, at least in my opinion.

5.1 Vega

In mathematical finance (with the pricing of stock options) the so-called Greeks occur: Gamma, Delta — and Vega (these are quantities representing the sensitivities of derivatives):

$$\mathit{\Delta}_c = \frac{\partial C}{\partial S} = \phi(d_1) \quad \text{and} \quad \mathit{\Delta}_p = \frac{\partial P}{\partial S} = -\phi(-d_1)$$

$$\mathit{\Gamma} = \frac{\partial^2 C}{\partial S^2} = \frac{\phi(d_1)}{S\sigma\sqrt{T-t}}$$

$$\text{Vega} = S\sqrt{T-t}\,\phi(d_1)$$

While Gamma and Delta are denoted by Greek letters, Vega is either written out, or a script letter $\mathscr{V}$ is used, or sometimes even a lowercase Greek "nu" (ν) — a horrible misuse of notation. So it seems appropriate to design a new pseudo-Greek letter for Vega:

$$\mathit{\Upsilon} = S\sqrt{T-t}\,\phi(d_1).$$

This works well in uppercase (roman Υ and italic $\mathit{\Upsilon}$), but it would be hard to find a new distinct lowercase shape: this is overcrowded territory, with v, ν and upsilon υ.

Yet people in mathematical finance are quite inventive: there's not only Vega, but also "Vanna" and "Volga" (also called "Vomma") — all derived from or related to volatility, so all start with "V" — and then also "speed", "color", "charm", and "zomma". So while the idea for a special letter for Vega might be nice, it seems quite hopeless to design proper letter-like symbols for all these quantities.

5.2 Field extension

In algebra, a common way to denote a field extension "L over K" is by $L : K$, alternatively L/K or $L|K$ is used. All three notations are over-used: "$:$" for index of a subgroup; "$/$" for quotient ring, quotient group, division; "$|$" for "divides".

To get an unambiguous notation, my idea is a special "field extension colon", formed by two small triangles, thus: LːK (the international phonetic alphabet IPA contains a similar idea: a colon of two triangles pointing towards each other is used to denote length of a vowel).

By its asymmetric form it shows that L is the extended field. This is close to the current notation. It does not disturb the reader, but it is there to help when he is in doubt. Of course this can't be used in handwriting, and admittedly it is not very visible in print (and it needs high-quality printing). But it might work well in online documents, where the reader could magnify the text — yet a properly tagged pdf file might be more helpful.

5.3 Algebraic substructures

One can often read sentences like

Let $H \subset G$ be a subgroup of G.

This is logically wrong, as it tries to express two statements in one sentence (*"let H be a subset of G"* and *"let H be a subgroup of G"*), and the "$\subset G$" part is redundant, as any subgroup is a subset *ipso facto*. So it suffices to say

Let H be a subgroup of G.

But obviously people like to use symbols (it seems that to many people the logically correct version without symbols feels somewhat weaker), so I thought of a way to express "is a subgroup of" with a symbol. I suggest to use "$\subset$" with a small "G" set atop, or alternatively below on the right (preferably, the "G" should be upright and sans-serif, as this better separates it from other letters and makes clear it is part of the symbol):

$$H \overset{\mathsf{G}}{\subset} G \quad \text{or} \quad H \subset_{\mathsf{G}} G.$$

Thus the sentence from above is shortened to

Let $H \overset{\mathsf{G}}{\subset} G$.

This works for other algebraic structures as well:

$$S \overset{\mathsf{R}}{\subset} R, \qquad E \overset{\mathsf{F}}{\subset} F, \qquad U \overset{\mathsf{V}}{\subset} V, \qquad B \overset{\mathsf{A}}{\subset} A, \quad \text{etc.}$$

(ring, field, vector space, algebra). The obvious TeX macro names for these would be `\subgroup`, `\subfield`, etc., and one can easily construct these symbols in TeX with `\stackrel` and appropriate font switches.

But some names are not international, e.g. "field" (Latin "campus", French "corps", German "Körper") and "ring" (Latin "anellus", French "anneau") — according to our quality criteria, abbreviations should come from Latin, but "anellus" and "algebra" would both require an A. So to make this notation really useful, we need a list of standard abbreviations for algebraic structures.

5.4 Stirling numbers

Stirling numbers are used to convert from factorial powers to ordinary powers, and vice versa. For definition and properties see [6, pp. 66–69] or [3, pp. 257-267].

Stirling numbers of the first kind are notated with brackets:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} \quad (\text{for } k > 0),$$

with the initial conditions

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = \delta_{n0} \quad \text{and} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0.$$

Stirling numbers of the second kind are notated with braces:

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

with

$$\begin{Bmatrix} n \\ 1 \end{Bmatrix} = 1 \quad \text{and} \quad \begin{Bmatrix} n \\ n \end{Bmatrix} = 1.$$

These notations are similar to those for binomial coefficients and Eulerian numbers (denoted by $\binom{n}{k}$ and $\left\langle {n \atop k} \right\rangle$, respectively). According to Knuth:

> "These notations [...] have compelling advantages over the many other symbolisms that have been tried." [6, p. 66].

This is certainly true for existing symbols. But brackets and braces are used for many concepts, so how about new, distinct delimiters?

My idea was to keep close to the notation with braces and brackets, but to make it mnemonic by including an "s" form. But this proved to be unsuccessful. It seemed nice as an idea, it still seemed possible in handwriting, but when tried in print, it becomes clear that this is not working (at least it would need some reworking to make it useful).

First I tried new special brace-like delimiters, with an "s" in the top for Stirling numbers of the first kind, and in the bottom for those of the second kind. This looks just too obtrusive, too distracting:

$$\left\{ {n \atop k} \right\} \quad \text{and} \quad \left\{ {n \atop k} \right\},$$

especially when tried in a formula:

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\}.$$

Instead of helping the reader, it hampers readability.

Then I tried to keep brackets and braces, but now including a small "s" form (again in the top for the first kind, in the bottom for the second kind):

$$\begin{bmatrix} n \\ k \end{bmatrix} \quad \text{and} \quad \begin{Bmatrix} n \\ k \end{Bmatrix};$$

used in a formula:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}.$$

I consider this worse than the first version. Both just do not work. So it seems best to keep brackets and braces.

5.5 Greatest common divisor, least common multiple

My last example is one where I think a new notation is really needed. For "greatest common divisor" and "least common multiple", a standard notation is missing. To me, this is the most severe shortcoming in mathematical notation in general. So far mathematicians have failed to come up with a good notation, which is completely incomprehensible, as the concept of greatest common divisor is important and in wide use. It seems each language just uses some abbreviations:

English:	$\text{gcd}(a, b)$	and	$\text{lcm}(a, b)$
French:	$\text{pgcd}(a, b)$	and	$\text{ppcm}(a, b)$
German:	$\text{ggT}(a, b)$	and	$\text{kgV}(a, b)$
Dutch:	$\text{ggd}(a, b)$	and	$\text{kgv}(a, b)$
Polish:	$\text{NWD}(a, b)$	and	$\text{NWW}(a, b)$
Spanish:	$\text{mcd}(a, b)$	and	$\text{mcm}(a, b)$

The French version needs four letters (while almost all abbreviations in mathematics use three letters at most), the German version gives us the additional ugliness of mixed-case abbreviations. The Spanish version makes clear why Latin abbreviations would not work here: the "m" stands for "máximo" in one case and for "mínimo" in the other.

Also a very common notation for greatest common divisor, especially in number theory, is just (a, b). As I said above when discussing interval notation, this is ambiguous and over-used, and a reader always has to check the context to make sure what is meant. And there isn't any matching fixed notation for the least common multiple, so either $[a, b]$ or $\{a, b\}$ is used, but both notations always require an explanation in the text.

Formulae written in any of these notations are either lengthy, or not very readable, or not self-explanatory:

$$\gcd(F_m, F_n) = F_{\gcd(m,n)} \qquad (F_m, F_n) = F_{(m,n)}$$
$$\varphi([d, k]) = \varphi(d)\varphi(k) / \varphi((d, k))$$

and often they need additional explanation (this example with mixed notation is taken from [11, p. 63]):

Let $\mathscr{D}_1$ and $\mathscr{D}_2$ be sequences such that $((\operatorname{lcm} \mathscr{D}_1), (\operatorname{lcm} \mathscr{D}_2)) = 1$, where lcm *denotes the least common multiple of the members of a sequence, and the outer parentheses denote greatest common divisor. Then [...]*

Therefore I suggest the following new notation with new, special delimiters:

$\backslash a, b/$ for the greatest common divisor,
$/a, b\backslash$ for the least common multiple.

Many of our quality criteria are easily checked: obviously, these notations are *readable, needed, international, distinct and unambiguous* (at least in print).

They are also *writable,* although in sloppy handwriting they might be confused with 1 or 7 (depending on writing style) and possibly also with "floor" $\lfloor x \rfloor$ and "ceiling" $\lceil x \rceil$, but these two functions always have just one argument, while $\backslash a, b/$ usually has two or more arguments.

Of course these notations are *pronounceable:* in English, $\backslash a, b/$ is pronounced as the "greatest common divisor of a and b", and so analogously in any other language (just as it is the case with many notations which have their language-specific names).

They are *adaptable:* taking our example from above, we see that we could stretch the notation to more than two arguments: $\backslash a, b, c/$, but also to a single argument: $\backslash S/$ (e.g. when taking the greatest common divisor over all members of a set S).

They are *similar and consistent:* they fit into the general system, $\backslash a, b/$ is close to the common notation (a, b), and now we have *dual* symbols for dual concepts.

But are these new delimiters *available?* Well, yes — at least in my mathematical fonts ("Minion Math", as used here). But then it is such a simple, almost primitive design that it does not pose any problem to a font designer who wants to add these to his font. For the METAFONT sources of Computer Modern, it is a very simple addition: just take the code for "slash" or "backslash", add a third point whose coordinates are already known by the other two points, and connect the three points.

So the last remaining point is *"mnemonic"*, and the mnemonic aspect is the reason for the particular form I chose. For two positive integers a and b, the following chain of inequalities holds:

$$\backslash a, b/ \le \min(a, b) \le \max(a, b) \le /a, b\backslash.$$

Therefore $\backslash a, b/$ should remind of two vertices of a triangle pointing downwards to a lesser number (lesser than $\min(a, b)$, that is), and $/a, b\backslash$ is meant to point upwards to a greater number. Let's illustrate this with an example:

$$\underset{\substack{\backslash\quad/\\ 7}}{\backslash 14, 21/} \le \min(14, 21) \le \max(14, 21) \le \overset{\substack{42\\ /\quad\backslash}}{/14, 21\backslash}.$$

Of course the concept of greatest common divisor applies not only to positive integers. It could be generalized to negative integers, to polynomials, or to elements of a commutative ring. Then the above inequalities do not hold in general. Still the mnemonic is correct as it only serves to remember which is which of our notations.

When we apply these new notations to the above examples, we see that the formulae get shorter and do not need any additional explanation anymore:

$$\gcd(F_m, F_n) = F_{\gcd(m,n)} \qquad \backslash F_m, F_n/ = F_{\backslash m,n/}$$
$$\varphi(/d, k\backslash) = \varphi(d)\varphi(k) / \varphi(\backslash d, k/)$$

We could even agree upon saving parentheses, so the last formula might be written as

$$\varphi/d, k\backslash = \varphi(d)\varphi(k) / \varphi\backslash d, k/,$$

but this would be overdoing it maybe. Our last example above could now be shortened to

Let $\mathscr{D}_1$ and $\mathscr{D}_2$ be sequences such that $\backslash /\mathscr{D}_1\backslash, /\mathscr{D}_2\backslash / = 1$, where $/\ldots\backslash$ denotes the least common multiple of the members of a sequence. Then [...]

or even, when it's clear what we mean by $/\ldots\backslash$ with a single argument, simply to

Let $\mathscr{D}_1$ and $\mathscr{D}_2$ be sequences such that $\backslash /\mathscr{D}_1\backslash, /\mathscr{D}_2\backslash / = 1$. Then [...]

This is considerably shorter, and yet clearer than the original version.

One objection to the particular form of these new delimiters is that the notation seems counter-intuitive, especially compared with “floor” $\lfloor x \rfloor$ and “ceiling” $\lceil x \rceil$, where the “serifs” or little bars of the delimiters point to smaller and greater numbers just in the opposite way. But I do not see this as a contradiction: while the entire form $\backslash a, b /$ should remind one of a lesser number, the serifs remind one of the *greatest* number which divides both a and b.

I think that mathematicians can all agree that a good, distinct, international notation is really necessary here. The exact form of the symbols is up for discussion, but I can’t think of a more suitable form. I consider the notations $\backslash a, b /$ and $/ a, b \backslash$ necessary and important innovations, and I hope that mathematicians will adopt these symbols.

References

[1] Florian Cajori. *A History of Mathematical Notations.* Dover, New York, 1993. Reprint. Originally published in two volumes in 1928 and 1929 by The Open Court Publishing Company, Chicago.

[2] Friedrich Forssman and Ralf de Jong. *Detailtypografie*. Hermann Schmidt Verlag, Mainz, fourth edition, 2008.

[3] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading (MA), second edition, 1994.

[4] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading (MA), 1986.

[5] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading (MA), 1986.

[6] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading (MA), third edition, 1997.

[7] Johannes Küster. *Mathematischer Formelsatz*, pages 203–233. In *Detailtypografie* [2], fourth edition, 2008.

[8] Johannes Küster. *Sonderzeichen: Mathematikzeichen*, pages 382–389. In *Detailtypografie* [2], fourth edition, 2008.

[9] Fritz Reinhardt and Heinrich Soeder. *dtv-Atlas Mathematik*, volume 1: Grundlagen, Algebra und Geometrie. Deutscher Taschenbuch Verlag, München, twelfth edition, 2001.

[10] Fritz Reinhardt and Heinrich Soeder. *dtv-Atlas Mathematik*, volume 2: Analysis und angewandte Mathematik. Deutscher Taschenbuch Verlag, München, eleventh edition, 2003.

[11] R. J. Simpson and Doron Zeilberger. Necessary conditions for distinct covering systems with square-free moduli. *Acta Arithmetica*, 59(1):59–70, 1991.

[12] Friedrich Wilhelm Weitershaus, editor. *Duden Satz- und Korrekturanweisungen (Duden Taschenbuch Band 5).* Bibliographisches Institut, Mannheim, fifth edition, 1986.

◇ Johannes Küster
typoma GmbH
Karl-Stieler-Str. 4
D–83607 Holzkirchen
Germany
`info (at) typoma dot com`
`http://www.typoma.com`

Are virtual fonts obsolete?

Boris Veytsman

[Integrating third-party fonts] is unfortunately a messy topic. Forget about it unless you want to delve into many details of the TEX installation.

TEX Live Manual [1]

Abstract

Virtual fonts (VF) were created to address a shortcoming of TEX fonts: each slot address occupied exactly one byte, so there were no more than 256 different characters per font. Later, when PostScript fonts got popular, VF became the way of choice for integration of these fonts with TEX. Today new font formats can be directly read by the modern TEX engines, and, for example, XƎTEX can directly work with system fonts. There is a temptation to declare VF obsolete.

In this paper we show that there is much more in VF than just making PostScript fonts available for TEX. There are various tricks developed over the years that use VF technology to achieve new striking effects.

The aim of this paper is to convince the users to learn how to employ VF, and to convince the programmers of the new engines to provide the interface for font manipulation comparable to VF.

1 Introduction

Several years ago I attended a presentation of XƎTEX by Jonathan Kew. He typed in the editor

```
\font\x="Adobe Garamond Pro" at 11pt
\x This is Adobe Garamond Pro
```

and then clicked “Compile”. The preview window showed the phrase — in the beautiful Adobe Garamond Pro! I remember coming to Jonathan after the presentation and telling him, “You just closed one of the sources of my income!” I knew that for decades the use of third party fonts in TEX was one of the most difficult endeavors. It had a certain positive value for a consultant like me — and now this was going to change.

I was wrong. Actually in the years after that meeting I had a number of virtual font projects sponsored by various customers. Also, I learned a lot about fonts in TEX and came to the understanding that there is much more there than just typesetting the text in this or that third-party font.

In this paper I try to discuss some tricks possible with the traditional VF and compare them to the mechanism provided by XƎTEX.

2 A bit of history

In its early days TEX was practically synonymous with Computer Modern fonts. If you met a TEX document, you could bet it was typeset in Computer Modern. On the other hand, if you saw Computer Modern, you knew this text was typeset by TEX. Many people wanted to use the large number of free and commercial fonts available. However, there were two main obstacles to the use of these fonts with TEX: first, they were not in the METAFONT format, and second, they were not rich enough. TEX expected to find a lot of characters used for mathematical typesetting etc., and many free or commercial fonts lacked them — or did not have them in the proper places.

There were some hacks floating around — until Knuth proposed a unified interface of virtual fonts in TEX [7]. This interface used the fact that TEX itself is quite agnostic about the way the fonts are internally presented. What is needed is *metric information*: the dimensions of the letters, ligatures and kerning. The job of putting the letters themselves on paper or screen is done by *drivers* like `dvips`. The idea of virtual fonts is that the developer should create a metric file and a virtual font description with the instructions for the driver. These instructions could be quite complex, like “Take the letter A from this position in the file `abca`, the letter B from that position in the file `xyzz`, . . . ”. They also could contain transformations of the letters: expansion, contraction, slanting, kerning and ligature changes.

In this way one can overcome both obstacles for using “foreign” fonts with TEX. The font files could be in any format as long as the drivers recognized them. Also, if a font lacked certain characters, they could be taken from another font, maybe created with the explicit purpose to extend the original font for TEX.

The creation of virtual font files involved some repetitious steps. Thus there were several attempts to automate them. The most successful of them is probably the widely used program `fontinst` [5]. There is a great guide for this program by Philipp Lehman [8] and a very detailed discussion of many related topics including mathematical typesetting in the book by Alan Hoenig [4]. Still, even with these resources the setup of virtual fonts remains one of the most complex tasks for an apprentice TEXnician. Knuth gave his paper [7] a very apt title: virtual fonts indeed provide more fun to grand wizards.

In the next section we explore some tricks with virtual fonts.

3 Some virtual font tricks

As discussed in the previous section, the most direct application of virtual fonts is adding mathematical symbols to the free and commercial fonts that lack them. There are several good reviews of the many fonts created in this way. They include the survey of free fonts by Hartke [3], the "rogues gallery" in Hoenig's book [4], the tables in the LaTeX Companion [9], etc.

A good example of virtual fonts is the `mathptmx` package from PSNFSS [13]. It uses Times and Symbol PostScript fonts for mathematical typesetting (Figure 1).

Journal d'Analyse Mathematique uses Times, but with a slight modification: the letters are expanded in the horizontal direction. Math is provided by Belleek fonts [6]. Both the expansion and addition of math symbols are done with virtual fonts (Figure 2).

This example shows that VF can be used not only for mathematical typesetting, but also for text effects. Let us expand on this topic.

Eric Gill thought that italics should be used with lower case only, and the capitals should be taken from a matching Roman font [2]. To check the appearance of such combinations it is easy to "gillize" common fonts using the VF technique. The package `gillcm` [16] was created to demonstrate the setup of VF using this task. In Figure 3 we combine Roman uppercase letters with unslanted lower case italics.

Many examples of VF tricks can be found in the book [4]. In Figure 4 the Mantinia font is shown. Its unusual ligatures and swashes give the font a high decorative value. Of course the original font had all these characters, but it lacked ligature rules: evidently the designer had in mind manual typesetting only. Alan Hoenig added the ligatures through VF.

While this version of the font is nice to look at, its readability may suffer. Therefore Alan created another, more "subdued" version of this font usable for titling (Figure 5).

We conclude this section with the example of the use of Unicode fonts. The situation with these fonts is in a sense opposite to the situation with non-TeX fonts described in Section 2. They have *many* glyphs in one font — while TeX expects them to be in the different text and mathematical ones. However, VF can help in this case too. We can extract ranges of characters from a Unicode font and assign them to different TeX virtual fonts.

An example of this use of VF is `mathgifg` package [14] for use of Georgia font in text and math with TeX. Georgia is a nice Unicode font from Microsoft distributed with more or less recent Windows installations. It has Greek characters and mathematical symbols. The US Army Corps of Engineers sponsored the creation of TeX support for typesetting text and mathematics with it. The package [14] first separates the font into TeX virtual fonts, and then combines them for typesetting. Note that the Georgia font has "old style" (lowercase) numerals. While some designers used lowercase numerals in math in the past, it is probably too disturbing for a modern eye, so the package uses Franklin Gothic numerals (from another font distributed by Microsoft) in math and Georgia numerals in text. The results are shown in Figure 6.

4 Font setup with XeTeX

XeTeX at present has the full support of virtual fonts, as with other TeX engines. However, in this section we discuss the features specific to this engine which provide another way of dealing with fonts. Namely, XeTeX uses system libraries for font handling, and thus can "see" and directly use all fonts available in the current computer.

One of the striking things about XeTeX is the great ease with which third party text fonts are included in TeX documents. In this section we will discuss the LaTeX variant of fonts support based on fontspec [12] package. However, the corresponding variants for other flavors of TeX are relatively straightforward to set up. Actually the example in Section 1 was written with plain TeX commands.

With LaTeX and fontspec one can define *ad hoc* font selection schemes "on the fly". Consider the following invocation:

```
\fontspec[BoldFont={Helvetica Neue}]%
   {Helvetica Neue Ultralight}
```

As the result of this command *Helvetica Neue Ultralight* becomes the new Roman family. Moreover, *Helvetica Neue* becomes its bold variant, so macros like `\bfseries` and `\textbf` switch to this font.

The setup of mathematics fonts depends on the features of these fonts. If the OpenType font supports mathematics typesetting, then the experimental package unicode-math [11] can be used with the following easy interface:

```
\setmathfont[math-style=TeX]{Cambria Math}
```

However, if no mathematics support is provided by the font designer, then the situation becomes more complicated. The mathspec package [10] is intended to set up such fonts for typesetting mathematics in TeX. It has commands for font selection like `\setmathsfont`, `\setmathrm`, `\setmathsf`, etc.

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities $a_1, a_2, \ldots, a_m$. If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i}\int_\gamma f = \sum_{k=1}^{m} n(\gamma; a_k)\mathrm{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). *Let G be a bounded open set in $\mathbb{C}$ and suppose that f is a continuous function on G^- which is analytic in G. Then*

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

$\mathrm{A}\Lambda\Delta\nabla\mathrm{BCD}\Sigma\mathrm{EF}\Gamma\mathrm{GHIJKLMNO}\Theta\Omega\mho\mathrm{P}\Phi\Pi\Xi\mathrm{QRSTUVWXY}\Upsilon\Psi\mathrm{Z}$ 1234567890
$a\alpha b\beta c\partial d\delta e\epsilon\varepsilon f\zeta\xi g\gamma h\hbar\hslash\iota i\imath j\jmath k\kappa\varkappa l\ell\lambda mn\eta\theta\vartheta o\sigma\varsigma\phi\varphi\wp p\rho\varrho qrst\tau\pi u\mu\nu v\upsilon w\omega\varpi x\chi y\psi z\infty\propto\emptyset\varnothing\mathrm{d}\eth\backepsilon$

Figure 1: Times text with Symbol math (`mathptmx` package [13]), from [3]

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities $a_1, a_2, \ldots, a_m$. If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i}\int_\gamma f = \sum_{k=1}^{m} n(\gamma; a_k)\mathrm{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). *Let G be a bounded open set in $\mathbb{C}$ and suppose that f is a continuous function on G^- which is analytic in G. Then*

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

$\mathrm{A}\Lambda\Delta\nabla\mathrm{BCD}\Sigma\mathrm{EF}\Gamma\mathrm{GHIJKLMNO}\Theta\Omega\mho\mathrm{P}\Phi\Pi\Xi\mathrm{QRSTUVWXY}\Upsilon\Psi\mathrm{Z}$ 1234567890
$a\alpha b\beta c\partial d\delta e\epsilon\varepsilon f\zeta\xi g\gamma h\hbar\hslash\iota i\imath j\jmath k\kappa\varkappa l\ell\lambda mn\eta\theta\vartheta o\sigma\varsigma\phi\varphi\wp p\rho\varrho qrst\tau\pi u\mu\nu v\upsilon w\omega\varpi x\chi y\psi z\infty\propto\emptyset\varnothing\mathrm{d}\eth\backepsilon$

Figure 2: Times expanded text with Belleek math (`jamtimes` package [15])

Properly speaking, there is no such thing as an alphabet of italic capitals, and where upright or nearly upright italics are used ordinary upright Roman capitals go perfectly well with them. ¶Eric Gill, "An Essay on Typography", 1931.

Properly speaking, there is no such thing as an alphabet of italic capitals, and where upright or nearly upright italics are used ordinary upright Roman capitals go perfectly well with them. ¶Eric Gill, "An Essay on Typography", 1931.

Figure 3: Comparison of Computer Modern Italics with "Gillized" Computer Modern Italics: unslanted italic lowercase and Roman uppercase (package `gillcm` [16])

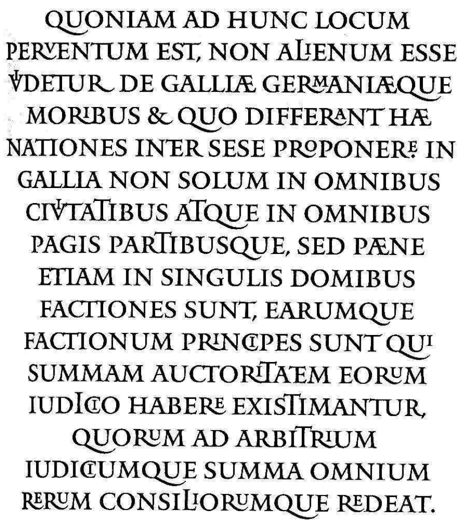
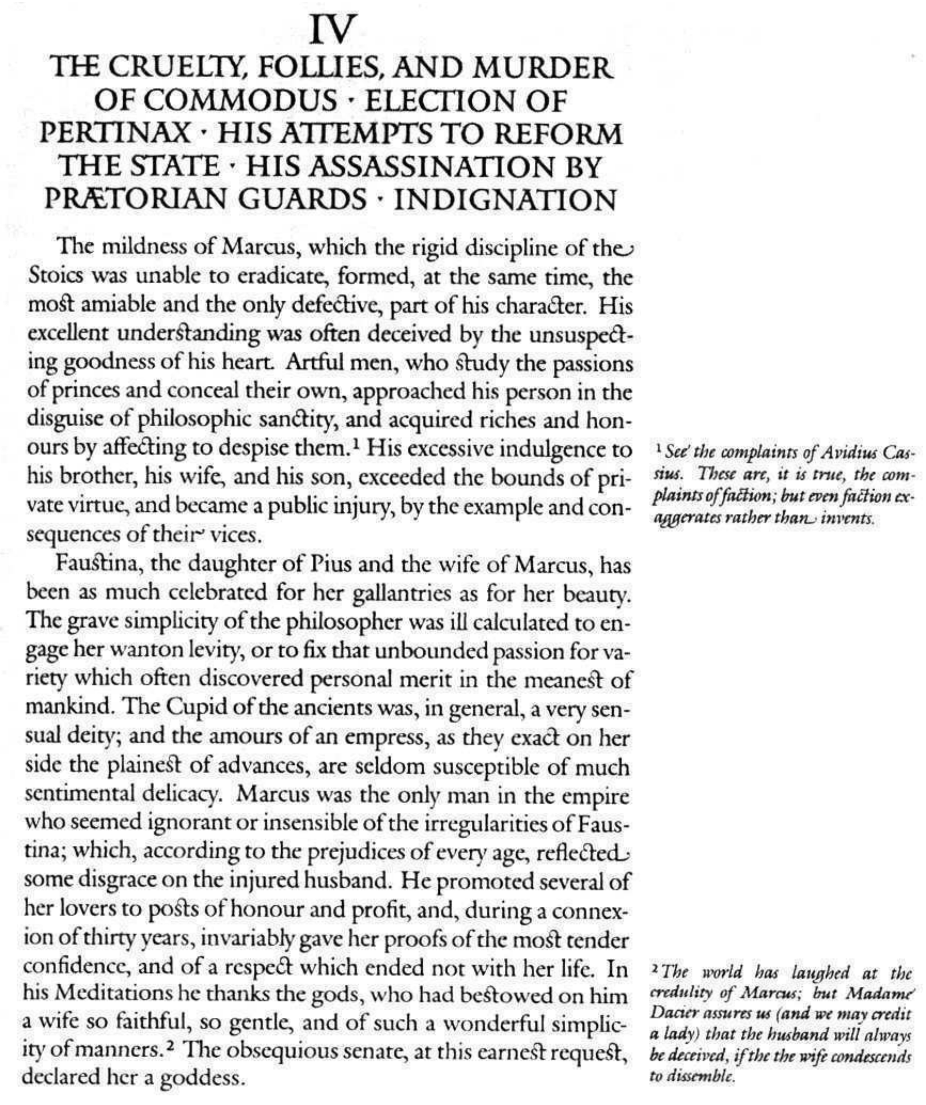

QUONIAM AD HUNC LOCUM
PERVENTUM EST, NON ALIENUM ESSE
VIDETUR DE GALLIÆ GERMANIÆQUE
MORIBUS & QUO DIFFERANT HÆ
NATIONES INTER SESE PROPONERE IN
GALLIA NON SOLUM IN OMNIBUS
CIVITATIBUS ATQUE IN OMNIBUS
PAGIS PARTIBUSQUE, SED PÆNE
ETIAM IN SINGULIS DOMIBUS
FACTIONES SUNT, EARUMQUE
FACTIONUM PRINCIPES SUNT QUI
SUMMAM AUCTORITATEM EORUM
IUDICIO HABERE EXISTIMANTUR,
QUORUM AD ARBITRIUM
IUDICIUMQUE SUMMA OMNIUM
RERUM CONSILIORUMQUE REDEAT.

Figure 4: Mantinia font with swashes and ligatures (from [4])


IV

THE CRUELTY, FOLLIES, AND MURDER OF COMMODUS · ELECTION OF PERTINAX · HIS ATTEMPTS TO REFORM THE STATE · HIS ASSASSINATION BY PRÆTORIAN GUARDS · INDIGNATION

The mildness of Marcus, which the rigid discipline of the Stoics was unable to eradicate, formed, at the same time, the most amiable and the only defective, part of his character. His excellent understanding was often deceived by the unsuspecting goodness of his heart. Artful men, who study the passions of princes and conceal their own, approached his person in the disguise of philosophic sanctity, and acquired riches and honours by affecting to despise them.[1] His excessive indulgence to his brother, his wife, and his son, exceeded the bounds of private virtue, and became a public injury, by the example and consequences of their vices.

Faustina, the daughter of Pius and the wife of Marcus, has been as much celebrated for her gallantries as for her beauty. The grave simplicity of the philosopher was ill calculated to engage her wanton levity, or to fix that unbounded passion for variety which often discovered personal merit in the meanest of mankind. The Cupid of the ancients was, in general, a very sensual deity; and the amours of an empress, as they exact on her side the plainest of advances, are seldom susceptible of much sentimental delicacy. Marcus was the only man in the empire who seemed ignorant or insensible of the irregularities of Faustina; which, according to the prejudices of every age, reflected some disgrace on the injured husband. He promoted several of her lovers to posts of honour and profit, and, during a connexion of thirty years, invariably gave her proofs of the most tender confidence, and of a respect which ended not with her life. In his Meditations he thanks the gods, who had bestowed on him a wife so faithful, so gentle, and of such a wonderful simplicity of manners.[2] The obsequious senate, at this earnest request, declared her a goddess.

[1] *See the complaints of Avidius Cassius. These are, it is true, the complaints of faction; but even faction exaggerates rather than invents.*

[2] *The world has laughed at the credulity of Marcus; but Madame Dacier assures us (and we may credit a lady) that the husband will always be deceived, if the the wife condescends to dissemble.*

Figure 5: Mantinia font (titling) and Galliard font (body text). From [4]

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities $a_1, a_2, \dots, a_m$. If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i}\int_\gamma f = \sum_{k=1}^{m} n(\gamma; a_k)\operatorname{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). *Let G be a bounded open set in $\mathbb{C}$ and suppose that f is a continuous function on G^- which is analytic in G. Then*

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

ΑΛΔ∇BCDΣEFΓGHIJKLMNOΘΩ℧PΦΠΞQRSTUVWXYϒΨZ 1234567890
aαbβc∂dδeϵεfζξgγhħℏıiȷjkκϰlℓλmnηθϑoσςφϕ℘pρϱqrstτπuμνυvwωϖxχyψz ∞ ∝ ∅⌀dð ∍
Text numerals: 123567890
Math numerals: 123567890

Figure 6: Georgia and Franklin Gothic fonts (`mathgifg` package [14])

One can use these commands to tell TeX to use a certain font for one of TeX's mathematical alphabets, for example

```
\setmathsfont(Digits)%
   [Numbers={Lining,Proportional}]%
   {Minion Pro}
\setmathsfont(Latin)%
   [Numbers={Lining,Proportional}]%
   {Minion Pro}
\setmathsfont(Greek)%
   [Numbers={Lining,Proportional}]%
   {Minion Pro}
```

The fontspec package allows for selecting font variants and alternates if these are provided by the font designer, for example

```
\fontspec[Variant=1]{Zapfino} Zapfino 1
\fontspec[Variant=2]{Zapfino} Zapfino 2
...
```

The same is true for variants in ligatures and kerning, for example

```
\fontspec[Ligatures=Rare]%
  {Adobe Garamond Pro} ...
\fontspec[Ligatures=NoCommon]%
  {Adobe Garamond Pro} ...
```

This very short discussion of the possibilities of XeTeX shows that it can help you to get from the font everything the font designer put there. Since OpenType format is very rich in features, this may be quite a lot indeed.

However, this approach has a flip side: it allows you to get from the font *only* what the font designer put there. Thus it assumes an ideal world of wise and TeX-aware font designers. If a font designer does not envision TeX-like automatic typesetting and, for example, assumes manual selection of characters for ligatures and manual kerning adjustments, you are out of luck.

In the past most font designers were of the second category (see the description of font adjustments in [4]). They often did not care about anything but either the crude typesetting without ligatures or the manual work with the so called *expert fonts*. It is true that OpenType provides many features for automatic typesetting; whether the font designers are going to use them is quite a different question.

5 Conclusions

The technique of virtual fonts, initially developed to allow the inclusion of third party fonts in TeX, became quite versatile in many different tasks. They can be used to achieve various effects in text and math. Unfortunately, they are known for their steep learning curve.

XeTeX provides an alternative way to incorporate third party fonts in TeX. This way allows using the full potential of such fonts — which can be impressive for feature-rich OpenType fonts. However, the "old" technique of virtual fonts allows low level font manipulation, including mixing different fonts. This manipulation is useful for adding to the font new features, not foreseen by the font creators.

References

[1] Karl Berry, ed. *The TeX Live Guide.* TUG, July 2010. `http://tug.org/texlive`.
[2] Eric Gill. *An Essay on Typography. With a new introduction by Cristopher Skelton.* David R. Godine, Boston, 2007.

[3] Stephen G. Hartke. *A Survey of Free Math Fonts for TEX and LATEX*, 2006. `http://mirror.ctan.org/info/Free_Math_Font_Survey`.

[4] Alan Hoenig. *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, and More*. Oxford University Press, USA, 1998.

[5] Alan Jeffrey, Rowland McDonnell, and Lars Hellström. *fontinst: Font installation software for TEX*, December 2004. `http://mirror.ctan.org/fonts/utilities/fontinst`.

[6] Richard Kinch. *Belleek: Free replacement for basic MathTime fonts*, August 1998. `http://mirror.ctan.org/fonts/belleek/`.

[7] Donald Knuth. Virtual fonts: More fun for grand wizards. *TUGboat*, 11(1):13–23, 1990. `http://www.tug.org/TUGboat/Articles/tb11-1/tb27knut.pdf`.

[8] Philipp Lehman. *The Font Installation Guide*, December 2004. `http://mirror.ctan.org/info/Type1fonts/fontinstallationguide`.

[9] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LATEX Companion*. Addison-Wesley Series on Tools and Techniques for Computer Typesetting. Addison-Wesley Professional, Boston, 2nd edition, 2004.

[10] Andrew Gilbert Moschou. *The mathspec package*, September 2009. `http://mirror.ctan.org/macros/xetex/latex/mathspec/`.

[11] Will Robertson. *Experimental Unicode mathematical typesetting: The unicode-math package*, June 2010. `http://mirror.ctan.org/macros/latex/contrib/unicode-math/`.

[12] Will Robertson and Khaled Hosny. *The fontspec package*, June 2010. `http://mirror.ctan.org/macros/xetex/latex/fontspec/`.

[13] Walter Schmidt. *Using Common PostScript Fonts with LATEX. PSNFSS Version 9.2*, September 2004. `http://mirror.ctan.org/macros/latex/required/psnfss`.

[14] Boris Veytsman. *LATEX Support for Microsoft Georgia and ITC Franklin Gothic in Text and Math*, July 2009. `http://mirror.ctan.org/fonts/mathgifg/`.

[15] Boris Veytsman. *Expanded Times Roman Fonts As Used in Journal d'Analyse Mathematique*, July 2010. `http://mirror.ctan.org/fonts/jamtimes/`.

[16] Boris Veytsman. *Unslanted Italic Computer Modern Fonts Based on Eric Gill's Ideas*, July 2010. `http://mirror.ctan.org/fonts/gillcm/`.

◇ Boris Veytsman
Computational Materials Science Center, MS 6A2, George Mason University, Fairfax, VA 22030
`borisv (at) lk dot net`
`http://borisv.lk.net`

TeX in the GLAMP world: On-demand creation of documents online

Boris Veytsman and Leila Akhmadeeva

Abstract

The acronym GLAMP is used to denote a combination of GNU/Linux, Apache, MySQL and Perl, Python or PHP, which now is one of the most common technologies for the creation of dynamic web pages. In this paper we describe the use of this technology for automatic creation of medical pedigrees.

To make drawing pedigrees easy for medical professionals, we put TeX and PostScript processing of their input on a Web site (`http://pedigree.varphi.com`). Here we cover both technical aspects of this (integration of TeX with GLAMP) and the preliminary results of using this site in an educational environment.

1 Introduction

One of the most popular Web packages today is the combination of GNU/Linux operating system, Apache HTTP server, MySQL database and Perl, Python or PHP scripting language [15]. Michael Kunze coined the acronym LAMP for such a combination [6]. Following the advice of Richard Stallman [8] we prefer to use the term GNU/Linux instead of Linux, so we use the acronym GLAMP instead. However, the letter G here is silent, like in 'GNU', so both versions of the word sound the same.

When a web site needs to perform on-demand typesetting, TeX seems to be a good choice for the back-end typesetting engine. Fortunately, it can be easily integrated in the GLAMP system (see, for example [14]). In this paper we describe yet another GLAMP/TeX solution, where we use TeX for the on-demand typesetting of graphical material.

The problem we addressed here is the problem of automatic creation of pedigree charts for genetic researchers and medical professionals [1, 2, 7]. Earlier we proposed a TeX/PSTricks-based program for pedigree creation [10, 12, 13]. Its spreadsheet-like interface turned out to be quite successful for non-technical users. However, there remained a problem of cross-platform installation and technical support of a complex program on many different architectures. At TUG'09 Karl Berry suggested we circumvent this problem by creating a web-based pedigree drawing tool. We followed his advice, and created such tool at `http://www.pedigree.varphi.com`. This paper describes the lessons learned from making this tool and the preliminary user survey.

2 Some challenges

One of the challenges for using TeX on the web is the power of this program. Powerful programs are potential security risks. Even when TeX is restricted from calling system programs (by disabling the `write18` mechanism), its ability to write files may be dangerous [4]. There are ways to deal with this problem: TeX can be limited to the current directory (this is the default in the TeX Live distribution [3]), and `chroot` may be used to additionally enforce this restriction. We decided to circumvent the challenge altogether: we do not allow the user to run arbitrary TeX files on the server. Instead we create the file from sanitized user input.

Another challenge is caused by the fact that medical pedigrees are used for, well, medical purposes. Laws in the USA and other countries enforce rather strict rules for safeguarding the privacy of medical records. Again, we chose not to implement the infrastructure for storing users' credentials and compartmentalizing the sensitive data. Instead we require the user to anonymize the patients' names and agree with the following statement:

> ☐ I agree to the Terms of Service and certify that my input does not contain individually Identifiable Health Information as defined by HIPAA Privacy Laws in the USA and similar laws of my jurisdiction.

Any attempt to create a pedigree without confirming this statement fails with the gentle reminder to check the box.

3 Under the hood: the missing M and the not so silent G

The setup of the server is very simple. The user input is collected with a spreadsheet-like interface (see `http://www.pedigree.varphi.com/cgi-bin/pedigree.cgi`). The result is fed to the Perl program which uses our library [11] to create a TeX file, which is processed by TeX/PSTricks program [9].

At this point an astute reader may have realized that a part of the GLAMP system is missing: we do not use MySQL or any other database. This is true: while the program [11] was created with an SQL interface in mind, we do not need it for the pedigrees created with this online tool. However, the letter M still plays a role in our solution.

We output the pedigrees in the format selected by the user: TeX, DVI, EPS, PDF, GIF or PNG. However, to lower the load on the server, we decided to create only the files requested by the user. This means that we need to decide which ones to

```
PSTOIMG = pstoimg -density 300 -antialias \
          -aaliastext -transparent
%.dvi:  %.tex
        latex  $<
%.ps:   %.dvi
        dvips -T"WIDTH cm, HEIGHT cm"  -o $@ $<
%.pdf:  %.ps
        ps2pdf $<
%.png:  %.ps
        $(PSTOIMG) -type png $<
%.gif:  %.ps
        $(PSTOIMG) -type gif $<
.SECONDARY: chart.ps chart.dvi
```

Figure 1: `Makefile` for pedigree creation

generate — or regenerate if the input has been updated. The trusted Unix `make` utility is the best tool for this purpose (and it starts with M!). So our program creates a special `Makefile` in the working directory (Figure 1). Note the special markers `WIDTH` and `HEIGHT`: since PSTricks has trouble communicating with TEX to tell it the dimensions of the chart, we calculate them separately and give them as options to `dvips`. In this way we can create charts exceeding “standard” paper dimensions.

The `Makefile` in Figure 1 uses the program `pstoimg`. This is actually a wrapper around Ghostscript; it is distributed as a part of the `latex2html` package [5]. Therefore the letter G in our version of GLAMP may not be silent after all.

4 Preliminary user survey

To test our program, we used the site `http://www.pedigree.varphi.com` during the classes at Bashkir State Medical University, Russia. There were 33 participants: 20 Russian-speaking fourth year students, 6 English-speaking fourth year students, 7 English-speaking first year students. The first year students were taking a Genetics course, and the fourth year students were taking a Clinical Neurology course. For the first year students this was the first introduction to medical pedigrees, while the fourth year students had some experience with manual creation of pedigrees in earlier courses. All students were asked to create their own pedigrees with our online tool. Additionally, the fourth year students used the tool to create pedigrees of their patients to be included in the patients’ medical chart. The plotting of a patient’s pedigree for the academic medical chart is a standard part of the curriculum for clinical classes. Usually this chart is created manually.

The students were asked to fill a questionnaire about their experiences. Some of their answers are shown in Table 1.

Question	Yes	No	Not Sure
1. Did you manage to create your first pedigree yourself?	83.9%	16.1%	—
2. Do you plan to use this program in your studies?	93.9%	6.1%	—
3. Do you plan to use this program in your future work?	75.8%	3.0%	21.2%
4. Do you like the chart you drew with the program?	100%	—	—

Table 1: Questionnaire answers

Class	Number of responses
Biology	27
Genetics	24
Neurology	19
Internal diseases	4
Oncology	3
Obstetrics	3
Psychiatry	2
All clinical subjects	1

Table 2: Students’ responses about the classes that can benefit from the use of the pedigree drawing program

The ease of use of the program was assessed by question 1 in Table 1 and by some specific questions. We asked the students how much time it took to create their first pedigree chart. The answers varied from 2 minutes to 40 minutes with the average of 19 minutes. We also asked the students to rate the ease of use of the software on a scale from 0 (impossible) to 10 (no problem at all). The answers varied from 5 to 10 with the average of 8.9.

Those participants who answered affirmatively to question 2 in Table 1 (“Do you plan to use this program in your studies?”) were asked to specify for which disciplines they would like to use the software. It was an open-ended question: the students wrote in the answers. Their responses are summarized in Table 2.

The students were asked the traditional question, what they liked and disliked the most about the software (open-ended questions). Their answers are shown in Table 3. Some of the problems found by the students are corrected in the recent version of the program. For example, it is now possible to save the results and return to the pedigree later.

Feature	Number of responses
What did you like most?	
Easy to use	26
Fast	13
Functionality	12
Free	7
Beautiful results	6
Exact	5
Many output formats	1
Helps in genetic studies	1
What did you dislike most?	
English only	12
Sometimes freezes	3
Cramped text	2
Online only	2
Only genetic relatives are plotted	2
Cannot save the input	2

Table 3: Features of the software most liked or disliked by the students

We asked the students to rate their general impression from the software on the scale from 0 (miserable) to 10 (outstanding). The answers varied from 7 to 10 with the average being 8.9.

Some pedigrees created by the students participated in the study are shown in Figure 2.

5 Conclusions

TEX is a natural typesetting engine in the GLAMP world. Our experience shows that it can be used not only for creating texts, but also for creating graphical materials with charts like those for medical pedigrees. Our web site `http://www.pedigree.varphi.com` demonstrates this point. We were successful in using it for teaching medical students. Our preliminary user survey demonstrates the potential of the online tool for teaching, research, genetic consulting and medical care.

Acknowledgements

One of the authors (LA) would like to acknowledge the support from TUG, Russian Federation President's Board for Support of Young Russian Scientists and Scientific Schools (grant MD-1195.2008.7), Russian Foundation for Basic Research (grants 06-04-58811-z, 08-04-09280-mob-z, 09-04-08060-z, 10-04-08177-z) and Ministry of Education and Science of the Russian Federation (grant 1256).

References

[1] Robin L. Bennett, Kathryn Steinhaus French, Robert G. Resta, and Debra Lochner Doyle. Standardized human pedigree nomenclature: Update and assessment of the recommendations of the National Society of Genetic Counselors. *J. Genet. Counsel.*, 17(424–433), 2008.

[2] Robin L. Bennett, Kathryn A. Steinhaus, Stefanie B. Uhrich, Corrine K. O'Sullivan, Robert G. Resta, Debra Lochner-Doyle, Dorene S. Markei, Victoria Vincent, and Jan Hamanishi. Recommendations for standardized human pedigree nomenclature. *Am. J. Hum. Genet.*, 56(3):745–752, 1995.

[3] Karl Berry, ed. *The TEX Live Guide*. TUG, July 2010. `http://tug.org/texlive`.

[4] Stephen Checkoway, Hovav Shacham, and Eric Rescorla. Are text-only data formats safe? Or, use this LATEX class file to pwn your computer. In *Leet'10. 3 USENIX Workshop on Large Scale Exploits and Emergent Threats*, April 2010. `http://www.usenix.org/event/leet10/tech/tech.html`.

[5] Nikos Drakos and Ross Moore. *The LATEX2HTML Translator*, March 2005. `http://mirror.ctan.org/support/latex2html`.

[6] Michael Kunze. Freeware web publishing system. *c't*, 12:230, 1998. English translation is available at `http://web.archive.org/web/20071212203835/http://www.heise.de/ct/english/98/12/230/`.

[7] Robert G. Resta. The crane's foot: The rise of the pedigree in the human genetics. *J. Genetic Couns.*, 2(4):235–260, 1993.

[8] Richard M. Stallman. Some confusing or loaded words and phrases to avoid (or use with care). `http://www.gnu.org/philosophy/words-to-avoid.html`, 2010.

[9] Boris Veytsman and Leila Akhmadeeva. *Creating Medical Pedigrees with PSTricks and LATEX*, July 2007. `http://mirror.ctan.org/graphics/pstricks/contrib/pedigree/pst-pdgr`.

[10] Boris Veytsman and Leila Akhmadeeva. Drawing medical pedigree trees with TEX and PSTricks. *TUGboat*, 28(1):100–109, 2007. `http://www.tug.org/TUGboat/Articles/tb28-1/tb88veytsman-pedigree.pdf`.

[11] Boris Veytsman and Leila Akhmadeeva. *A Program for Automatic Pedigree Construction with pst-pdgr. User Manual and Algorithm Description*, July 2007. `http://mirror.ctan.org/graphics/pstricks/contrib/pedigree/pedigree-perl`.

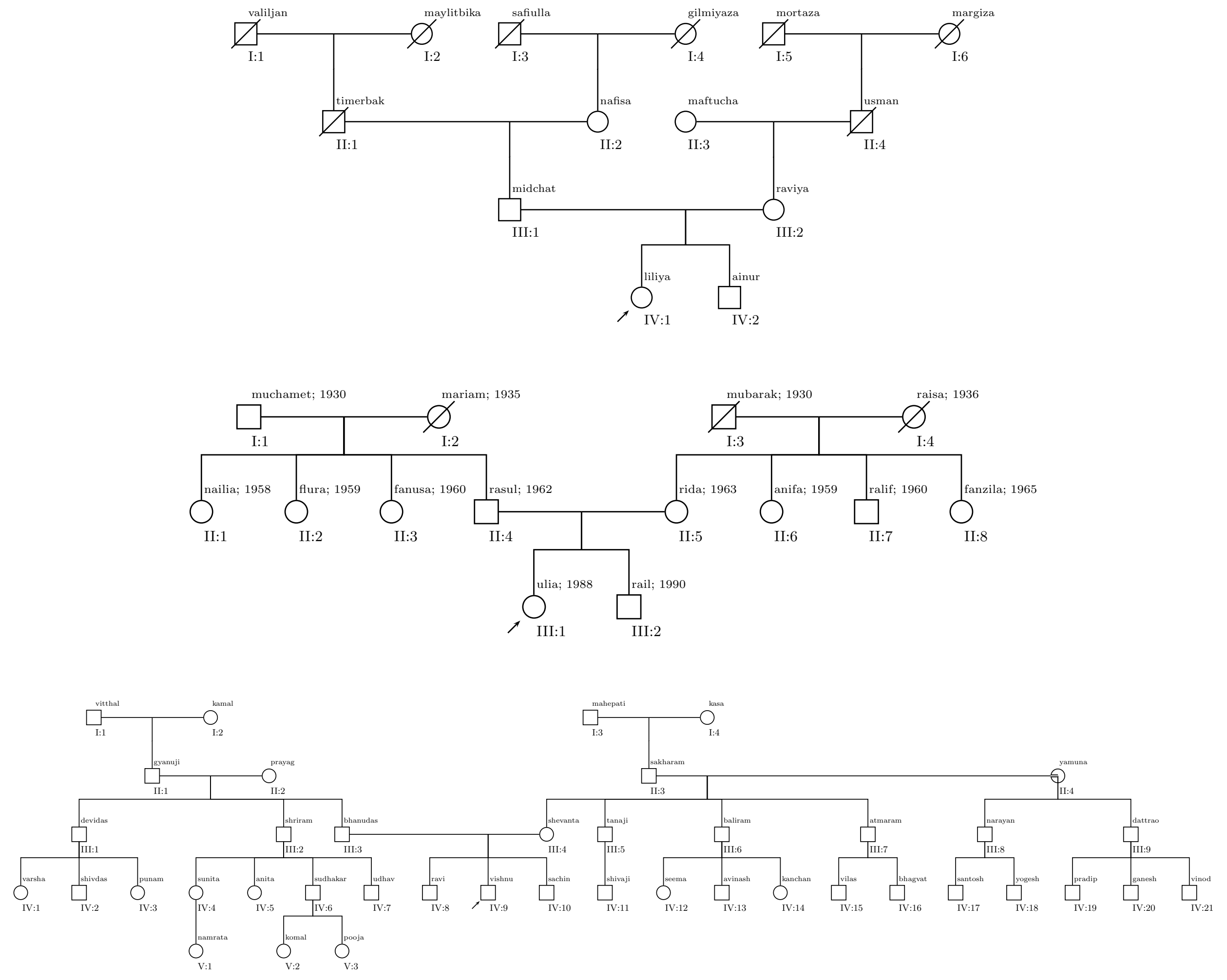


Figure 2: Pedigrees created by students; used with permission

[12] Boris Veytsman and Leila Akhmadeeva. Medical pedigrees with TEX and PSTricks: New advances and challenges. *TUGboat*, 29(3):484, 2008. `http://www.tug.org/TUGboat/Articles/tb29-3/tb93abstracts.pdf`.

[13] Boris Veytsman and Leila Akhmadeeva. Medical pedigrees: Typography and interfaces. *TUGboat*, 30(2):227–235, 2009. `http://www.tug.org/TUGboat/Articles/tb30-2/tb95veytsman-pedigree.pdf`.

[14] Boris Veytsman and Maria Shmilevich. Automatic report generation with Web, TEX and SQL. *TUGboat*, 28(1):77–79, 2007. `http://www.tug.org/TUGboat/Articles/tb28-1/tb88veytsman-report.pdf`.

[15] Wikipedia. LAMP (software bundle). `http://en.wikipedia.org/wiki/LAMP_(software_bundle)`, February 2010.

◇ Boris Veytsman
Computational Materials Science Center, MS 6A2, George Mason University, Fairfax, VA 22030
`borisv (at) lk dot net`
`http://borisv.lk.net`

◇ Leila Akhmadeeva
Bashkir State Medical University
3 Lenina Str. Ufa, 450000, Russia
`la (at) ufaneuro dot org`
`http://www.ufaneuro.org`

LaTeX profiles as objects in the category of markup languages

William F. Hammond

Abstract

The mathematical notion of "category" in the context of markup languages raises the idea of widespread use of reliable automatic translations between markup languages.

LaTeX profiles, which are dialects of LaTeX with a fixed command vocabulary where all macro expansions must be effective in that vocabulary, are suitable domains for defining translations to other profiles and, where sensible, to other markup languages.

The construction of reliable translators from several journal-neutral LaTeX profiles to many journal-specific LaTeX profiles would eliminate the need for technical editing in the production flow for academic journals.

1 Profiled usage of LaTeX

We now have 15 years of experience with various efforts to translate LaTeX into HTML, the language of the World Wide Web. We know that the task is more difficult than it appeared to be to many of us in the mid-1990s. Part of what makes LaTeX difficult to translate is that LaTeX, contrary to the impression one might gain from an initial reading of Leslie Lamport's book [4], has never been entirely formalized as a language unto itself independent of any implementation for processing the language. Indeed, the only implementation of LaTeX is that originating with Lamport — now maintained by The LaTeX Project (`http://www.latex-project.org/`) — as a very large macro package under TeX.

As slide 1 says, success with such translation requires profiled usage of LaTeX.

Slide 1: Translation of LaTeX

Question: What works well with translation software?

Answer: Profiled usage of LaTeX.

- Carefully limited command vocabulary.
- Tuned translation software.

Slide 2 provides a succinct statement of what I wish to suggest.

Slide 2: Today's Suggestion

formalize profiled usage

1.1 LaTeX profiles

What might be involved in formalizing profiled usage? First, I am suggesting the notion of *LaTeX profile* as the framework for multi-purpose LaTeX documents. Slide 3 specifies what is meant by "LaTeX profile":

Slide 3: The Notion of LaTeX Profile

- A dialect of LaTeX with a fixed command vocabulary where all macro expansions must be effective in that vocabulary.
- A language essentially equivalent to an SGML document type with a canonical XML shadow.

My project on *Generalized Extensible LaTeX-Like MarkUp* (GELLMU), `http://www.albany.edu/~hammond/gellmu/`, begun in 1998, underlies what I am suggesting today. The GELLMU "didactic production system" provides, in particular, a fairly elaborate example of what might be regarded as a LaTeX profile although some names in its command vocabulary are not part of current standard LaTeX usage. Slide 4 shows the source markup for a minimal document instance under this profile.

Slide 4: A Simple Example

```
\documenttype{article}
\surtitle{LaTeX Profiles}
\title{\latex{} Profiles: An Example}
\begin{document}

It's easier to learn to write in a
\latex{} profile than to learn to
write \latex.

The numbers $pi$, $i = \sqrt{-1}$,
and $e = \func{exp}(1)$ are related
by the equation
\[ e^{i\pi} = -1 \ . \]
\end{document}
```

Slide 5 shows a typeset rendition of this minimal document instance.

Slide 5: LaTeX Profiles: An example

It's easier to learn to write in a LaTeX profile than to learn to write LaTeX.
The numbers π, $i = \sqrt{-1}$, and $e = \exp(1)$ are related by the equation

$$e^{i\pi} = -1 .$$

Slide 6: The GELLMU Project

- Demonstrates that the ideas in this presentation can be implemented
- Provides a didactic document type which may be viewed as close enough to being a LaTeX profile that it can serve as a base for constructing profiles

From the outset I should make clear that:

1. The totality of standard usage of classical LaTeX, as we have it, is not suitable for modeling as a LaTeX profile.
2. There should be many LaTeX profiles.

1.2 HTML translation history

A close examination of our 15 years of experience with the most successful projects for the automatic translation of LaTeX to HTML will suggest that such translations should take place in two stages: (a) first, capture the LaTeX document as an XML document under a document type that closely models LaTeX, and (b) second, translate from that document type to HTML. This is, in fact, the design in the GELLMU project (see [6], [7], and [8]) except that GELLMU source, though LaTeX-like, is not LaTeX nor a dialect of present LaTeX.

The two stage design is also integral to the remarkably successful translator *LaTeXML*,[1] initiated around 2001 at the U.S. National Institute of Standards and Technology (NIST) under the very capable leadership of Bruce Miller for the purpose of providing a translation route to HTML (actually XHTML+MathML) for the *NIST Handbook of Mathematical Functions*. As time passed *LaTeXML* became the translation engine for the ambitious project "arXMLiv",[2] led by Michael Kohlhase of Jacobs University in Bremen, for translation to XHTML +MathML of Paul Ginsparg's large e-print archive,[3] originally housed at Los Alamos and now located at Cornell.

[1] `http://http://dlmf.nist.gov/LaTeXML/`
[2] `http://kwarc.info/projects/arXMLiv/`
[3] `http://www.arxiv.org`

An older very successful project for translation from LaTeX to HTML (including, as desired, XHTML+MathML) is *tex4ht*,[4] developed by the late Eitan Gurari of Ohio State University, in the years after 1995. While its core technique is the use of special-loading in DVI files generated by LaTeX with *tex4ht* macros, since the time that the project's scope was extended to generate a number of XML formats other than HTML, it has seemed clear to me that the *tex4ht* design would be improved by generating XML under a document type modeling the supported parts of LaTeX and then using standard XML libraries for translation to HTML and the various other XML document types.

2 The advantage of using a LaTeX profile

The crux of the problem in translating LaTeX documents to HTML is that LaTeX, as a whole, is not well-defined as a language unto itself. In the LaTeX community there is a well-known "newbie" question: *How can I know if my LaTeX document is correct?* A commonly heard answer is that a LaTeX document is correct if it runs seamlessly through LaTeX, the program.[5]

The whole of LaTeX, with maximally sane mixing and matching of packages and arbitrary "legal" (see Lamport [4], Appendix E) excursions into the world of plain TeX, is suitable for translation to printer languages (either specific printer languages or, more commonly DVI and PDF) but not suitable for reliable automatic translation to HTML or to common author-level document formats.

2.1 Language specification

When a list of LaTeX commands, perhaps 500 to 1500 in number,[6] is fixed, and when rules for usage of those commands in relation to each other, i.e., what commands are allowed to appear in a given context, are given, then one has something that is essentially equivalent to an SGML[7] document type. It is straightforward to construct an XML[8] shadow.

[4] `http://www.cse.ohio-state.edu/~gurari/TeX4ht/`
[5] A tougher standard, probably unreasonably tough, might be that a LaTeX document is correct if it runs through LaTeX and also through the *tex4ht* driver script `"mzlatex"` for generating XHTML+MathML.
[6] Imagine culling these commands from the LaTeX core and from one's favorite packages; but note that one is just assembling a list of commands and is not in any way thereby adopting segments of packages.
[7] *Standard Generalized Markup Language*, an ISO standard. See Goldfarb's *Handbook* [2] for a copy of the standard, and see `http://www.sgmlsource.com/` for amendments to the standard.
[8] *Extensible Markup Language*, a World Wide Web Consortium (W3C) standard [1].

The reason for the dual track is that dialects of classical LaTeX can be more closely modeled with SGML than with XML, which is SGML dumbed down for use on the World Wide Web. The difference here between SGML and XML is a question of convenience for authors. For example, if we want blank lines to begin new paragraphs without the tediously redundant explicit closing of previous paragraphs, then we want the umbrella of SGML in-house for the formal structuring of a LaTeX profile rather than the XML umbrella.

When a document language is implemented under the SGML umbrella, then

1. It is possible to know with varying degrees of precision, as required, when a document instance is technically correct.
2. Correct document instances can be translated automatically to other suitable formats with a very high degree of reliability.
3. Software libraries are available for most computer languages to facilitate automated translation and other forms of automatic processing such as, for example, automatic extraction of metadata.

2.2 Categories: A metaphor

At this point it is relevant to mention the mathematical notion of *category*.

Slide 7: Notion of Category

- A category consists of:
  1. Objects
  2. Arrows between objects
- Rule: An arrow followed by a second is also an arrow
- Relevance: to suggest a way of thinking about markup
- (No plans for actually using category theory)

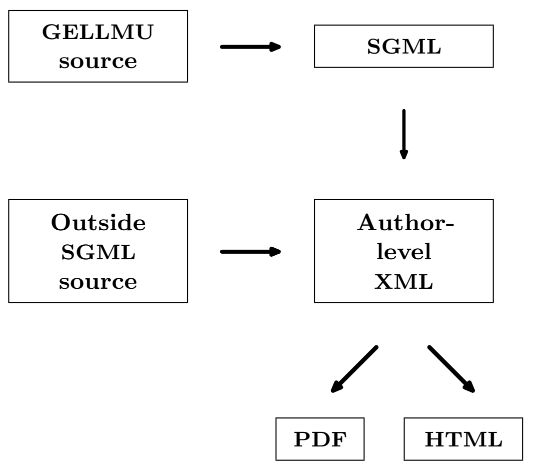
My reason for introducing the concept of *category* here is not for the purpose of applying category theory but for the purpose of suggesting different ways of thinking about how the community handles its documents. In particular, I'm trying to suggest a way of thinking about how the systematic use of translations between document formats can improve the way we operate with our documents. The largest relevant category here is the category of all markup languages:

Slide 8: The Category of Markup Languages

- Markup languages are the objects
- Translations are the arrows

Slide 9 shows the principal objects and arrows in the category of markup languages associated with the GELLMU didactic production system.

Slide 9: Objects and Arrows in GELLMU



Classical LaTeX is, more or less, a markup language. While many useful arrows can point toward LaTeX, few useful arrows point from LaTeX toward markup languages other than printer languages.

Slide 10: Classical LaTeX: An object in the category

(to the extent that classical LaTeX is a well-defined language)

- LaTeX is a reasonable translation target (for author-level markup languages).
- LaTeX is a poor domain for translation to languages other than printer languages.

Classical LaTeX does not fall under the umbrella of SGML, but classical HTML does. While classical HTML is not under the umbrella of XML, there is a variant of HTML called XHTML that is.

2.3 SGML and XML

Slide 11: SGML & XML

- SGML is a subcategory of the category of all markup languages
- XML is a subcategory of SGML
- XML is SGML made suitable for the World Wide Web

SGML is designed so as to facilitate the construction of arrows emanating from an SGML document type.

Slide 12: Good domains for translation

- Author-level SGML and XML document types are, by design, good domains for translation, i.e., arrows can flow **from** these document types.
- Arrows can be “chained”; these pipelines work well.

Let me suggest that in a properly designed system the chaining of arrows, i.e., translations, should take place at the command line. This makes it possible for the user to switch components (translators) from time to time, and it provides the opportunity for the user to run correctness tests at intermediate stages.

While the use of translations between document formats has not been part of regular LaTeX practice, the use of translations has not been far away.

2.4 Example: *Texinfo*

Texinfo, the language of the GNU Documentation System, was originally a vehicle for the simultaneous generation from a single source of (1) print output (via DVI) and (2) “Info”, an early form of hypertext predating HTML. Historically, *Texinfo* had been processed by the TeX engine with *Texinfo* macros for print and either by the GNU Emacs Lisp engine or by a free-standing C program, called *makeinfo*, for Info hypertext. When HTML came along, it became easily possible for the program *makeinfo* to generate HTML as well as Info. It has been obvious from the beginning to anyone asking the question that *Texinfo*, which has a careful language definition, is equivalent to an SGML document type. This was formalized in the year 2000 with Daniele Giacomini’s *Sgmltexi*,[9] and a few years later an independent XML version of *Texinfo* was incorporated in the *Texinfo* distribution.

[9] `http://www.archive.org/details/sgmltexi`

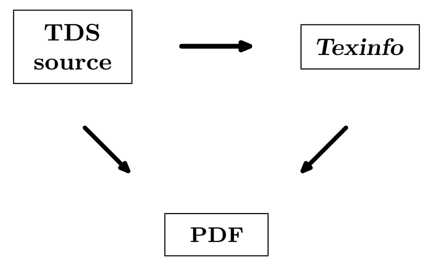
Because of its different markup syntax and command vocabulary relative to LaTeX, it is not sensible to think of *Texinfo* as a LaTeX profile. But aside from those differences, it is a long-standing example of something that is the same type of object as a LaTeX profile.

As an object in the category of markup languages *Texinfo* can serve usefully as both origin and target for arrows. This is often the case with author-level SGML and XML document types.

2.5 Example: The *tdsguide* document class

Another instance of the use of format translations lies in the production during the late 1990s of the specification document for the TeX Directory System (TDS), `http://mirror.ctan.org/tds.zip`. Ulrik Vieth devised a set-up under which the TDS specification could be written in LaTeX and processed either as a LaTeX document or as a *Texinfo* document (for Info and HTML outputs). Of course, there was a finite list of LaTeX commands (and environments) associated with the TDS LaTeX source. In effect, this system can be said to amount to the ad hoc construction — not formalized — of (1) a LaTeX profile and (2) a program for translating that LaTeX profile to *Texinfo*.

Slide 13: *tdsguide*: Two Routes to PDF



Note that although the LaTeX source `tds.tex` is very nicely structured, there are two reasons why a direct-to-html translation program might be expected to have trouble with it. First, it is written under the document class *tdsguide*; I am not aware of any direct-to-html translator with knowledge of that document class. Second, the document uses *manmac* syntax, which is part of plain TeX rather than LaTeX.

Moreover, inasmuch as it is reasonable to think about constructing a translation to regular LaTeX from any author-level XML document type, it is, in particular, reasonable to imagine the possibility of someone constructing a translation from (the XML guise of) *Texinfo* to LaTeX. That would offer a third (and different) route to PDF for *tdsguide* documents. Beyond that it is reasonable to think about constructing a translation from *Texinfo* to GELLMU article, and, following that with the translation to LaTeX provided for GELLMU article would provide a fourth route to PDF for *tdsguide* documents.

3 How LaTeX profiles might be deployed

There are many advantages.

3.1 Traditional LaTeX authors

We know that a very substantial portion of the community of LaTeX authors is interested in being able to "pull" HTML from LaTeX documents. There are frequent questions in the Usenet newsgroup `comp.text.tex` about using translation programs. There is a recurring theme in discussions there where the participants are seeking a LaTeX authoring interface — a better form of LaTeX markup — that is robust for translation to other formats, particularly HTML. Another recurring theme that is sometimes intertwined with the former theme is that of having a version of LaTeX that enables an author to focus on content. Of course, there is some irony in this latter theme in that Lamport in his book [4] explicitly states that this is the purpose of LaTeX — as is indeed the case, when LaTeX is used well. Thus, the latter theme may be interpreted to be about somehow "enforcing" the use of LaTeX as Lamport intended.

One or more LaTeX profiles sponsored by the LaTeX project could meet these requests.

3.2 LaTeX for beginners

With a suitably literate system for constructing LaTeX profiles it should be a relatively simple matter to produce a command glossary for a given LaTeX profile.

Combine the availability of a definitive command glossary with the enforcement of sane markup provided by routine structural validation in the process of running LaTeX on a document instance under a LaTeX profile, and it is not difficult to see why it should be easier for a beginner to learn a LaTeX profile than to learn classical LaTeX.

3.3 LaTeX's interface with established SGML document types

Although provision for typesetting SGML and XML has been a stated goal of the LaTeX3 Project, almost all of the Project's work so far has been focused on developing the infrastructure for writing document classes and packages. The community is just beginning to harvest the results of that work.

Another aspect of the situation with SGML and XML in relation to LaTeX is that while LaTeX is used by a very large community of authors, it does not seem to be the case that large crowds of original authors have migrated to well-known document types such as that of the Text Encoding Initiative (TEI), `http://www.tei-c.org`, which provides a vehicle for capturing electronic versions of classical printed texts, and the *DocBook*[10] document type, of the Organization for the Advancement of Structured Information Standards (OASIS), oriented (like *Texinfo*) toward documentation of technical work.

Well supported SGML and XML document types such as these typically are accompanied by formatting code for reaching both HTML for online presentation and PDF for print presentation.

Some tentative and rather incomplete experiments[11] I have undertaken, with the GELLMU didactic document type playing the role of a LaTeX profile, suggest to me that the process of formatting SGML and XML documents for print and HTML can be improved by first translating to a suitable LaTeX profile. This may seem contrary to common sense. The first point, however, is that numbering and cross-references can be worked out at an early stage, so that one can have consistency in this regard between the print and HTML outputs. The second point is metaphorical. Think of the trip from a source document to an end format as a downhill journey. One can jump the entire distance, or one can move more slowly, checking in at way stations along a path; at each of these stages one strives to retain as much as possible for that stage of the author's expressed intention as found in the source.

3.4 Suggestions for LaTeX evolution

Slide 14 presents a capsule description of what I think the LaTeX project should undertake in order to support the generation of HTML and other non-traditional formats from LaTeX source.

Because the LaTeX project needs to continue to provide support for legacy documents, a document instance written under a LaTeX profile must be clearly identifiable at the outset. My suggestion, used from the outset in the GELLMU project, is that one of these new document instances, prepared under

[10] `http://www.oasis-open.org/docbook/`

[11] These involved using *sgmlspl*, about which there is a section in *The LaTeX Web Companion* [3], and which is the staged translation workhorse in the GELLMU project, to begin building a translator from *DocBook* to GELLMU *article*, sufficient for handling four particular document instances.

a LaTeX profile, should begin with `\documenttype` rather than `\documentclass`.

Slide 14: Suggestion for the LaTeX Project

- Sponsor one or more reference profiles.
- Sponsor translations from reference profiles to PDF and HTML.

The argument of *documenttype* should be the name of the root element in the XML document type corresponding to the LaTeX profile being used. The name of the root element in an XML document type falls far short of specifying completely what document type definition (command list) is involved. With GELLMU `\documenttype` has an option whose content is typically a string that serves as a key to (or name for) a data structure that has been made known to the syntactic translator. (See §3.1 of the *GELLMU Manual* [5].) If this option is missing, the syntactic translator looks for a default value for that key corresponding to the name of the root element. While presently in GELLMU the document-type option points to information characterizing the document type and style choices are determined by the manner of running processors on a document instance, it would be possible with LaTeX profiles, if desired, to incorporate style choice information in the data structures behind these keys.

As with the ad hoc system created for the TDS specification (section 2.5) there will be various choices for processing a document instance prepared under a LaTeX profile. In particular, there are a number of choices for print formatting. Slide 15 indicates three possibilities:

Slide 15: Paper Typesetting of a LaTeX Profile

There are several possibilities:

- Translate to classical LaTeX
- Translate to ConTeXt
- Teach LaTeX itself to digest the profile

In case it is not completely clear, I should point out that the third of these routes might be substantially different from the first two in that the latter probably should involve first explicitly generating the XML shadow of the document instance and then translating from there. If one is also going to have HTML output, it would, of course, be expected that the HTML output would be translated from the same XML shadow. With the third route, however, there will be the question of bypassing generation of a formal XML shadow and then losing the advantage of structural validation — or maybe constructing the XML shadow for structural validation and for staging non-print outputs but generating the print output directly from the source.

At this point I hope that the suggestions in Slide 16 will be obvious.

Slide 16: Publishing

- Encourage maintainers of XML document types to reach HTML and PDF by translating first to reference LaTeX profiles.
- Encourage authors to submit articles to journals as LaTeX instances under reference profiles.

Technical editing by a journal should not be necessary when an article is submitted under a reference LaTeX profile. The journal will have its own profile that is a modification of a reference profile enabling the journal to incorporate metadata. For each article a journal will prepare metadata, and the journal's processing stream will merge that metadata with the author's source to generate a document instance under the journal's profile that, in turn, will be processed to end formats using the journal's formatting software.

4 Issues with implementation

Within the GELLMU project, particularly its didactic production system, there is a demonstration that everything discussed here is possible. The question is how these ideas might be introduced so as to provide as smooth as possible a transition in authoring techniques for those LaTeX users who wish to avail themselves of the advantages of source markup that is amenable to generalized processing rather than only processing for print by TeX engines.

4.1 Syntax

The discussion following slide 15 suggests that a possible route to print for a document instance under a LaTeX profile might bypass the profile's XML shadow. There are related questions:

1. Might a modern engine like `luatex` be a good vehicle for generating the SGML shadow? Similarly, might such an engine be a better vehicle than the GNU Emacs Lisp engine for generating the *Texinfo* version of the TDS specification (see section 2.5)?

2. Shall the engine used to generate the SGML shadow use knowledge (unlike the GELLMU syntactic translator) of the corresponding document type definition[12] in so doing? For example, shall this stage of processing be allowed to know that `\frac` takes two arguments, numerator and denominator, and to know what the names of those arguments are?

In regard to the second of these questions, if the processor for generation of the SGML shadow is allowed to use knowledge of command vocabulary, then the syntactical requirements for a document instance could be somewhat looser than in GELLMU. For example, as in LaTeX but not in GELLMU, spaces could be allowed between the successive arguments and options in a command invocation. This would be both possible and, in a sense, author-friendly. However, I do not recommend loose syntax. The syntactic tightness of GELLMU was motivated by that of *Texinfo*. The enforcement of syntactic tightness can help prevent author errors. Moreover, the overall design of processing is more "modular" when the first stage deals only with syntax.

Also, if the processor for generation of the SGML shadow incorporates knowledge of the command vocabulary, then it may be reasonable for that processor to write the XML shadow directly without first going through an SGML shadow and then relying on an SGML parser and a subsequent processor with knowledge of the command vocabulary to write the XML shadow.

4.2 Counters

While LaTeX has infrastructure for managing counters, there is nothing entirely parallel in the SGML world. Document types can provide hooks for counting, and the processors operating on those document types can take those into account. Section 5.5 in the *GELLMU Manual* [5], "Labels, References, and Anchors", explains the approach to this in the GELLMU didactic production system.

Where there are to be multiple end formats of a given document instance, it is best if counters, numbering, and cross-references are managed centrally, as in the GELLMU didactic production system, so that there is consistency among the various end formats.

[12] That the GELLMU syntactic translator operates only on syntax, largely without knowledge of command vocabulary, enables the syntactic translator to be employed for writing virtually any SGML or XML document type using directly the vocabulary of that document type with the advantage for the author of being able to use GELLMU's *newcommand* with arguments.

4.3 Names

For a faithful representation[13] of source markup under a LaTeX profile as an XML document every item of LaTeX markup needs to have a name.

For the document type of the GELLMU didactic production system the minimum number of characters in a command name is three. One and two character names are reserved for user macros. While this is not necessary, I recommend it.

A number of frequently used commands in classical LaTeX do not have names. The SGML and XML shadows need names for them. For example, `~` is "non-breaking space" in LaTeX. In the GELLMU didactic production system it becomes the empty element named *nbs*.[14]

While the argument in a LaTeX command such as `\emph` simply corresponds to the content of an element `<emph>` in the XML shadow, the arguments in a command taking more than one argument such as `\frac` need to be named. For example, `\frac{3}{7}` would be represented in the XML shadow as

`<frac><numr>3</numr><denm>7</denm></frac>` .

4.4 Special ASCII characters

One commonly sees the 128 characters in the ASCII character set in a table of 8 rows of 16 characters each. The first 2 rows are control characters that are, apart from newlines, non-printable and not used either in LaTeX or in SGML document types. Thus there are 6 rows of 16 characters that are the printable ASCII characters except for the very last of these, which is not printable and not relevant here. Within the ASCII realm, we need to deal with the 95 printable characters. Of these 62 are alphanumeric — upper- and lowercase letters and numerals. The 33 remaining printable ASCII characters are the "special" ASCII characters.

Within the realm of electronic formats each of the 33 special ASCII characters is a candidate for use as a control character of some type. For this reason it is wise that a name be provided for each of these characters in a LaTeX profile. This is the case in the GELLMU didactic document type. Thus, for example, `{` has markup meaning in LaTeX, and one may use `\{` to place an actual left brace in one's document; in the GELLMU didactic production system `\{` is represented by the empty element `<lbr/>` in the XML shadow. The character `@` is only a bit different. Normally it is perfectly safe to use this character for

[13] Faithful to the source apart from *newcommand*s that should appear expanded.

[14] Of course, "non-breaking space" is the Unicode character U+00A0, but writing that in the XML shadow would make the shadow not a faithful representation of the source.

itself in a LaTeX document, and normally there is no harm in passing it through as itself to the XML shadow. If, however, the author, as in the case of the TDS specification (see 2.5), wants to translate the document to *Texinfo*, there is suddenly a problem since `@` has markup significance in *Texinfo*.

For each of these 33 special characters there can be found contexts where they have markup or "control" significance. Names should be available for all of them.

4.5 Non-ASCII characters

Because TeX and LaTeX originally handled only 7-bit characters, LaTeX has legacy names for characters that are no longer strictly necessary or, at least, almost no longer strictly necessary in the sense that we expect Unicode-capable modern TeX engines soon to be mainstreamed. For example, there is the character 'ł', U+0142, which may be entered as `\l`[15] in LaTeX and as `&lstrok;` in classical HTML.

For the long term it seems clear that names for Unicode characters beginning with the Latin 1 range of Unicode are not needed.

4.6 CSS

CSS stands for "Cascading Style Sheets", which is a web technology for controlling the appearance of HTML and of arbitrary author-level XML document types for display in mainstream web browsers.

At the point where some LaTeX documents have XML shadows, one may become interested in writing CSS sheets for governing the display of those XML shadows in web browsers. Beyond that the question arises whether there might some day be gain in using CSS sheets to govern, at least in part, the typesetting of such a document by LaTeX.

References

[1] Tim Bray, Jean Paoli, & C.M. Sperberg-McQueen, *Extensible Markup Language (XML) 1.0*, World Wide Web Consortium Recommendation, 10 February 1998, `http://www.w3.org/TR/1998/REC-xml-19980210`, currently superseded by `http://www.w3.org/TR/REC-xml/`.

[2] Charles F. Goldfarb, *The SGML Handbook*, Oxford University Press, 1990.

[3] Michel Goossens and Sebastian Rahtz et al., *The LaTeX Web Companion*, Addison-Wesley, 1999.

[4] Leslie Lamport, *LaTeX: A Document Preparation System*, 2nd edition, Addison-Wesley, 1994.

[5] William F. Hammond, *The GELLMU Manual*, 2007, `http://mirror.ctan.org/support/gellmu/doc/glman.pdf`, or `http://mirror.ctan.org/support/gellmu/doc/glman.xhtml` (XHTML+MathML).

[6] William F. Hammond, "GELLMU: A Bridge for Authors from LaTeX to XML", *TUGboat: The Communications of the TeX Users Group*, vol. 22 (2001), pp. 204–207; also available online at `http://www.tug.org/TUGboat/Contents/contents22-3.html`.

[7] William F. Hammond, "Dual presentation with math from one source using GELLMU", *TUGboat: The Communications of the TeX Users Group*, vol. 28 (2007), pp. 306–311; also available online at `http://www.tug.org/TUGboat/Contents/contents28-3.html`. A video recording of the presentation at TUG 2007, July 2007, in San Diego is available at `http://www.river-valley.tv/conferences/tex/tug2007/`.

[8] William F. Hammond, "Multipurpose LaTeX-like markup for math", talk given in the AMS-MAA Special Session *Putting Math on the Web the Correct Way* at the Joint Mathematics Meetings in San Diego in January 2008. This has not been published, but HTML slides that link to many examples are available on the web at `http://math.albany.edu/math/pers/hammond/Presen/JMM08/Putting/`.

◇ William F. Hammond
Dept. of Mathematics & Statistics
University at Albany
Albany, New York 12222
USA
`hammond (at) albany dot edu`
`http://www.albany.edu/~hammond/`

[15] In keeping with the thought that one-character names should be reserved for user macros, the name for this character in the GELLMU didactic production system is `\csll`.

TUG 2010 abstracts

Editor's note: Many of the conference presentations are available at `http://www.river-valley.tv` in video form, thanks to Kaveh Bazargan and River Valley Technologies.

Kaveh Bazargan

Batch Commander: An interactive style writer for TEX

Batch Commander is a general graphic user interface for any batch system that runs a text file as a batch job and creates an output. It allows quick manipulation of parameters which it writes to an external config file and which it then uses to show the output. The latest incarnation of the system will be shown, with a live demo.

William Cheswick

Ebooks: New challenges for beautiful typesetting

TEX and other traditional text layout markup languages are predicated on the assumption that the final output format would be known to the nanometer. Extensive computation and clever algorithms let us optimize the presentation for a high standard of quality. But ebooks are here. The iPad has sold more than two million units in under three months, and, combined with other book readers, offers a new way to store and read documents. While these readers offer hope to newspapers (and perhaps doom to many physical bookstores), they are an increasing challenge to high quality text layout. Ebook users are accustomed to selecting text size (for aged eyes and varied reading conditions) and reader orientation. We can't run TEX over a document every time a reader shifts position. Do we precompute and download layouts for various devices, orientations, and text sizes? Do we compromise our standards of quality to use HTML- and XML-based solutions? These are new challenges to the TEX community.

Jean-luc Doumont

Quantum space: Designing pages on grids

Most (LA)TEX documents are vertical scrolls: essentially, they place content elements under each other, possibly running the scroll in two columns, but hardly more. With the exception of floats, they basically place items on the page in the order in which these are encountered in the source file: that is, they construct pages by piling up boxes horizontally and vertically, gluing them carefully together to achieve the desired (elastic) spacing.

Effective page design, in contrast, often benefits from a more global approach to the page or spread, one that replaces the scroll paradigm by a true two-dimensional layout. Pages are then usually constructed on an underlying grid, in reference to which the items can be positioned flexibly yet harmoniously. To produce all the documents created by our company (Principiae), I have developed such an approach in TEX. The session will present the ideas behind both grid designs in general and the corresponding TEX macros, and illustrate these ideas with a variety of examples (flyers, brochures, slides, etc.).

Our grid approach works in two steps: first create all the items that will appear on a page or spread (text blocks, illustrations, etc.), then place them in the desired locations on the grid, in any order. In a sense, the macro allow the user to specify, "this block of text goes there, that figure goes here, this title goes there, etc." — not unlike what page layout software allows, but with the infinitely superior accuracy that TEX allows. The macros I created to this end are simple, they have worked well for me for many years now, and the resulting documents very often surprise people ("This was done with TEX?!?"). The grid approach in TEX is best exemplified with my recent book (sample pages available at `http://www.treesmapsandtheorems.com`), in which grid alignments are pushed to an extreme, but it is behind all our documents, notably slides.

Steve Grathwohl and David Ruddy

Math on the Web: Implementing MathJax in Project Euclid

Project Euclid, a collaboration between Cornell University Library and Duke University Press, provides an online repository and publishing environment for independent mathematics and statistics journals. We discuss the issues surrounding the online display of mathematics at Project Euclid and, more specifically, the implementation of MathJax, an open-source, Ajax-based math display solution supporting both TEX and MathML notation.

Hans Hagen and Taco Hoekwater

How TEX and Meta finally got married

A story of TEX and METAFONT, 1996–2010. At first, information was passed between them via `\write18` and specials, e.g., for backgrounds and layers. Then LuaTEX started, and it became clear METAPOST should be a library, to avoid overhead. The MPlib project was funded by TEX user groups and others, with much code cleanup and backends for SVG and self-contained EPS. Ultimately, MP(lib) was married to LuaTEX. In the happy couple's future is arbitrary precision arithmetic, complete dynamic memory management, and more. As an ultimate test, Khaled Hosny created PunkNova, an OpenType font using the RAND feature. The text here uses that font.

Don Knuth 46 Don Knuth 47 Don Knuth 48 Don Knuth 49 Don Knuth 50 Don Knuth 51 Don Knuth 52 Don Knuth 53 Don Knuth 54 Don Knuth 55 Don Knuth 56 Don Knuth 57 Don Knuth 58 Don Knuth 59 Don Knuth 60 Don Knuth 61 Don Knuth 62 Don Knuth 63 . . . 4096 . . .

John Hobby

Is boxes.mp the right way to draw diagrams?

This talk explains the motivation behind `boxes.mp` and discusses some alternatives. Automatic graph layout can be combined with MetaPost in various ways, but this technology is somewhat hard to control.

Jonathan Kew

TeXworks for newcomers — and what's new for old hands

This presentation introduces TeXworks, a simple TeX environment based on modern standards — including Unicode text encoding, and PDF output by default — with an uncluttered interface that does not overwhelm the newcomer. It is built using cross-platform, open-source tools and libraries, so as to be available on all today's major operating systems, with a native "look and feel" for each.

First conceived during discussions at the time of TUG'07 in San Diego, TeXworks is now widely available, being included in both TeX Live and MiKTeX for Windows, MacTeX for Mac OS X, and in packages for various GNU/Linux, *BSD, and similar systems.

Following the first "stable" release (v0.2) in September 2009, the most significant new feature added to the application is a scripting interface. This allows users to extend and enhance the basic program in several ways, both by adding custom menu commands and by providing "hook" scripts that are automatically run at specific times, such as when a file is opened or after a typesetting run finishes. We will look at examples of how TeXworks can thus be extended using any of several available scripting languages.

The TeX community is invited to participate in the ongoing development of this environment, either at the level of actual code or in any supporting area, such as document templates or interface localization.

Frank Mittelbach

Exhuming coffins from the last century

In *The TeXbook* Don Knuth poses the following exercise: "Why do you think the author of TeX didn't make boxes more symmetrical between horizontal and vertical, by allowing reference points to be inside the boundary instead of insisting that the reference point must appear at the left edge of each box?" and gives the following answer: "No applications of such symmetrical boxes to English-language printing were apparent; it seemed pointless to carry extra generality as useless baggage that would rarely if ever be used, merely for the sake of symmetry. In other words, the author wore a computer science cap instead of a mathematician's mantle on the day that TeX's boxes were born. Time will tell whether or not this was a fundamental error!"

In this talk we will show how multiple reference points on boxes allow for a completely different approach to design specification and what can be done to successfully overcome the limitations resulting from Don's cap worn that day.

Ross Moore

TeX+MathML for Tagged PDF, the next frontier in mathematical typesetting

This talk will be a follow-on to the introduction to "Tagged PDF" given at last year's TeX Users Group meeting. Here I'll present several examples of tagged PDF documents containing real-world mathematical layouts, which demonstrate the advantages that tagging provides, in terms of long-term Archivability (PDF/A) and Accessibility (PDF/UA) and sharing of content and markup via export to XML.

A script, written in Perl, is under continuing development. This script combines the MathML presentational description of a piece of mathematics with corresponding LaTeX source for its visual appearance, creating a detailed TeX coding using new primitives that are processed by an enhanced version of pdfTeX to produce fully tagged PDF documents. If time permits we can discuss some of the complications that arise due to differences in the way mathematical structures are handled by TeX and for MathML.

This is joint work with Hàn Thế Thành (River Valley Technologies), author of pdfTeX.

Chris Rowley

A brief history of LaTeX — with a prediction

Not only brief, but *very* brief and with a lot of personal bias! History with attitude!! Left as unpredictable until the last minute will be both of these: What I mean by LaTeX; and of course the prediction!

Robert Rundell

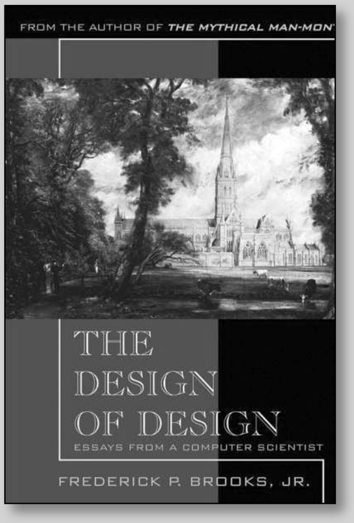
Using the Knuth-Plass algorithm to help control widow and orphan lines

The Knuth-Plass line-breaking algorithm is one of the many exceptional features of TeX, taking a paragraph of text and converting it to a vertical list of well-proportioned lines. Through glue and penalty markers TeX gives the user almost complete control over the spacing and look of the paragraph.

However, in some instances TeX does not provide the user quite as rich a set of options to control the vertical list as in other areas. In particular, eliminating widow and orphan lines can require inserting forced break points into the text, break points that can only be found from previous passes of TeX. Subsequent changes to the document can require changes to some or all of these manually inserted line or page breaks.

In AML, an experimental typesetting program under development, the Knuth-Plass algorithm is enhanced to find not only the optimal line-break points for a paragraph, but also to give alternate mappings of the paragraph into different numbers of lines (where possible). AML stores these different sets of break points and uses this information, along with natural page break points, to automatically eliminate widow and orphan lines in many cases. Once a bad page break point is detected, AML will backtrack and adjust previous paragraphs to create better page breaks. With far greater memory and processing capabilities than were available at TeX's creation, multiple pages can be examined and processed before a final page break needs to be finalized, allowing the overall document layout to be improved. The combination of keeping multiple pages and also keeping alternative paragraph line-breaking sets in memory allows AML to automate and improve this aspect of document typesetting.

NOW AVAILABLE from FRED BROOKS!


FROM THE AUTHOR OF THE MYTHICAL MAN-MONTH
THE DESIGN OF DESIGN
ESSAYS FROM A COMPUTER SCIENTIST
FREDERICK P. BROOKS, JR.

9780201362985 (Paperback)
Safari Books Online
Also available in all major eBook formats

These new essays by legendary author Fred Brooks contain extraordinary insights for designers in every discipline. Brooks pinpoints constants inherent in all design projects and uncovers process and patterns likely to lead to excellence. Throughout, Brooks reveals keys to success that every designer, design project manager, and design researcher should know.

For more information please visit informit.com/title/9780201362985

Addison-Wesley

Available wherever technical books are sold

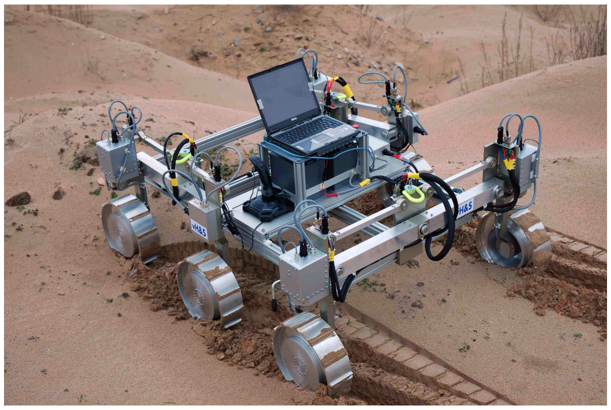
vH&S

vonHOERNER&SULGER

SPACE • RESEARCH • INDUSTRY

Happily using pdfTEX, LuaTEX, MetaPost, & tools in our space projects.

Executive summaries · Parts approval documents · Experiment user manuals · Risk management plans · MGSE user manuals · End-user statements · Parts count reliability predictions · Patent applications · Study reports · Progress reports · Failure mode, effects, and criticality analyses · Contract change notices · Acceptance data packages · Code listings · Requests for approval · Payload test specification input · Top level drawings · Electrical interfaces verification reports · PA/QA plans · Schedules · Mating records · User manuals · Thermal analyses · Thermal test reports · Age-sensitive item records · System engineering plans · EMC test reports · Metrology reports · Requirements documents · Document lists · Vibration test reports · Functional test reports · Transport, handling, and installation procedures · Fracture control plans · Approvals to ship · Final reports · Letters · Interface control documents · Software configuration status lists · Structural analyses · Declared materials lists · Declared components lists · Handouts · Compliance matrices · Lists of non-conformance reports · Declared processes lists · Lists of waivers · Physical properties reports · Test reports · Technical notes · Product logbooks · EGSE user manuals · Conceptual design reports · Design & development plans · Test & verification plans · Non-conformance reports · Certificates of conformance · Functional diagrams · Derating analyses · Instrument configuration lists · Open items lists · Minutes of meetings · Grounding & bonding diagrams · Connector mating records · Radiation control plans · Viewgraphs · Lists of engineering change requests · Quotations · Posters · Bench checkout procedures · Configuration drawings · Test matrices · Project plans · Shipping documents · Red-tag item tracking records · Qualification status lists · Christmas cards · Structural test reports · Worst case analyses · ITAR components lists · Calibration data records · Advertisements · Bills · Detailed design reports. . .


Mars rover breadboard for ESA's ExoMars mission 2018, built by vH&S with industry team (the flowers won't be there then).

von Hoerner & Sulger GmbH
Schlossplatz 8, D-68723 Schwetzingen, Germany
http://www.vh-s.de

Do you need on-site training for LaTeX?

Contact Cheryl Ponchin at

`cponchin@comcast.net`

Training will be customized for your company needs.

Any level, from Beginning to Advanced.

RIVER VALLEY

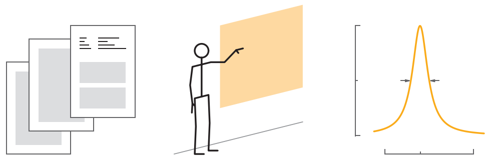
TECHNOLOGIES

“Awesome… groundbreaking… essential”

IEEE Trans. Prof. Commun.

Trees, maps, and theorems

Effective communication for rational minds



Get your copy at the discounted rate of $72 (instead of $96) by ordering within two weeks from site www.treesmapsandtheorems.com and entering “special order” code TUGXX004.

University Science Books

Publishers of Chemistry, Physics, Astronomy, Biology and Earth Environmental text and reference books.

Orders on their website, `http://www.uscibooks.com`, *receive a 15% discount.*

Calendar

2010

Aug 17 – 22	TypeCon 2010: "Babel", Los Angeles, California. `www.typecon.com`
Aug 31 – Sep 3	Book history workshop, École de l'institut d'histoire du livre, Lyon, France. `ihl.enssib.fr`
Sep 5 – Oct 25	"Marking Time": A traveling juried exhibition of books by members of the Guild of Book Workers. Lafayette College, Easton, Pennsylvania. Sites and dates are listed at `palimpsest.stanford.edu/byorg/gbw`
Sep 8 – 12	Association Typographique Internationale (ATypI) annual conference, "The Word", Dublin, Ireland. `www.atypi.org`
Sep 13 – 18	Fourth International ConTEXt User Meeting, "ConTEXt typesetting documentation, teach as we preach", Brejlov (Prague), Czech Republic. `meeting.contextgarden.net`
Sep 16 – 19	TEXperience 2010, CS TUG, Brejlov (Prague), Czech Republic. `striz9.fame.utb.cz/texperience`
Sep 24 – 26	DANTE Herbsttagung and 43rd meeting, Trier, Germany. `www.dante.de/events/mv43.html`
Oct 1 – 3	Oak Knoll Fest XVI, and Fine Press Book Association annual meeting, New Castle, Delaware. `www.oakknoll.com/fest`
Oct 15 – 16	American Printing History Association's 35th annual conference, "Learning to Print, Teaching to Print", Washington, DC. `www.printinghistory.org/programs/conference/conference_2010.php`
Nov 5 – Mar 20	"Marking Time": A traveling juried exhibition of books by members of the Guild of Book Workers. Dartmouth College, Hanover, New Hampshire. Sites and dates are listed at `palimpsest.stanford.edu/byorg/gbw`
Nov 6 – 8	The Eighth International Conference on the Book, University of St. Gallen, St. Gallen, Switzerland. `booksandpublishing.com/conference-2010`
Nov 19	"Letterpress: Forward thinking", St Bride Library, London, England. `stbride.org/events`

2011

Jan 13 – 16	College Book Art Association Biennial Conference, "Word, Image, Text, Object", University of Indiana, Bloomington, Indiana. `www.collegebookart.org`
Jan 28	"The Design of Understanding", St Bride Library, London, England. `stbride.org/events`
Feb 1	**TUG election:** nominations due. `tug.org/election`
Apr-May	BachoTEX 2011: 19th BachoTEX Conference, Bachotek, Poland. `www.gust.org.pl/bachotex`
Jul 14 – 17	SHARP 2011, "The Book in Art & Science", Society for the History of Authorship, Reading & Publishing. Washington, DC. `www.sharpweb.org`
Aug 7 – 11	SIGGRAPH 2011, Vancouver, Canada. `www.siggraph.org/s2011`
Oct 14 – 16	The Ninth International Conference on the Book, University of Toronto, Ontario, Canada. `booksandpublishing.com/conference-2011`

TUG 2011
Cairo, Egypt.

Nov 14 – 17	The 32nd annual meeting of the TEX Users Group. `tug.org/tug2011`

Status as of 20 August 2010

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

A combined calendar for all user groups is online at `texcalendar.dante.de`.

Other calendars of typographic interest are linked from `tug.org/calendar.html`.

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `http://tug.org/consultants.html`. If you'd like to be listed, please see that web page.

Dangerous Curve
PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
Web: `http://dangerouscurve.org/tex.html`
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. If you use XƎTEX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TEX book.

Hendrickson, Amy
Brookline, MA, USA
Email: `amyh (at) texnology.com`
Web: `http://www.texnology.com`
LATEX macro writing our speciality for more than 25 years: macro packages for major publishing companies, author support; journal macros for American Geophysical Union, Proceedings of the National Academy of Sciences, and many more.

Scientific journal design/production/hosting, e-publishing in PDF or html.

LATEX training, at MIT, Harvard, many more venues. Customized on site training available.

Please visit our site for samples, and get in touch. We are particularly glad to take on adventurous new uses for LATEX, for instance, web based report generation including graphics, for bioinformatics or other applications.

Martinez, Mercè Aicart
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, TEX and LATEX typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Peter, Steve
295 N Bridge St.
Somerville, NJ 08876
+1 732 306-6309
Email: `speter (at) mac.com`
Specializing in foreign language, linguistic, and technical typesetting using TEX, LATEX, and ConTEXt, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Shanmugam, R.
No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.
Choolaimedu, Chennai-600094, Tamilnadu, India
+91 9841061058
Email: `rshanmugam92 (at) yahoo.com`
As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices.
I provide consultation in building business models & technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in TEX, LATEX2$_\varepsilon$, XMLTEX, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sievers, Martin
Im Treff 8, 54296 Trier, Germany
+49 651 4936567-0
Email: `info (at) schoenerpublizieren.com`
Web: `http://www.schoenerpublizieren.com`
As a mathematician with ten years of typesetting experience I offer TEX and LATEX services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents.

From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (BIBTEX, biblatex) to typesetting your math, tables or graphics — just contact me with information on your project.

Veytsman, Boris
46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`
TEX and LATEX consulting, training and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions and much more. I have about fifteen years of experience in TEX and twenty-eight years of experience in teaching & training. I have authored several packages on CTAN, published papers in TEX related journals, and conducted several workshops on TEX and related subjects.

TUG Institutional Members

American Mathematical Society, *Providence*, *Rhode Island*

Aware Software, Inc., *Midland Park*, *New Jersey*

Banca d'Italia, *Roma*, *Italy*

Center for Computing Sciences, *Bowie*, *Maryland*

Certicom Corp., *Mississauga*, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

Florida State University, School of Computational Science and Information Technology, *Tallahassee*, *Florida*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

Konica Minolta Systems Lab Inc, *Boulder*, *Colorado*

MacKichan Software, Inc., *Washington/New Mexico*, *USA*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czech Republic*

MOSEK ApS, *Copenhagen*, *Denmark*

New York University, Academic Computing Facility, *New York*, *New York*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

Stanford University, Computer Science Department, *Stanford*, *California*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

University College, Cork, Computer Centre, *Cork*, *Ireland*

University of Delaware, Computing and Network Services, *Newark*, *Delaware*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

2011 TeX Users Group election

Jim Hefferon for the Elections Committee

The positions of TUG President and nine members of the Board of Directors will be open as of the 2011 Annual Meeting, which will be held in November 2011 in Cairo, Egypt.

The directors whose terms will expire in 2011: Barbara Beeton, Jon Breitenbucher, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, and Philip Taylor. Two additional director positions are currently unoccupied.

Continuing directors, with terms ending in 2013, are: Jonathan Fine, Steve Grathwohl, Jim Hefferon, Klaus Höppner, Steve Peter, and David Walden.

The election to choose the new President and Board members will be held in Spring of 2011. Nominations for these openings are now invited.

The Bylaws provide that "Any member may be nominated for election to the office of TUG President/to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures." The term of President is two years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office at least two weeks (14 days) prior to the mailing of ballots. (A candidate's membership dues for 2011 will be expected to be paid by the nomination deadline.) The term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via `http://tug.org/election`.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt of nomination forms and ballot information at the TUG office is **1 February 2011**. Forms may be submitted by FAX, or scanned and submitted by e-mail to `office@tug.org`.

Ballots will be mailed to all members within 30 days after the close of nominations. Marked ballots must be returned no more than six (6) weeks following the mailing; the exact dates will be noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by early June, and will be announced in a future issue of *TUGboat* as well as through various TeX-related electronic lists.

2011 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2011 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: ____________________

Signature: ____________________

Date: ____________________

for the position of (check one):

☐ **TUG President**

☐ **Member of the TUG Board of Directors**

for a term beginning with the 2011 Annual Meeting, **November 2011**

1. ____________________
(please print)

____________________ (signature) ____________ (date)

2. ____________________
(please print)

____________________ (signature) ____________ (date)

Return this nomination form to the TUG office (forms submitted by FAX or scanned and submitted by e-mail will be accepted). Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received in the TUG office no later than **1 February 2011**.[1] It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

☐ nomination form

☐ photograph

☐ biography/personal statement

TeX Users Group **FAX:** +1 206 203-3960
Nominations for 2011 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

[1] Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same form.

TUGBOAT Volume 31 (2010), No. 2