TUG 2010 — program and information

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	Johannes Küster, Typoma	*Math never seen*
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	William Hammond, SUNY Albany	*LATEX profiles as objects in the "category" of markup languages*
12:15 pm	Boris Veytsman, George Mason U.	*Are virtual fonts obsolete?*
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*
4:35 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	*A Special Announcement!*
≈6:00 pm	*end*	
7:30 pm	banquet	at *Le Colonial* (`lecolonialSF.com`)

Conference logistics

- **Conference location**: the Sir Francis Drake Hotel (`http://www.sirfrancisdrake.com`) in the Union Square area, San Francisco, California. The sessions will take place in the Franciscan Room on the mezzanine level, and breaks and lunches will take place in the same area. The workshops on the first day will be taught in the adjacent Windsor Room. Mezzanine level floor plan: `http://www.sirfrancisdrake.com/meet-wed-celebrate/floor-plans`.
- **Opening reception**: the Franciscan Room (mezzanine level), Sunday evening, 5–7 pm. Hors d'oeuvres and nonalcoholic beverages will be served. You can purchase drinks from the lobby bar to bring in to the Franciscan room if you wish.
- **Registration**: also the Franciscan Room, Sunday evening, 5–7 pm(during the opening reception), and again Monday morning from 8–9 am. Please check in at the registration table to pick up your name tag, conference booklet, and other items.
- **Internet access:** Complimentary wireless is available to all conference participants in guest rooms and public spaces. Be sure to ask for an access code upon check-in at the hotel. Wireless is also available in the Franciscan Room, however, access may be spotty depending on usage. A business center, open 24 hours a day, is located near the front desk.
- **Hotel location:** Maps, directions and parking information can be found on the hotel web site `http://www.sirfrancisdrake.com/sfdmapa/index.html`.
- **Public transportation:** If you're flying into SFO or OAK you might consider taking Bay Area Rapid Transit (BART) from the airport to the hotel. Trains run every 15–20 minutes. A comprehensive list of stations is at `http://www.bart.gov/stations`. The train stop closest to the hotel is Powell Street. If you're walking from the Powell Street station to the Sir Francis Drake Hotel, start out going northeast on Market Street toward Powell Street. Turn left onto Powell Street, travel 4–5 blocks. The hotel is located just past Union Square and just before Sutter Street. It's about a seven minute walk from the station. Alternatively, you can ride in a cable car up Powell Street to the hotel.

TUG members meeting

After the regular session on Tuesday, we will hold a TUG user group meeting for anyone interested. Several TUG board members will be present to report on TUG's current status and future outlook.

We invite discussion of any TUG-related business at this time: ideas for outreach to additional communities, additional initiatives to undertake, existing projects to support, or other topics.

Banquet & soapbox

The conference banquet will be held at Le Colonial restaurant (`http://www.lecolonialsf.com`) at 7 pm on Wednesday, June 30. If you haven't signed up for the banquet, it's not too late. Just let us know.

The restaurant is located about 3 blocks from the hotel, at 20 Cosmos Place, San Francisco, 94109. Dinner will be served family-style.

We will have a few door prizes at the banquet. In addition, we will hold a 32–128 second soapbox at the banquet, where anyone can speak for a minimum of 32 seconds and a maximum of 128 seconds:

- You can reminisce about Stanford, TEX, Knuth, or hold forth on something else: report a success, gripe about a problem, lament a failure, share an insight, ask a question, or explain a solution.
- No intros, no questions, no hacking on earlier speakers; just you, the mike, and the audience ...
- ... and a moderator with a timer who will cut you off when your time is up.
- No slides, overheads, whiteboards, blackboards, flipcharts, chalk, markers, or other props.
- Come prepared or make it up on the spur of the moment — no experience necessary.

Pavneet Arora
Using LaTeX to generate dynamic mathematics worksheets for the web

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 practise. Beginning with an exploration on effective learning strategies for grade school mathematics the use of LaTeX to generate dynamic mathematics worksheets — lots and lots of them — is discussed.

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.

Mathieu Bourgeois and Roger Villemaire
Introduction to drawing structured diagrams in SDDL

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.

In SDDL, one adds objects to a canvas in order to produce a drawing. Objects are either basic building blocks such as circles, lines, arrows or even already defined canvas. This allows reusing existing representations by integrating them at various positions in the main canvas. Furthermore, inner objects can always be referred to. It is hence easy to add linking objects, such as lines and arrows, between inner objects.

SDDL uses an object-oriented inspired syntax, using the dot to access attributes, such as specific points (center, corner, etc.), in a natural way. Diagrams are hence constructed by combining existing parts and linking them in various ways.

Our tool is implemented in Java, but, since SDDL offers its own simple syntax, no knowledge of Java is required in order to learn SDDL. The tool translates the SDDL input into Asymptote code and uses the Asymptote engine to produce EPS output.

SDDL is hence a simple and clear language in which one can combine graphical objects and LaTeX code in order to produce structured diagrams such as those used in discrete mathematics and computer science.

John Bowman
Interactive TeX-aware 3D vector graphics

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.

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.

Bart Childs

Thirty years of literate programming and more?

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*. There then appeared a number of "Literate Programming systems".

The use of the term Literate Programming is often applied to systems that have few of the characteristics of Knuth's `WEB`. There are at least two systems that are still in use that are quite faithful to the philosophy that Knuth elucidated in his original Pascal based `WEB` system: `CWEB` and `FWEB`. These support at least three languages each. Most other systems are relatively independent of language.

I will propose a definition for Literate Programming that will be used in my comments about some of these systems. I will also discuss some items from my archives (or memory) about this and related subjects. Some come from teaching the freshman year of computer science using literate programming.

I believe that this style of program development is a great contribution to the goal of creating excellent and maintainable programs. 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. I will offer my opinions as to why this tragedy persists, what I/we should have done — based on my humble view from my faulty crystal ball.

Michael Doob

A web-based TEX previewer: Ecstasy and agony

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 presentation includes a web-based viewer of the type just described. It will be used to:

- 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.

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.

Walter Gander

Writing the first LATEX book

In 1984 I wanted to write a German textbook called *Computermathematik* using the typesetting system TEX developed by Don Knuth, which I had always admired and which I had 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.

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
Just in time: Things we can do only with LuaTEX

All the time that I've been using TEX, I've been lucky enough to stumble into a solution just in time to save my day (or some project). In most cases it involved starting from scratch with the strong belief that TEX can do everything. After a while you reach a state where you can predict if something can be done or not.

An extreme example of operating on the edge is backgrounds that span paragraphs and pages, adapt to paragraph characteristics, and can be nested. Another mechanism that made some projects possible was HTML-like table building. Imagine combining these two mechanisms.

Such traditional solutions already benefit a lot from LuaTEX for instance because MetaPost is integrated. Although we could stretch TEX's lifespan in for instance the font arena it is no fun to stay eight bit among the Unicoders and OpenType lovers. But it can get worse. The last couple of years I started running into designers' demands that simply are not possible in traditional TEX. Especially in automated typesetting you need tricks that are (with good reason) beyond standard TEX engines. Here we need to really adapt or extend the engine to get things done.

In this talk I will show a few examples and solutions. These also show the MkIV approach to solve such nasty problems. I will discuss some experiments with providing Lua based variants of internal functions that we can use in exceptional situations where performance (in terms of speed) is not an issue.

Hans Hagen
Building paragraphs with the help of Lua

In the Oriental TEX project we use a combined approach to get nicely typeset paragraphs. We use a font with so many features that it drives font programs crazy but it works out well. We combine that with a special paragraph optimizer that improves the quality using different feature sets. This is a typical example of a local optimization that only kicks in on demand.

In this talk I will show how input is converted into nice looking output and how the already acceptable output can be further improved. I will also show how we visualize the process.

A byproduct of this effort is the TEX pararagraph builder rewritten in Lua. I will discuss a few issues that showed up when converting the original code into Lua and some of the outcomes that will be fed back into the LuaTEX code base.

Hans Hagen and Taco Hoekwater
How TEX and Meta finally got married

MetaPost 2.000 is planned for release in the summer of 2010. This presentation is a short report on the project history and current status. MetaPost version 1.500 was released around BachoTEX 2010, and in that release all memory arrays will have been replaced by dynamic memory allocation.

Idris Hamid
Oriental TEX: Culturally authentic typesetting of the Qur'an

After years of research, the Oriental TEX Project can proudly announce that it is closing in on the holy grail of paragraph-based Arabic typography. We illustrate this by demonstrating the typesetting of the Qur'an in LuaTEX and ConTEXt MkIV.

William Hammond
LATEX profiles as objects in the "category" of markup languages

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] William F. Hammond, "GELLMU: A Bridge for Authors from LATEX to XML", *TUGboat*, vol. 22 (2001), no. 3, pp. 204–207; also available online at `http://www.tug.org/TUGboat/Contents/contents22-3.html`.

[2] William F. Hammond, "Dual presentation with math from one source using GELLMU", *TUGboat*, vol. 28 (2007), no. 3, 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/.
[3] 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/.

Jim Hefferon

Characterizing CTAN packages

CTAN has many packages that solve many problems, but users can have trouble finding the package that solves the problem that they are having today. We now support text-based searches of the package descriptions. Here we will demonstrate keyword and tree-based characterizations of packages.

Stephen Hicks

Improving margin paragraphs and float control

Authors using LaTeX to typeset books with significant margin material often run into the problem of long notes running off the bottom of the page. A typical workaround is to insert vertical shifts by hand, but this is a tedious process that is invalidated when pagination changes. Another workaround is `memoir`'s sidebar function, but this can be unsatisfying for short textual notes, and standard marginpars cannot be mixed with sidebars. I will discuss a solution I put together to make marginpars "just work" by keeping a list of floating inserts and placing them intelligently in the output routine. Time permitting, I will also discuss some thoughts on improving LaTeX's float placement specifiers.

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.

Alan Hoenig

TeX helps you learn Chinese character meanings!

I've recently used XeTeX to typeset and maintain a manuscript which develops a mnemonic technique for remembering the meanings for the 2000 most common Chinese characters. Following a brief introduction to this method, I discuss how painless XeTeX makes it to typeset Chinese and English together, and how TeX makes it (relatively) simple to implement this memory method in a handbook such as this. Some concluding comments emphasize aspects that are familiar to old TeX-hands, but may be overlooked by newer users. Because TeX source is ASCII text (or its Unicode extension), it's easy to manage and maintain the information in these source files in a straightforward way via Perl or any other scripting language. TeX coding often becomes simpler, as it's possible for Perl to make some decisions (not typesetting ones, to be sure) for you, so your TeX macros have less work to do.

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.

Johannes Küster

Math never seen

Why have certain symbols and notations gained general acceptance, while others fell into oblivion? And why did mathematicians happily adopt TeX as a standard, while they hardly ever used METAFONT (or other tools) to develop new notations?

In this presentation I will give quality criteria for mathematical symbols. I will show many unknown, little-known or little-used notations, some of which deserve to be much more widely used.

Also I will show new symbols and ideas for new notations, especially for some well known notions which lack a good notation (e.g., gcd and lcm, Stirling numbers, and more).

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 TUG 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.

Will Robertson

Unicode mathematics in LATEX: Advantages and challenges

Over the last few years I've been tinkering with Unicode mathematics in XETEX. In late 2009 I spent a few weeks ironing out the significant bugs and think I've got a pretty good handle on the whole system now. In this presentation, I'll discuss the advantages Unicode maths brings to LATEX, challenges faced dealing with Unicode, challenges with maths fonts (including the STIX fonts), challenges with compatibility with `amsmath` and/or MathML, and assorted related remarks. In future plans, I hope to use this system as the basis for equivalent development in LuaTEX as well.

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.

Didier Verna

Classes, styles, conflicts: The biological realm of LATEX

Every LATEX user faces the "compatibility nightmare" one day. With so much intercession capabilities 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 some general principles or a systematic approach to handling class/style cross-compatibility in a smooth and gentle manner, but I ultimately failed. Instead, 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 small entities, themselves in perpetual evolution.

In this presentation, I would like to draw 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.

Boris Veytsman, Leila Akhmadeeva

TEX in the GLAMP world: On-demand creation of documents online

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 dynamic creation of Web pages [2,6]. In this talk we describe the use of this technology for automatic creation of medical pedigrees [1,3,4,5].

To make the drawing of pedigrees easy for medical professionals, we put TeX and PostScript processing of their input on a Web site (`http://pedigree.varphi.com`). In this talk we cover both technical aspects of this (integration of TeX with GLAMP) and the preliminary results of using this site in the education environment.

[1] Leila Akhmadeeva. Using a new package for drawing pedigrees for teaching medical genetics. *Eur. J. Hum. Gen.*, 15(Suppl. 1):338, 2007.

[2] 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`, February 2010.

[3] 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`.

[4] 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`.

[5] 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`.

[6] Wikipedia. LAMP (software bundle). `http://en.wikipedia.org/wiki/LAMP_(software_bundle)`, February 2010.

Boris Veytsman

Are virtual fonts obsolete?

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 [2,3]. Today new font formats (OTF, TTF, etc.) can be directly read by the modern TeX engines, and, for example, XeTeX can directly work with system fonts. There is a temptation to declare VF obsolete.

In this talk we show that there is much more functionality 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. Some of these tricks are described in Alan Hoenig's great book [1], and some are used by the author [4].

The aim of this presentation 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] Alan Hoenig. *TEX Unbound: LaTeX and TeX Strategies for Fonts, Graphics, and More*. Oxford University Press, USA, 1998.

[2] Alan Jeffrey, Rowland McDonnell, and Lars Hellström. Fontinst. *Font Installation Software for TEX*, December 2004. `http://mirrors.ctan.org/fonts/utilities/fontinst`.

[3] Philipp Lehman. *The Font Installation Guide*, December 2004. `http://mirrors.ctan.org/info/Type1fonts/fontinstallationguide`.

[4] Boris Veytsman. *LaTeX Support for Microsoft Georgia and ITC Franklin Gothic In Text and Math*, July 2009. `http://mirrors.ctan.org/fonts/mathgifg/`.

Herbert Voss

From PostScript to PDF

There are still several reasons to use the "traditional" way of creating PDF output, namely the sequence `latex` → `dvips` → `ps2pdf`. Using pdfLaTeX is only possible when the PostScript

related code is handled before the pdfLaTeX run. Thus, several packages and/or scripts have been developed which supports EPS images, or general PostScript-related code, in a document which is compiled at least one time with pdfLaTeX: `pst-pdf`, `auto-pst-pdf`, `pdftricks`, `epstopdf`, `pst2pdf`, `pstools`, ... All have the same general goal, but each works in a different way. We will demonstrate with several examples.

Uwe Ziegenhagen

Dynamic reporting with R/Sweave and LaTeX

R is a sophisticated statistical programming language available on various platforms. Since its initial development in 1992 it has become a major tool for many scientists all over the world. For the integration with LaTeX it provides various tools allowing a dynamic creation of reports. In my presentation I am going to present a hands-on demonstration of how to work with R and generate impressive reports using the packages Sweave, `xtable` and `tikzdevice`.

A web-based TEX previewer: the ecstasy and the agony

Michael Doob
Department of Mathematics
The University of Manitoba
Winnipeg, Manitoba, Canada R3N 2T2
mdoob@ccu.umanitoba.ca

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 presentation includes a web-based viewer of the type just described. It will be used to:

- 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 some 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) in 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 uses 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

Sometime it's desirable to try out a new definition that may take a few tries to get it 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 <texhax@tug.org>`, 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 replicate that experience using a reasonable web connection.

1.2 LAMP Implementation

1.2.1 Environment

Our environment used for this application is sometimes called LAMP: the Linux operating system, the Apache web server, the MySQL database management system (unused in this application) and PHP (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.

1.2.2 Desired Elements

The minimum implementation would allow 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. It's also easy to have only file uploads and to display (portions of) the log file.

Additionally, it's also possible to preload TeX input or specific packages. For example, it could be more convenient to have the material in the input window inserted between the lines:

```
\documentclass{article}
\begin{document}

\end{document}
```

Similarly, it's also easy to preload either document classes of packages using pulldown menus. Examples are given in the documentation.

1.2.3 Browser Peculiarities

Ideally simple 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 command, but these shortcomings can be overcome in a LAMP environment. It's always possible to generate unique names with each call to TeXto avoid the cache problem. It's also possible to use freely available software to generate graphics (`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 system calls available. Nonetheless, there are precautions that must be taken.

2.1 Need to know

Clearly, the more widespread the audience is for a web application, the less is the information that should be disclosed about the the operating environment. There are two options: control the access to the web pages or control the amount of information disclosed. In a LAMP environment this is 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.

On the other hand, the log file, even when there is only one line of 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
```

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

Denial of Service (DOS) attacks 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
   \thecnt\newpage \stepcounter{cnt}
   \ifnum \value{cnt}<10000
\repeat
```

This produces a 10,000 page document with one integer (actually two if you include the page number) on each page. 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. Now suppose that `\thecnt\newpage` is also omitted. Then no memory is used, and TEX will run indefinitely using up any cpu resources available. There are two solutions 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 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.

2.3 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 three possible approaches.

2.3.1 Chroot Jail

The `chroot` command is available on all UNIX implementations. All the software (binaries and libraries) needed for the application are put on 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 chroot jail. This makes the rest of the computer environment safe even if the application is broken.

2.3.2 Software isolation of the Operating System

It is now fairly easy to set up virtual computers within a UNIX environment. It's possible to take a snapshot of the original setup, and then refresh the installation regularly. This means that any damage can be instantly repaired.

2.3.3 Hardware isolation of the Operating System

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 enclose, RAM and storage for less than $200.

3 Documentation

Finally, we want the actual PHP code that implements the web-based previewer. This is included in the attached TEX file. Running the file through LATEX prints the documentation along with instructions for extracting the PHP code.

Writing the first LaTeX-Book

Walter Gander, ETH Zurich

June 10, 2010

Abstract

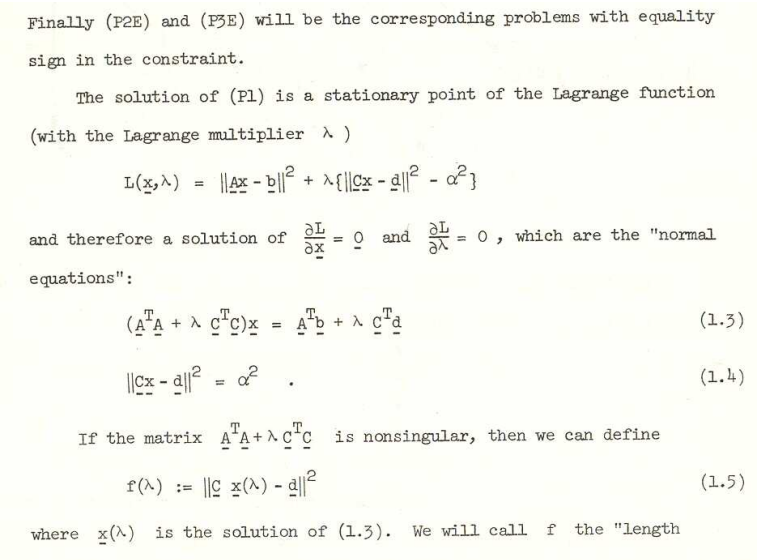
In 1984 I wanted to write a German text-book called "Computermathematik" using the typesetting system TeX developed by Don Knuth, which I have always admired and which I have been aware of since my first sabbatical year in Stanford in 1977. Mark Kent, a graduate student at Stanford in 1984, pointed out 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 postdoc writing my Habilitation. It was still the time where technical typists were writing 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 in 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.

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.



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 1977 in Stanford

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)

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 in 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 a lot 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

This amounts to two real equations to be satisfied.

Denote by $\Gamma_1, \ldots, \Gamma_m$ 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 $2m + 1, \ldots, N - 1$)

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

8

Figure 2: Part of Nick Trefethen's PhD Thesis

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 would 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.

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:

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:

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

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

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

Somit lautet für dieses Beispiel die Gleichung (4.1)

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


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

{\defi \it Seien $a_0, a_1, \ldots, a_n$ mit $a_n\ne 0$ gegebene
Zahlen. Es ist dann
\[
P_n(x) = a_0 + a_1 x +\cdots +a_nx^n
\]
ein Polynom vom Grade $n$. Die Zahlen $a_i$ heissen die Koeffizienten
von $P_n$.}
\section{Division durch einen Linearfaktor}
Oft stellt sich die Aufgabe, ein Polynom $P_n(x)$ durch den {\em
  ^{Linearfaktor} } $(x-z)$ zu dividieren. Man erh\"alt dabei ein
Polynom vom Grade $n-1$ und, falls die Division nicht aufgeht, eine
Zahl $r$ 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  & &    &         &
\\[-2\smallskipamount]
&\multicolumn{4}{c}\hrulefill&&&&\\
          & &       & & 9x  &+& 1  &         &                          \\
          & &       & &-9x  &+& 18 &         &
\\[-2\smallskipamount]
&&&\multicolumn{4}{c}\hrulefill&&\\
          & &       & &     & & 19 &  =r     &
    \end{array}
\end{equation}
Somit lautet f\"ur dieses Beispiel die Gleichung (\ref{44.2})
\[ \frac{3x^3+x^2-5x+1}{x-2}=3x^2+7x+9+\frac{19}{x-2}.\]
```

Figure 3: First page typeset

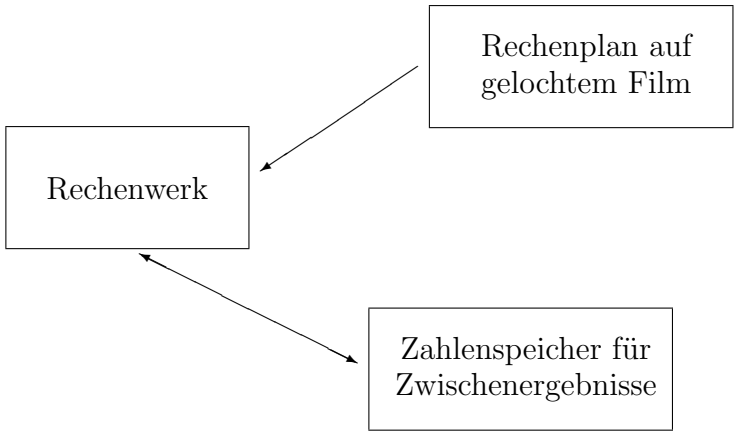


Figure 4: Erste Computer

```
\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 to much 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 was to replace a capitalized reserve word like `BEGIN` by `\BEGIN` where I had defined `\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:

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. In 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 in 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 in 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 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 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 includes 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 on a 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 LaTeXbook 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`.

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.

Grouping

Hans Hagen
PRAGMA ADE, Ridderstraat 27, 8061GH Hasselt NL
pragma@wxs.nl

Abstract

In this article I will discuss a few things that are hard to do in traditional TEX, but reasonable well in LuaTEX.

1 Variants

After using TEX for a while you get accustomed to one of its interesting concepts: grouping. Programming languages like Pascal and Modula have keywords `begin` and `end`. So, one can say:

```
if test then begin
    print_bold("test 1")
    print_bold("test 2")
end
```

Other languages provide a syntax like:

```
if test {
    print_bold("test 1")
    print_bold("test 2")
}
```

So, in those languages the `begin` and `end` and/or the curly braces define a 'group' of statements. In TEX on the other hand we have:

```
test \begingroup \bf test \endgroup test
```

Here the second `test` comes out in a bold font and the switch to bold (basically a different font is selected) is reverted after the group is closed. So, in TEX grouping deals with scope and not with grouping things together.
In other languages it depends on the language of locally defined variables are visible afterwards but in TEX they're really local unless a `\global` prefix (or one of the shortcuts) is used.
In languages like Lua we have constructs like:

```
for i=1,100 do
    local j = i + 20
    ...
end
```

Here `j` is visible after the loop ends unless prefixed by `local`. Yet another example is METAPOST:

```
begingroup ;
    save n ; numeric n ; n := 10 ;
    ...
endgroup ;
```

Here all variables are global unless they are explicitly saved inside a group. This makes perfect sense as the resulting graphic also has a global (accumulated) property. In practice one will rarely needs grouping, contrary to TEX where one really wants to keep changes local, if only because document content is so unpredictable that one never knows when some change in state happens.
In principle it is possible to carry over information across a group boundary. Consider this somewhat unrealistic example:

```
\begingroup
    \leftskip 10pt
    \begingroup
        ....
        \advance\leftskip 10pt
        ....
    \endgroup
\endgroup
```

How do we carry the advanced leftskip over the group boundary without using a global assignment which could have more drastic side effects? Here is the trick:

```
\begingroup
    \leftskip 10pt
    \begingroup
        ....
        \advance\leftskip 10pt
        ....
        \expandafter
```

```
    \endgroup
    \expandafter \leftskip \the\leftskip
\endgroup
```

This is typical the kind of code that gives new users the creeps but normally they never have to do that kind of coding. Also, that kind of tricks assumes that one knows how many groups are involved.

2 Implication

What does this all have to do with LuaTEX and MkIV? The user interface of ConTEXt provide lots of commands like:

```
\setupthis[style=bold]
\setupthat[color=green]
```

Most of them obey grouping. However, consider a situation where we use Lua code to deal with some aspect of typesetting, for instance numbering lines or adding ornamental elements to the text. In ConTEXt we flag such actions with attributes and often the real action takes place a bit later, for instance when a paragraph or page becomes available. A comparable pure TEX example is the following:

```
{test test \bf test \leftskip10pt test}
```

Here the switch to bold happens as expected but no leftskip of 10pt is applied. This is because the set value is already forgotten when the paragraph is actually typeset. So in fact we'd need:

```
{test test \bf test \leftskip10pt test \par}
```

Now, say that we have:

```
{test test test \setupflag[option=1] \flagnexttext{
test}
```

We flag some text (using an attribute) and expect it to get a treatment where option 1 is used. However, the real action might take place when TEX deals with the paragraph or page and by that time the specific option is already forgotten or it might have gotten another value. So, the rather natural TEX grouping does not work out that well in a hybrid situation.

As the user interface assumes a consistent behaviour we cannot simply make these settings global even if this makes much sense in practice. One solution is to carry the information with the flagged text i.e. associate it somehow in the attribute's value. Of course, as we never know in advance when this information is used, this might result in quite some states being stored persistently.

A side effect of this 'problem' is that new commands might get suboptimal user interfaces (especially inheritance or cloning of constructs) that are somewhat driven by these 'limitations'. Of course we may wonder if the end user will notice this.

To summarize this far, we have three sorts of grouping to deal with:

- TEX's normal grouping model limits its scope to the local situation and normally has only direct and local consequences. We cannot carry information over groups.
- Some of TEX's properties are applied later, for instance when a paragraph or page is typeset and in order to make 'local' changes effective, the user needs to add explicit paragraph ending commands (like `\par` or `\page`).
- Features dealt with asynchronously by Lua are at that time unaware of grouping and variables set that were active at the time the feature was triggered so there we need to make sure that our settings travel with the feature. There is not much that a user can do about it as this kind of management has to be done by the feature itself.

It is the third case that we will give an example of in the next section. We leave it up to the user if it gets noticed on the user interface.

3 An example

A group of commands that has been reimplemented using a hybrid solution is underlining or more generic: bars. Just take a look at the following examples and try to get an idea on how to deal with grouping. Keep in mind that:

- Colors are attributes and are resolved in the backend, so way after the paragraph has been typesetting.
- Overstrike is also handled by an attribute and gets applied in the backend as well, before colors are applied.
- Nested overstrikes might have different settings.
- An overstrike rule either inherits from the text or has its own color setting.

First an example where we inherit color from the text:

```
\definecolor[myblue][b=.75]
\definebar[myoverstrike][overstrike][color=]

Test \myoverstrike{%
    Test \myoverstrike{\myblue
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Because color is also implemented using attributes and processed later on we can access that information when we deal with the bar.

The following example has its own color setting:

```
\definecolor[myblue][b=.75]
\definecolor[myred] [r=.75]
\definebar[myoverstrike][overstrike][color=myred]

Test \myoverstrike{%
    Test \myoverstrike{\myblue
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

See how can we color the levels differently:

```
\definecolor[myblue] [b=.75]
\definecolor[myred]  [r=.75]
\definecolor[mygreen][g=.75]

\definebar[myoverstrike:1][overstrike][color=myblue]
\definebar[myoverstrike:2][overstrike][color=myred]
\definebar[myoverstrike:3][overstrike][color=mygreen]

Test \myoverstrike{%
    Test \myoverstrike{%
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Watch this:

```
\definecolor[myblue] [b=.75]
\definecolor[myred]  [r=.75]
\definecolor[mygreen][g=.75]

\definebar[myoverstrike][overstrike][max=1,dy=0,offset=.5
\definebar[myoverstrike:1][myoverstrike][color=myblue]
\definebar[myoverstrike:2][myoverstrike][color=myred]
\definebar[myoverstrike:3][myoverstrike][color=mygreen]

Test \myoverstrike{%
    Test \myoverstrike{%
        Test \myoverstrike{Test}
        Test}
    Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

It this the perfect user interface? Probably not, but at least it keeps the implementation quite simple.

The behaviour of the MkIV implementation is roughly the same as in MkII, although now we specify the dimensions and placement in terms of the ratio of the x-height of the current font.

```
Test \overstrike{Test \overstrike{Test \overstrike{Test}
Test} Test} Test \blank
Test \underbar  {Test \underbar  {Test \underbar
{Test} Test} Test} Test \blank
Test \overbar   {Test \overbar   {Test \overbar
{Test} Test} Test} Test \blank
Test \underbar  {Test \overbar   {Test \overstrike{Test}
Test} Test} Test \blank
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Test Test Test Test Test Test Test

Test Test Test Test Test Test Test

Test Test Test Test Test Test Test

As extra this mechanism can also provide simple backgrounds. The normal background mechanism uses METAPOST and the advantage is that we can use arbitrary shapes but it also carries some limitations. When the development of LuaTEX is a bit further along the road I will add the possibility to use METAPOST shapes in this mechanism.

Before we come to backgrounds, first take a look at these examples:

```
\startbar[underbar]  \input zapf \stopbar \blank
\startbar[underbars] \input zapf \stopbar \blank
```

Coming back to the use of typefaces in electronic

publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

First notice that it is no problem to span multiple lines and that hyphenation is not influenced at all. Second you can see that continuous rules are also possible. From such a continuous rule to a background is a small step:

```
\definebar
  [backbar]
  [offset=1.5,rulethickness=2.8,color=blue,
   continue=yes,order=background]

\definebar
  [forebar]
  [offset=1.5,rulethickness=2.8,color=blue,
   continue=yes,order=foreground]
```

The following example code looks messy but this has to do with the fact that we want properly spaced sample injection.

```
from here
    \startcolor[white]%
        \startbar[backbar]%
            \input zapf
            \removeunwantedspaces
        \stopbar
    \stopcolor
\space till here
\blank
from here
    \startbar[forebar]%
        \input zapf
        \removeunwantedspaces
    \stopbar
\space till here
```

from here Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on. till here

from here  till here

Watch how we can use the order to hide content. By default rules are drawn on top of the text.

Nice effects can be accomplished with transparencies:

```
\definecolor [tblue] [b=.5,t=.25,a=1]
\setupbars [backbar] [color=tblue]
\setupbars [forebar] [color=tblue]
```

We use as example:

```
from here {\white \backbar{test test}
    \backbar {nested nested} \backbar{also also}}
till here
from here {\white \backbar{test test
    \backbar {nested nested}          also also}}
till here
from here {\white \backbar{test test
```

Grouping

```
    \backbar {nested nested}          also also}}
till here
```

from here test test nested nested also also till here
from here test test nested nested also also till here
from here test test nested nested also also till here
The darker nested variant is just the result of two transparent bars on top of each other. We can limit stacking, for instance:

```
\setupbars[backbar][max=1]
\setupbars[forebar][max=1]
```

This gives
from here test test nested nested also also till here
from here test test nested nested also also till here
from here test test nested nested also also till here
There are currently some limitations that are mostly due to the fact that we use only one attribute for this feature and a change in value triggers another handling. So, we have no real nesting here.
The default commands are defined as follows:

```
\definebar[overstrike]  [method=0,dy= 0.4,offset=
0.5]
\definebar[underbar]    [method=1,dy=-0.4,offset=-0.8]
\definebar[overbar]     [method=1,dy= 0.4,offset=
1.8]

\definebar[overstrikes] [overstrike] [continue=yes]
\definebar[underbars]   [underbar]   [continue=yes]
\definebar[overbars]    [overbar]    [continue=yes]
```

As the implementation is rather non-intrusive you can use bars almost everywhere. You can underbar a whole document but equally well you can stick to fooling around with for instance formulas.

```
\definecolor [tred]   [r=.5,t=.25,a=1]
\definecolor [tgreen] [g=.5,t=.25,a=1]
\definecolor [tblue]  [b=.5,t=.25,a=1]

\definebar [mathred]   [backbar] [color=tred]
\definebar [mathgreen] [backbar] [color=tgreen]
\definebar [mathblue]  [backbar] [color=tblue]

\startformula
    \mathred{e} = \mathgreen{\white mc} ^ {\mathblue{\white
e}}
\stopformula
```

We get:

$e = mc^e$

We started this chapter with some words on grouping. In the examples you see no difference between adding bars and for instance applying color. However you need to keep in mind that this is only because behind the screens we keep the current settings along with the attribute. In practice this is only noticeable when you do lots of (local) changes to the settings. Take:

```
{test test test \setupbars[color=red] \underbar{test}
test}
```

This results in a local change in settings, which in turn will associate a new attribute to `\underbar`. So, in fact the following underbar becomes a different one that previous underbars. When the page is prepared, the unique attribute value will relate to those settings. Of course there are more mechanisms where such associations take place.

4 More to come

Is this all there is? No, as usual the underlying mechanisms can be used for other purposes as well. Take for instance inline notes:

```
According to the wikipedia this is the longest
English word:
pneumonoultramicroscopicsilicovolcanoconiosis~\shiftup
{other long
words are pseudopseudohypoparathyroidism and
flocci-nauci-nihili-pili-fication}. Of course
in languags like Dutch and
German we can make arbitrary long words by pasting
words together.
```

This will produce:
According to the wikipedia this is the longest English word: pneumonoultramicroscopicsilicovolcanoconiosis other long words are pseudopseudohypoparathyroidism and floccinaucinihilipilification. Of course in languags like Dutch and German we can make arbitrary long words by pasting words together.
I wonder when users really start using such features.

5 Summary

Although under the hood the MkIV bar commands are quite different from their MkII counterparts users probably won't notice much difference at first sight. However, the new implementation does not

Hans Hagen

interfere with the par builder and other mechanisms. Plus, it configurable and it offers more functionality. However, as it is processed rather delayed, side effects might occur that are not foreseen.

So, if you ever notice such unexpected side effects, you know where it might result from: what you asked for is processed much later and by then the circumstances might have changed. If you suspect that it relates to grouping there is a simple remedy: define a new bar command in the document preamble instead of changing properties mid-document. After all, you are supposed to separate rendering and content in the first place.

1 Building paragraphs

1.1 Introduction

You enter the den of the Lion when you start messing around with the par-builder. Actually, as TEX does a pretty good job on breaking paragrphs into lines I never really looked in the code that does it all. However, the Oriental TEX project kind of forced it upon me. In the chapter about font goodies an optimizer is described that works per line. This method is somewhat simular to expansion level one support in the sense that it acts independent of the par builder: 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 looks 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 par 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 to really 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 lead to the ‘replace words in lines by narrower of wider variants’ appeoach.

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

1.2 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 bit 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 are good reasons for this.

- 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 build 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 succesive 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 mans grid snapping although that is quite isolated and not interwoven.

- Because it uses a slightly different way to deal 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 trasnslated into C. Actually in modern TEX compiling involves a translation into C first but the intermediate form is not meant for human eyes. As the LuaTEX project started from that merged code, Taco and Hartmut already spend 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 but also 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 divert too much from the original model simply because it's well documented.

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

1.3 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 par builder and protrusion code, but expansion gives different results (actually 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 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 we feel confident about this approach it will be backported into the C code base. So eventually the C might become a bit more readable and efficient.

Intercharacter kerning is dealt with somewhat strange. When a kern of subtype

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

zero is seen, and when it's 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 aproach 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 par builder
3 only character expansion as part of the par builder (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 if that we really have to wonder what certain code does and if it's still needed.

1.4 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. The times are in seconds an between parentheses the speed relative to the regular parbuilder is mentioned.

	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)

[2] As I more and more 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.

For a regular paragraph the Lua variant (currently) is 5 times slower and about 6 times when we use the Lua hpacker, which is not that bad given that it's interpreted code and that each access to a field in a node involves a function call. Actually, we can make a dedicated hpacker as soem code can be omitted, The reason why the protruding is relative slow is that we have quite some protruding characters in the test text (many commas and potential hyphens) and therefore we have quite some lookups and calculations. In the C variant 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 that the builder is more compartimized. Also, future versions of LuaTEX will pass around paragrapgh related parameters differently so that will have impact on the code as well.

1.5 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
```

This results in:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the

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

chaff and separate the sheep from the goats.

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

1.6 Conclusion

The module started working quiet well around the time that Peter Gabriels "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 below. 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.

Unicode mathematics in LaTeX: advantages and challenges

Will Robertson
School of Mechanical Engineering
University of Adelaide, SA, Australia
will.robertson@latex-project.org

Abstract

Over the last few years I've been tinkering with unicode mathematics in XeTeX. Late-2009 I spent a few weeks ironing out the significant bugs and think I've got a pretty good handle on the whole system now. In this presentation, I'd like to discuss the advantages unicode maths brings to LaTeX, challenges faced dealing with unicode, challenges with maths fonts (including the STIX fonts), challenges with compatibility with amsmath and/or MathML, and assorted related remarks. In future plans, I hope to use this system as the basis for equivalent development in LuaTeX as well.

1 Introduction

XeTeX was the first widely-used unicode extension to TeX. Several years ago Jonathan Kew [4] added OpenType maths support to XeTeX following from Microsoft's addition to the OpenType spec for Microsoft Word 2007. Around that time I built a prototype implementation of a unicode maths engine for LaTeX, but with very few OpenType maths fonts available, and other projects consuming my time, the project lost momentum and I never managed to get the package on CTAN.

Although the font situation has improved little since then, the STIX fonts have at least been shown in beta form and can be used to some degree. By the time you are reading this, an experimental version of the unicode-math package will hopefully have been released on CTAN. This extended abstract is a brief explanation of what it does and what it's good for.

2 What is unicode maths?

Barbara Beeton [1] has said it all much better than I could. In short: every known symbol to be used in technical writing has been included in the unicode specification. We now have a formal description of (some many thousand) exactly what glyphs a maths font should contain and the particulars of how they should behave [2]. (Contrast with maths fonts used with TeX documents that all contain a different set of glyphs.)

Since then, Microsoft has extended the OpenType specification to include tables of structured information for mathematics typesetting, essentially parameterising Knuth's original algorithms in TeX. For more context in this area, the historical development of unicode maths was summarised well by Ulrik Vieth [6].

Any unicode variant of TeX could always access the unicode maths glyphs. But typesetting them well is only possible with explicit layout intructions based on the Math OpenType specification.

2.1 So what does the package do?

After loading the package, users can write

```
\setmathfont{Cambria Math}
```

to select Cambria Math or any other OpenType font that has mathematics support. This can happen at any time during the document, not just in the preamble as with current LaTeX math font selection.

Control sequences are provided to access every defined unicode maths glyph, 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 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.

3 Advantages

The main advantage to having a unicode maths engine is that it becomes as easy to change maths fonts as it is to change text fonts. Contrast this with what packages such as euler must do to switch from the Computer Modern Maths font. Standardising around a common math font encoding was the original goal of the Math Font Group when they invented an 8-bit math font encoding but then realised that

unicode was the future. Unicode maths brings more benefits than simply standardising the way maths fonts are loaded, however.

3.1 Pragmatic

It's never fun when typesetting mathematics to have to go hunting for a specific symbol. (Scott Pakin's 'Symbols' guide makes the task far easier, however.) Which maths font glyph package to load? Which control sequence to use? Does the glyph design even match my document fonts?

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 — hopefully released by the time you are reading this — will be extremely useful as a fallback font for many symbols. After all, most maths symbols are geometrically abstract enough that they do not need to be directly matched with the text font.

3.2 Readable source

Unicode maths provides the ability for maths symbols and characters to be input in unicode directly in the source file. It is now possible to write a literal α instead of having to type longhand `\alpha`. This actually doesn't improve input speed, but makes source documents far more readable and amenable to casual editing.

Only small changes to the regular LaTeX syntax are required to approach the simplicity of Murray Sargent's 'nearly plain-text encoding of mathematics' [5].

3.3 Flexible output

Because there does not have to be a one-to-one mapping between the unicode characters in the source and the unicode glyphs in the output, the semantic differences between upright and italic Greek (and other) letters in the input source can be abstracted away. As shown in Table 1, documents are able to be typeset as per ISO standards or in a more classical TeX-like format without changing the typed 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 of characters. Figure 1 shows an example in which the maths was typed without presentational markup, but the different characters were assigned fonts with different shades of grey. This particular example may not be very practical, but it illustrates that the system is flexible enough to accomodate a wide range of effects.

	Example	
Package option	Latin	Greek
`math-style=ISO`	(a, z, B, X)	$(\alpha, \beta, \Gamma, \Xi)$
`math-style=TeX`	(a, z, B, X)	$(\alpha, \beta, \Gamma, \Xi)$
`math-style=french`	$(a, z, \mathrm{B}, \mathrm{X})$	$(\alpha, \beta, \Gamma, \Xi)$
`math-style=upright`	$(\mathrm{a}, \mathrm{z}, \mathrm{B}, \mathrm{X})$	$(\alpha, \beta, \Gamma, \Xi)$

Table 1: Same source, different styles of output.

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^\infty \mathrm{e}^{-st} f(t)\,\mathrm{d}t$$

Figure 1: Hooks make it possible to use a variety of fonts or styles — in this case, colours — for different maths characters or families of maths characters.

4 Challenges

The biggest problem I can see with the advent of unicode maths, besides fonts — in their case I believe they'll slowly start to appear now there are tools and programs to support them — is educating people into using them well.

4.1 Using the correct characters

Table 2 shows five different maths glyphs that are all triangular. And Table 3 shows the eight different slash-like glyphs; four in each direction.

Without careful documentation and good education, it may be hard for users to use the 'correct' glyphs 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, different authors may choose different (but visually similar) glyphs for the same meaning.

Slot	Command	Glyph	Class
`U+25B5`	`\vartriangle`	$\vartriangle$	binary
`U+25B3`	`\bigtriangleup`	$\bigtriangleup$	binary
`U+25B3`	`\triangle`	$\triangle$	ordinary
`U+2206`	`\increment`	$\increment$	ordinary
`U+0394`	`\mathup\Delta`	$\mathrm{\Delta}$	ordinary

Table 2: Four triangular glyphs with different meanings but similar shapes.

Slot	Name	Glyph	Command
U+002F	SOLIDUS	/	\slash
U+2044	FRACTION SLASH	⁄	\fracslash
U+2215	DIVISION SLASH	∕	\divslash
U+29F8	BIG SOLIDUS	⧸	\xsol
U+005C	REVERSE SOLIDUS	\	\backslash
U+2216	SET MINUS	∖	\smallsetminus
U+29F5	REVERSE SOLIDUS OPERATOR	⧵	\setminus
U+29F9	BIG REVERSE SOLIDUS	⧹	\xbsol

Table 3: The four slash-like glyphs in each direction.

My feelings are that new tools will be needed to encode mathematics more semantically. But such tools will need to be specific for each scientific field that uses different notation. This is an open problem. On the other hand, in the end how much does it really matter if my triangle is a different size to yours if the meaning is clear? (I'm not endorsing this line of thinking, just raising the question.)

4.2 Backwards compatibility vs. future compatibility

The two 'set minus' characters in Table 3 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: U+2216 is referred to by `setminus` *as well as* `smallsetminus` (among other synonyms); U+29F5 is as-yet unnamed [3]. This might make it difficult to move between MathML and LaTeX. Luckily these sorts of conflicts are few.

5 Summary

This extended abstract is an introduction and short summary of what I'll be talking about at TUG 2010. I'm looking forward to seeing you here.

Bibliography

[1] Barbara Beeton. Unicode and math, a combination whose time has come – Finally! *TUGboat*, 21(3):176–185, September 2000. URL http://tug.org/TUGboat/Articles/tb21-3/tb68beet.pdf.

[2] Barbara Beeton, Asmus Freytag, and Murray Sargent, III. Unicode support for mathematics. Unicode Technical Note 25 Version 9, Unicode, Inc., 2008. URL http://www.unicode.org/reports/tr25.

[3] David Carlisle and Patrick Ion. XML entity definitions for characters. Technical Report W3C Working Draft 21, W3C, 2008. URL http://www.w3.org/TR/xml-entity-names/.

[4] Jonathan Kew. XeTeX live. *TUGboat*, 29 (1):146–150, 2007. URL http://tug.org/TUGboat/Articles/tb29-1/tb91kew.pdf.

[5] Murray Sargent, III. Unicode nearly plaintext encoding of mathematics. Unicode technical note 28, Unicode, Inc., 2006. URL http://www.unicode.org/notes/tn28/.

[6] Ulrik Vieth. Do we need a 'cork' math font encoding? *TUGboat*, 29(3):426–434, 2008. URL http://tug.org/TUGboat/Articles/tb29-3/tb93vieth.pdf.

From PostScript to PDF

Herbert Voss

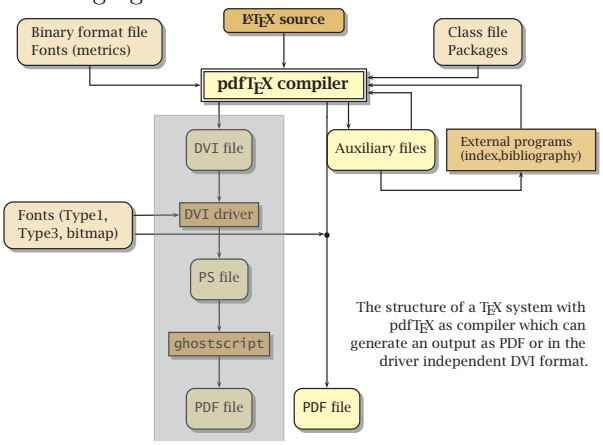
◇ Herbert Voss
Wasgenstrasse 21
Berlin, 14129
Germany
hvoss@tug.org

Abstract

There are still several reasons to use the "traditional" way of creating PDF output, namely the sequence `latex` → `dvips` → `ps2pdf`. Using pdfLaTeX is only possible when the PostScript related code is handled before the pdfLaTeX run. Thus, several packages and/or scripts have been developed which supports EPS images, or general PostScript-related code, in a document which is compiled at least one time with pdfLaTeX: `pst-pdf`, `auto-pst-pdf`, `pdftricks`, `epstopdf`, `pst2pdf`, `pstool`, ... All have the same general goal, but each works in a different way.

1 Introduction

The traditional way for running TeX is shown in the following figure.



The structure of a TeX system with pdfTeX as compiler which can generate an output as PDF or in the driver independent DVI format.

2 Demonstrations

We will demonstrate with several examples how `eps` images or any other PostScript related code can be used within a `pdflatex` run::

- Using the program `ps2pdf` to generate a `pdf` document
- Using `eps` images with `pdflatex` with Package `epstopdf`
- Using zipped `eps` images with `pdflatex`
- Using other image types with `pdflatex`
- Using `gif` and `png` images with `latex`
- Using `eps` images with `pdflatex` with Package `pstool`
- Using PSTricks with `pdflatex` and package `pdftricks`
- Using PSTricks with `pdflatex` and package `pst-pdf`
- Using PSTricks with `pdflatex` and package `auto-pst-pdf`
- Using PSTricks with `pdflatex` and the program `pst2pdf`