TUGboat

Volume 42, Number 1 / 2021

General Delivery

3 From the president / *Boris Veytsman*

4 Editorial comments / *Barbara Beeton*
R.I.P. Bob Morris; BaKoMa author has died;
J. W. Gibbs and why an annual lecture is named for him;
Jubilees to celebrate: *Computers & Typesetting* and 50 years of the ebook;
CTAN mirrors are now `https`; A self-published LaTeX book and ebook

6 Hendrik Vervliet: 1923–2020 / *Jacques André*

7 The TeX tuneup of 2021 / *Donald Knuth*

10 TeX entomology in 2021 / *Karl Berry*

11 Hyphenation exception log / *Barbara Beeton*

12 Year 2020 at GUTenberg / *Jérémy Just*

Dreamboat

13 Lapses in TeX — a look backward / *Barbara Beeton*

Typography

18 Typographers' Inn / *Peter Flynn*

20 Interview with Amelia Hugill-Fontanel / *David Walden*

LaTeX

34 The DuckBoat — Beginners' Pond: Crazy Little Thing Called Glue / *Carla Maggi*

40 Creating document commands: The good, the bad and the ugly / *Joseph Wright*

41 `\NewDocumentCommand` *versus* `\newcommand` *versus* ... / *Joseph Wright*

Surveys

44 Comparison of OpTeX with other formats: LaTeX and ConTeXt / *Petr Olšák*

Fonts

50 GUST e-foundry font projects, closing report 2019–2020 / *Jerzy Ludwichowski*

52 The NewComputerModern font family / *Antonis Tsolomitis*

56 An attempt at creating font transitions / *S.K. Venkatesan*

60 Scaled fonts and glyphs / *Hans Hagen*

65 Some fonts with recent TeX support / *Karl Berry*

Macros

66 Improvements to the generalized mediation macros in *The METAFONTbook* / *Hu Yajie*

Graphics

67 Animating Fourier series decomposition of a character with LuaTeX and MPLIB / *Maxime Chupin*

Software & Tools

72 Working remotely from an island: `arara` and other tools / *Island of TeX*

74 LuaMetaTeX programming features / *Hans Hagen*

81 UTF-8 installations of CWEB / *Igor Liferenko*

Reviews

83 Book review: *Learning LaTeX*, Second Edition, by David F. Griffiths and Desmond J. Higham / *Boris Veytsman*

Abstracts

84 *Die TeXnische Komödie*: Contents of issues 4/2020–1/2021

85 *Zpravodaj*: Contents of issue 2020/3–4

Cartoon

85 Comic: font or dog breed? / *John Atkinson*

Hints & Tricks

86 The treasure chest / *Karl Berry*

TUG Business

2 *TUGboat* editorial information

2 TUG institutional members

88 TUG 2021 election report

92 Spending MacTeX funds / *Richard Koch*

93 TUG financial statements for 2020 / *Karl Berry*

Advertisements

94 TeX consulting and production services

96 Calendar

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group. Web: `tug.org/TUGboat`.

Individual memberships
2021 dues for individual members are as follows:
- Trial rate for new members: $30.
- Regular members: $105.
- Special rate: $75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Members may also choose to receive *TUGboat* and other benefits electronically, at a discount. All membership options are described at `tug.org/join.html`.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership carries with it such rights and responsibilities as voting in TUG elections. All the details are on the TUG web site.

Journal subscriptions
TUGboat subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2021 is $115.

Institutional memberships
Institutional membership is primarily a means of showing continuing interest in and support for TEX and TUG. It also provides a discounted membership rate, site-wide electronic access, and other benefits. For further information, see `tug.org/instmem.html` or contact the TUG office.

Trademarks
Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is.

[printing date: April 2021]
Printed in U.S.A.

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

Telephone
+1 503 223-9994

Fax
+1 815 301-3568

Web
`tug.org`
`tug.org/TUGboat`

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

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

Volunteer technical support:
`support@tug.org`

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

dear barbara
i awoke this morning with the realization that TeX was created for on-lion publishing
don
[maybe i should go back to sleep]

Don Knuth
email, 5 Apr 2002

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
Editor Barbara Beeton

Volume 42, Number 1, 2021
Portland, Oregon, U.S.A.

TUGboat editorial information

This regular issue (Vol. 42, No. 1) is the first issue of the 2021 volume year. The deadline for the second issue in Vol. 42 (the TUG'21 conference proceedings) is August 15, 2021, and for the third issue is October 15, 2021. Contributions are requested.

TUGboat is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (`tug.org/store`), and online at the *TUGboat* web site (`tug.org/TUGboat`). Online publication to non-members is delayed for one issue, to give members the benefit of early access.

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

TUGboat advertising

For advertising rates and information, including consultant listings, contact the TUG office, or see:
`tug.org/TUGboat/advertising.html`
`tug.org/consultants.html`

Submitting items for publication

Proposals and requests for *TUGboat* articles are gratefully received. Please submit contributions by electronic mail to `TUGboat@tug.org`.

The *TUGboat* style files, for use with `plain` TeX and LaTeX, are available from CTAN and the *TUGboat* web site, and are included in common TeX distributions. We also accept submissions using ConTeXt. Deadlines, templates, tips for authors, and more, are available at `tug.org/TUGboat`.

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. Thus, the physical address you provide in the manuscript will also be available online. If you have any reservations about posting online, please notify the editors at the time of submission and we will be happy to make suitable arrangements.

Other TUG publications

TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the TeX community in general.

If you have such items or know of any that you would like considered for publication, please contact the Publications Committee at `tug-pub@tug.org`.

TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:
`tug.org/instmem.html`
Thanks to all for their support!

Adobe Inc., *San Jose*, *California*

American Mathematical Society, *Providence*, *Rhode Island*

Association for Computing Machinery, *New York*, *New York*

Aware Software, *Newark*, *Delaware*

Center for Computing Sciences, *Bowie*, *Maryland*

CSTUG, *Praha*, *Czech Republic*

Duke University Press, *Durham*, *North Carolina*

Harris Space and Intelligence Systems, *Melbourne*, *Florida*

Hindawi Foundation, *London*, *UK*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

Maluhy & Co., *São Paulo*, *Brazil*

Marquette University, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czech Republic*

Nagwa Limited, *Windsor*, *UK*

Overleaf, *London*, *UK*

StackExchange, *New York City*, *New York*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

TeXFolio, *Trivandrum*, *India*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

VTeX UAB, *Vilnius*, *Lithuania*

From the president

Boris Veytsman

Large organizations employ full-time staffers dealing with social media. The TEX Users Group is tiny, so this work is done by volunteers in their free time. In particular, one of my duties as TUG president is the support of our Twitter account. (By the way, if you do not follow `@TeXUsersGroup`, you may want to.) A significant number of tweets are announcements from CTAN: I forward the information about new TEX packages and the updates for existing ones. While it is easy enough to create an automatic Twitter gateway to post CTAN announcements, I prefer to do this manually; I enjoy starting my day with reading and sharing the news about the community. CTAN announcements are succinct and to the point — ideal Twitter content.

The information about contributions includes their licenses. Recently while posting it, I was startled to see that the license for a new package bans its commercial use, and decided to investigate. Gerd Neugebauer helped me to collect the information presented in Table 1. The LPPL seems to be the most popular license: 56% of packages use it. While the number of packages with "No commercial use" in the license is small (only 2.2%), the appearance of new packages under this is, in my opinion, troubling.

The free software community has a commitment to lofty ideals, and someone might consider the commercial use of our work "debasing" it. However, this would be a wrong conclusion. Many important and noble things are done by commercial entities with a profit motive. I am writing this in the middle of a pandemic, and our hope to overcome it is based on the vaccines, developed in record time by commercial companies with the clear intention to make a profit in the process of saving people. If you are a software author, do you really want to prevent your work from being used for vaccine development? Speaking about the TEX community, our software is widely used by publishers around the world. Do we consider the work of Johannes Gutenberg or Aldus Manutius less worthy because it was done with the need to make a living? On the other hand, there are, unfortunately, many examples of quite bad things done with motives other than direct profit.

Creative Commons guidelines helpfully explain that the most common non-commercial license, CC-NC, prohibits only the use "primarily intended for or directed toward commercial advantage or monetary compensation", and a commercial entity might still use software having an NC license if its primary intention is different. However, the cost of possible litigation makes the use of this software rather risky for a commercial entity in all cases. The even more helpful explanation that a non-profit organization can be in violation if its intentions are wrong makes it risky for non-commercial entities as well.

Table 1: CTAN package licenses (March 19, 2021)

License	Packages
LPPL, all versions	3448
GPL, all versions	748
Public domain or CC0	300
Open Font licenses (SIL, GFL, GFSL)	157
MIT license	125
BSD licenses, all versions	76
Knuth license	55
Free Documentation License	23
LGPL, all versions	22
Apache license	20
Perl Artistic license, all versions	12
CC BY	12
CC BY-SA, all versions	17
ISC license	2
Open Publication license	1
Other free licenses	353
'Collection' or 'Digest' license tag	24
No commercial use, all licenses	139
License that prevents distribution	73
No source available	29
Shareware	24
Unknown status	553

Our flagship distribution, TEX Live, does not include non-commercial software. As a non-profit, we probably have the right to distribute it (but see above). However, we want to create a TEX distribution that anybody can just use, without evaluating thousands of licenses for the included packages. Moreover, we want anybody to be able to redistribute it, for example, by selling computers with TEX Live among the preloaded software packages. Thus by using a restrictive license, you prevent your package from being included in TEX Live and significantly decrease the number of its users.

To tell the truth, I see only one use case for a non-commercial license: if you plan to separately license your software to commercial entities and charge them. However, this seems to be a rather rare situation for the TEX world.

I think that from an ethical and practical standpoint it makes most sense to follow Knuth's example and distribute your software under a non-restrictive license.

◇ Boris Veytsman
`president (at) tug dot org`

doi.org/10.47397/tb/42-1/tb130pres

Editorial comments

Barbara Beeton

R.I.P. Bob Morris

Bob Morris was one of the crew sent to Stanford by the AMS in July 1979 for the purpose of learning TEX, developing the AMS-TEX macros, and from there, communicating this new tool to the mathematical community as the mechanism for producing the publications in which they broadcast their research. This included the birth of the TEX Users Group in 1980 and active participation in the governance of TUG for several years.

Bob passed away on February 6, after a life filled with many adventures in digital typography, including leading research in this discipline at the University of Massachusetts, Boston. An interview with Dave Walden on the TUG web site[1] provides details of these activities as well as his earlier background.

He is remembered by those who knew him as a source of interesting ideas and a strong determination that something started should be investigated thoroughly.

His daughter Rachel remembers him thus,

> In life, my dad enjoyed taking things apart to discover how to put them back together better, whether physical or metaphysical. He was a philanthropist, a world traveler, an avid reader of non-fiction and science fiction, enjoyed a wide array of music, and dabbled in hobbies through the years, such as photography, ceramics, playing Go, and ham radio (which he gave up once Morse code was, to his disappointment, no longer required). He was proud that his Erdős number is three.

BaKoMa author has died

It was reported that Basil K. Malyshev passed away in 2019, but nothing more is known. Malyshev was best known in the TEX world for creating an extension of TEX that incorporated a WYSIWYG GUI, emulating MS Word.

A Russian physicist, Malyshev developed a wide array of TEX-related software, including a font converter to translate Metafont output to Type 1. All his software was released as shareware. The BaKoMa web site[2] remains intact, as do the packages posted to CTAN, and more information about the software and fonts can be found there.

[1] `tug.org/interviews/morris.html`

[2] `www.bakoma-tex.com`

J. W. Gibbs and why an annual lecture is named for him

J(osiah) Willard Gibbs was a professor of mathematical physics at Yale University in the late 19th century. Yale awarded Gibbs the first American doctorate in engineering in 1863. In 1866, he traveled to Europe, where he attended lectures in mathematics and physics. After several years, he returned to Yale, and spent the rest of his career there. He developed theories affecting physics, chemistry, mathematics and, perhaps best remembered today, applications of thermodynamics.

He submitted manuscripts reporting his work to a “local” journal, with this result.[3]

> Back in 1873, Gibbs had little idea of the epic consequences of his papers. Modest and unassuming, he sent his work to the little-known *Transactions of the Connecticut Academy of Arts and Science*, which had no readership outside Yale. Moreover, because Gibbs's papers were longer than the articles usually published by the *Transactions* and because they contained mathematical formulas, their typesetting costs exceeded the publication's budget. To cover these, the editorial committee had to obtain donations from other faculty members and local businessmen. One committee member, A. E. Verrill, later recalled that they had long discussions about the merits of Gibbs's papers even though no one on the committee understood them. “Yet we all believed what Gibbs wrote must be of intrinsic value in his branch of science. Therefore, we raised the money and printed each paper as it came in.”

(What are the chances that an editorial committee today would have such faith in the value of a first manuscript, even one by an author at their own institution?)

Gibbs's work was appreciated in both the U.S. and Europe, and in 1901 he was awarded the Copley Medal, the oldest and most prestigious award of the Royal Society of London. (Other recipients of this medal include Benjamin Franklin, Charles Darwin, and Albert Einstein.[4])

In 1923, the Council of the American Mathematical Society established the Josiah Willard Gibbs

[3] Paul Sen, *Einstein's Fridge: How the Difference between Hot and Cold Explains the Universe*, Scribner: An imprint of Simon & Schuster, Inc., New York, 2021. Quoted with permission.

[4] The list of recipients is stunning: `britannica.com/science/Copley-Medal`

Lectureship, “in order to show the public some idea of the aspects of mathematics and its applications.”[5]

Don Knuth was the Gibbs Lecturer in 1978, when he lectured on “Computers in the service of mathematics”. An important point made in that lecture was the characterization of mathematics composition as “penalty copy”, on account of the difficulty, and for that reason, the high cost. Thus began the public journey of TEX.

Jubilees to celebrate: *Computers & Typesetting* and 50 years of the ebook

The end of December 2020 marked the delivery of the accumulated bug reports to Don Knuth for the periodic tuneup of the TEX/METAFONT complex. Don’s report appears later in this issue.

In addition to the release of the updated software, included in TEX Live 2021, this update has resulted in the preparation of a “35th Jubilee Edition” of all five books comprising *Computers & Typesetting*. The new editions haven’t yet been announced on the usual web sites, but stay tuned.

Another jubilee was announced on the Project Gutenberg mail list: 50 years of electronic books. An illustrated history can be viewed at:
`geekupdated.com/50-years-of-ebooks-illustrated-history/`

CTAN mirrors are now `https`

As of approximately 20 April 2021, the CTAN feature that automatically chooses a nearby mirror is now available through `https`, as well as `http`. The new address is `https://mirror.ctan.org`. Previously, redirection was available only through `http://mirror.ctan.org` (which still works).

Whether the multiplexer is accessed through `https` or `http`, the mirror chosen will always be accessed through `https`. If you are a CTAN mirror administrator (current or prospective), please ensure that your mirror is available via `https`. See `ctan.org/mirrors` for the current list of mirrors, and general information about CTAN mirroring.

TEX Live incorporates a copy of the mirror list, which is updated every day or two. This is used for the network updates via `tlmgr`. The TL list filters out those mirrors which are more than 36 hours stale (which are relatively few), since it’s not useful to access old mirrors for current updates.

[5] `ams.org/meetings/lectures/meet-gibbs-lect`

A self-published LATEX book and ebook

Dan Grec, an inveterate traveler and travel book author, has described his “journey” producing his book, *The Road Chose Me*,[6] in an essay on the web page devoted to his travels.

Not wanting to go through the hassle of dealing with publishers, he decided to take the self-publishing route. Since he started out as a software engineer, he was used to (and not intimidated by) computer lingo, and chose LATEX as the main input medium.

One significant attraction to self publishing is also its biggest downside: It’s *all* up to you as the author!

He lists three goals for the result:

- The result should look professional, an effect he did not find satisfied by Word.
- The same source files should produce both the print book and the epub file.
- The whole process had to be automated.

After “a bunch of research”, he settled on LATEX:

> The more I dug into it, the more I realized it’s a full-blown programming language, and it would certainly give me the pixel-perfect layout control I was looking for, [. . .]

Searching further, he came across Pandoc, “often called the swiss army knife of document converters.” That is what he decided to use for the conversion from LATEX to an industry standard epub file. He did consider using Markdown, though he doubts that it can provide the fine control provided by LATEX.

The remainder of the essay presents the details of constructing the book, including technical physical details of the book layout and relevant examples of the code used. Whenever something confusing is encountered, an explanation is forthcoming.

Not all was smooth sailing — several later additions indicate changes in the production flow when certain online facilities became unavailable due to renaming or for other reasons. But LATEX itself remained stable.

The clarity of the presentation and explanations, and the subject matter of the book itself lead me to think that I would enjoy reading it, something I intend to consider seriously.

Thanks to Paul Campbell for calling this to my attention.

◇ Barbara Beeton
`https://tug.org/TUGboat`
`tugboat (at) tug dot org`

[6] `theroadchoseme.com/how-i-self-published-a-professional-paperback-and-ebook-using-latex-and-pandoc`

Hendrik Vervliet: 1923–2020

Jacques André

In the previous issue of *TUGboat*, Charles Bigelow reviewed two books by Hendrik D.L. Vervliet.[1] Alas, Hendrik Vervliet passed away on August 5th, 2020, at the age of 96.

In his career, Vervliet was a professor and librarian of the University of Antwerp as well as professor of book and library history of the University of Amsterdam. However, he worked mainly at the Museum Plantin-Moretus at Antwerp where he was still present a very few years ago: he never stopped working and, despite age and illness, was still writing papers, most recently on Granjon's civilité types,[2] his first love.

Vervliet was particularly known for his work on the sixteenth-century typefaces in France, Belgium and the Netherlands. Besides the two books on Granjon reviewed by Bigelow, his major books are, from my point of view:[3] *French Renaissance printing types: a Conspectus* (London, U.K.: The Bibliographical Society/ Printing Historical Society; New Castle, Delaware: Oak Knoll Press, 2010; `bibsoc.org.uk/publications/vervliet_printing_types`) and its complementary collection of essays, *The palaeotypography of the French Renaissance: Selected papers on sixteenth-century typefaces*, two volumes (Leiden: Koninklijke Brill NV, 2008; `brill.com/view/title/17788`).

The *Conspectus* is a meticulous catalogue of the four hundred and nine typefaces cut in sixteenth-century France. Each typeface is illustrated with an image of the characters at actual size, with examples of use in context, date of occurrences, and bibliography (an example page, cropped to the text, is shown below). This is done with Vervliet's customary rigor: precise measurement of a type with the x-height, the height of capital letters and the height of 20 solid text lines (a concept adapted from the old method of Proctor-McKerrow and which allows determination of the body size). These parameters minimize the errors of attribution of types, which are rather frequent. Remember, for instance, that Claude Garamont designed about two dozen Roman types with more or less the same appearance: without precise measurements, how could you distinguish them? Vervliet was an entomologist or a paleontologist! In fact, he defined himself as a palaeotypographer, and already some historians of type use his method. The history of types restarts on a good track.

I have been in touch with Hendrik for about twelve years and I always have been fascinated by his rigor together with his kindness. He always answered my questions and quite often went to the Plantin Museum to send me scans.

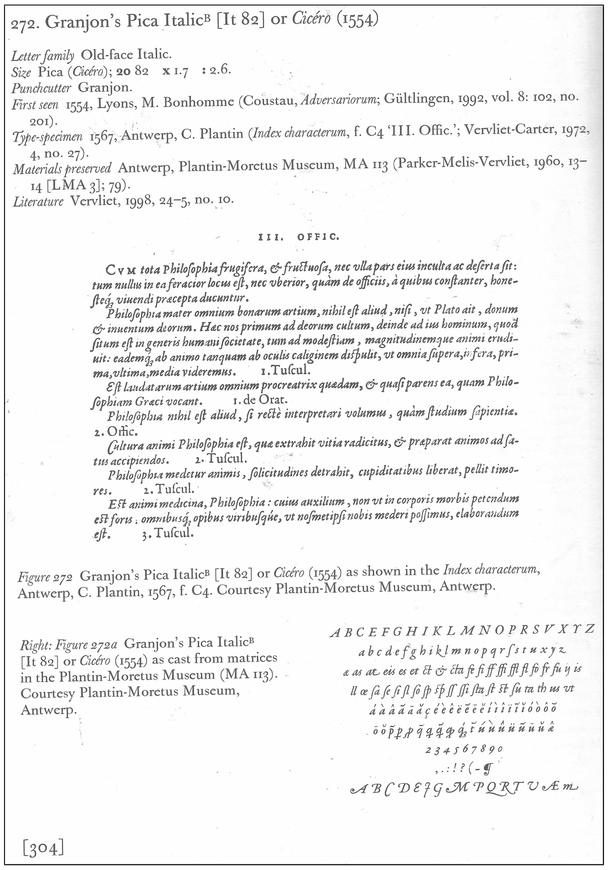
His passing has been as discreet as he was during all his life. However, he was a great man. Fortunately his printed work will remain. No valuable work on European typography can ignore his writings.

◇ Jacques André
`http://jacques-andre.fr`

[1] Charles Bigelow, 'Book reviews: *Robert Granjon, letter-cutter*, and *Granjon's Flowers*, by Hendrik D.L. Vervliet', *TUGboat* Vol. 41 (2020), No. 3, pp. 355–357; `tug.org/books/tb129reviews-vervliet.html`

[2] Hendrik D L Vervliet, 'Danfrie Reconsidered. Philippe Danfrie's (d. 1606) Civilite Types', *The Library*, Vol. 21, Iss. 1, March 2020, pp. 3–45; `academic.oup.com/library/article/21/1/3/5809221`

[3] About these books, more valuable opinions by historians of types can be found, e.g.: James Mosley, 'Review of *The Palaeotypography …*', *The Library*, Vol. 12, Iss. 2, June 2011, pp. 175–178; `doi.org/10.1093/library/12.2.175`; also, William Kemp and Henri-Paul Bronsard, 'The Types of the French Renaissance', *The Papers of the Bibliographical Society of America*, Vol. 106, No. 2, June 2012, pp. 231–256; `jstor.org/stable/10.1086/680637`



272. Granjon's Pica Italic^B [It 82] or *Cicéro* (1554)

Letter family Old-face Italic.
Size Pica (*Cicéro*); **20** 82 **x** 1.7 **:** 2.6.
Punchcutter Granjon.
First seen 1554, Lyons, M. Bonhomme (Coustau, *Adversariorum*; Gültlingen, 1992, vol. 8: 102, no. 201).
Type-specimen 1567, Antwerp, C. Plantin (*Index characterum*, f. C4 'III. Offic.'; Vervliet-Carter, 1972, 4, no. 27).
Materials preserved Antwerp, Plantin-Moretus Museum, MA 113 (Parker-Melis-Vervliet, 1960, 13–14 [LMA 3]; 79).
Literature Vervliet, 1998, 24–5, no. 10.

III. OFFIC.

CVM *tota Philoſophia frugifera, & fructuoſa, nec vlla pars eius inculta ac deſerta ſit: tum nullus in ea feracior locus eſt, nec vberior, quàm de officiis, à quibus conſtanter, honeſteq́ viuendi præcepta ducuntur.*

Philoſophia mater omnium bonarum artium, nihil eſt aliud, niſi, vt Plato ait, donum & inuentum deorum. Hæc nos primum ad deorum cultum, deinde ad ius hominum, quod ſitum eſt in generis humani ſocietate, tum ad modeſtiam, magnitudinemque animi erudiuit: eademq́ ab animo tanquam ab oculis caliginem diſpulit, vt omnia ſupera, infera, prima, vltima, media videremus. 1. Tuſcul.

Eſt laudatarum artium omnium procreatrix quædam, & quaſi parens ea, quam Philoſophiam Græci vocant. 1. de Orat.

Philoſophia nihil eſt aliud, ſi rectè interpretari volumus, quàm ſtudium ſapientiæ. 2. Offic.

Cultura animi Philoſophia eſt, quæ extrahit vitia radicitus, & præparat animos ad ſatus accipiendos. 2. Tuſcul.

Philoſophia medetur animis, ſolicitudines detrahit, cupiditatibus liberat, pellit timores. 2. Tuſcul.

Eſt animi medicina, Philoſophia: cuius auxilium, non vt in corporis morbis petendum eſt foris: omnibusq́ opibus viribusqúe, vt noſmetipſi nobis mederi poſſimus, elaborandum eſt. 3. Tuſcul.

Figure 272 Granjon's Pica Italic^B [It 82] or *Cicéro* (1554) as shown in the *Index characterum*, Antwerp, C. Plantin, 1567, f. C4. Courtesy Plantin-Moretus Museum, Antwerp.

Right: Figure 272a Granjon's Pica Italic^B [It 82] or *Cicéro* (1554) as cast from matrices in the Plantin-Moretus Museum (MA 113). Courtesy Plantin-Moretus Museum, Antwerp.

A B C E F G H I K L M N O P R S V X Y Z
a b c d e f g h i k l m n o p q r ſ s t u x y z
æ as at eis es et ct & cta ſe ſi ff ffi ffl fl fo fr fu y is
ll œ ſa ſe ſi ſl ſo ſp ſp ſſ ſſi ſta ſt st ſu ta th us vt
á à â ã ā ă ç é è ê ë ē ĕ í ì î ï ī ĭ ó ò ô õ
ō ŏ p̄ p̃ q̄ q̃ ę q́ ť ú ù û ü ũ ū ŭ â
2 3 4 5 6 7 8 9 0
, . : ! ? (- ¶
A B C D E F G M P Q R T V Æ m

[304]

An example page from Vervliet's *Conspectus*.

The TeX tuneup of 2021

Donald Knuth

This is the promised sequel to previous reports from 2008 [3] and 2014 [4]. Once again I'm immensely grateful to everybody who contributed potential errata to the "core" of TeX and METAFONT, and to the wonderful team of experts — led this time by Karl Berry — who checked their input carefully and filtered it down to a list of issues that definitely demanded attention. According to our longstanding plan, I received that list on 31 December 2020.

Karl will write separately about his role as a meta-filter. Let me just remark that, when I did the previous round of maintenance seven years ago, I had to deal with "more than two dozen potentially troublesome topics" [4]. This time the number was more than 250(!).

As in 2008 and 2014, both TeX and METAFONT have changed slightly and gained new digits in their version numbers. But again there's good news, because the changes are essentially invisible. I can't resist quoting once more from [3], because it reflects my unwavering philosophy (see [2]):

> The index to *Digital Typography* lists eleven pages where the importance of stability is stressed, and I urge all maintainers of TeX and METAFONT to read them again every few years. Any object of nontrivial complexity is non-optimum, in the sense that it can be improved in some way (while still remaining non-optimum); therefore there's always a reason to change anything that isn't trivial. But one of TeX's principal advantages is the fact that it does not change — except for serious flaws whose correction is unlikely to affect more than a very tiny number of archival documents.

Users can rest assured that I haven't "broken" anything in this round of improvements. Everyone can upgrade or not, at their convenience.

TeX Version 3.141592653

Let's get down to specifics. The new version of TeX differs from the old one in five not-completely-trivial ways, mostly having to do with corrections to TeX's attempts at recovering from errors.

The first two of these anomalies were found by Xiaosa Zhang and reported last summer on `tex.stackexchange` [5, 6]. He discovered a sneaky combination of keystrokes with which last year's TeX permitted you to get into `\batchmode` while continuing to interact at the terminal(!). Furthermore, he found that TeX's exit-and-edit option — typing '`E`' in response to an error prompt — was sometimes offered when it shouldn't have been, at times when an input file wasn't actively being read.

Both of those bugs could crash the system. So those two doors are now closed.

Another strange interaction had been noticed in 2017 by Udo Wermuth, who found that TeX could mysteriously seem to stop dead in its tracks while `\tracingparagraphs` was active. (The reason was that TeX had found and reported an error, which went into the transcript file. TeX was silently waiting for Udo to respond to that message, not realizing that messages are not echoed to the user's terminal while paragraphs are being traced.) In the future, TeX will not remain silent; the user will see the error message and be asked to respond.

Late last year Udo was bitten by quite a different sort of bug. This one has nothing to do with interaction, and it might possibly have occurred to others in some "real" runs of TeX during the past 35 years or so (although I doubt it): Previous versions of TeX have mistakenly allowed the ⟨replacement text⟩ of a macro to begin just after, say, '`#\bgroup`' — contrary to a rule that's stated clearly in the fine print of *The TeXbook* [A, bottom of page 275].

Henceforth TeX shall rigidly enforce that rule. Anybody who previously had written

```
\def\cs#1#\bgroup hi#1}
```

will now get an error message. And they should now write

```
\def\cs#1\bgroup{hi#1\bgroup}
```

if they want to reproduce the former behavior.

Finally, on 22 October 2020, Bruno Le Floch reported what might well turn out to be the historic "final bug in TeX." Again it's about macros. Suppose you've asked for nine parameters, specifying them one by one as `#1` through `#9`. Then you're not supposed to say '`#`' again until finishing off the ⟨parameter text⟩, because `#9` is TeX's upper limit. However, maybe you're feeling naughty and actually do type '`#`' improperly; TeX will complain:

```
! You already have nine parameters.
```

And its help message used to say

```
I'm going to ignore the # sign you just used.
```

Which was true. But henceforth the help message will state the *new* truth, which is that TeX will *also* ignore the next thing that *follows* the bad `#`. From now on, bad stuff won't be able to get through and foul things up.

doi.org/10.47397/tb/42-1/tb130knuth-tuneup21

All five of the bugs mentioned above are big ticket items, worth `0x$80.00` ($327.68) at the Bank of San Serriffe [1], because they exposed serious (although rarely tweaked) deficiencies in TeX's implementation. Besides those, TeX 3.141592653 also incorporates a number of other comparatively minor bugfixes. For example, with previous versions you could really screw up the end of your transcript file by saying `\newlinechar=‘p`.

Plain TeX has also changed in a minor way, for consistency: It now ensures that `\muskip255` and `\toks255` are available as "scratch registers" (never allocated by `\newmuskip` or `\newtoks`). The new incarnation defines `\fmtversion` as `3.1415926535`.

The least trivial of these additional changes are noted in updates to *TeX: The Program* [B], which now can be found in PDF form on the webpage [9] and in a file called `errata.tex`. They appear also in files called `errorlog.tex`, `tex82.bug`, and `plain.tex`. But the full truth resides, as always, in the updated master source file `tex.web`. All five of those key files continue to appear online in directory `systems/knuth/dist` of the CTAN archive [7].

The error log of TeX began in 1978, and its first 14 years are documented in [8, Chapters 10 and 11]. The next several years are covered in [2, Chapter 34], ending with bug #933, dated 10 March 1995 and found by Peter Breitenlohner. And hey, who knows, the log may at last have gained its final entry — which is #957.

While I was preparing this round of updates, I was overjoyed to see how well the philosophy of literate programming has facilitated everything. This multifaceted program was written 40 years ago, yet I could still get back into TeX's darkest corners without trouble, just by rereading [B] and using its index and mini-indexes! I can't help but ascribe most of TeX's success to the fact that it has enabled literate programming.

METAFONT Version 2.71828182

And what about TeX's partner? I almost thought that METAFONT's version number should stay at 2.7182818, because the outputs of the newly upgraded program don't differ from what would have been obtained last year except in trivial ways. For example, some of the help messages are now slightly different.

However, the two TeX bugs found by Xiaosa Zhang apply also to interaction with METAFONT. Therefore I now believe, in view of [6], that the historic "final bug in METAFONT" was found on 03 July 2020, although he was actually using TeX.

TeXware and METAFONTware

I made minor updates to the master `web` files for more than a dozen other programs, mostly to correct spelling errors, to add Oxford commas, and to make them more consistent with each other. Doug McKenna and David Fuchs found two obscure bugs in TANGLE and WEAVE that hadn't been noticed since the early 80s(!). Here is a current list of all the `web` files for which I have traditionally been responsible:

name	current version	date
`dvitype.web`	3.6	December 1995
`gftodvi.web`	3.0	October 1989
`gftopk.web`	2.4	January 2014
`gftype.web`	3.1	March 1991
`mf.web`	2.71828182	January 2021
`mft.web`	2.1	January 2021
`pltotf.web`	3.6	January 2014
`pooltype.web`	3.0	September 1989
`tangle.web`	4.6	January 2021
`tex.web`	3.141592653	January 2021
`tftopl.web`	3.3	January 2014
`vftovp.web`	1.4	January 2014
`vptovf.web`	1.6	January 2014
`weave.web`	4.5	January 2021

Typographic errors and other blunders

So far I've only been discussing anomalies that were detected in the software. But of course people have also reported problematic aspects of the documentation — which may actually be the hardest thing to get right. Although *The TeXbook* [A] has been under intense scrutiny for almost forty years, readers from around the world have continued to find significant ways to improve it, for instance by amending the answers to some of the more difficult exercises.

The most important new changes to *The TeXbook* involve the way it describes the intricate details of spacing within math formulas. My original discussion of "Inner atoms" was unfortunately quite wrong; yet apparently nobody noticed those mistakes until December 2018, when Sophie Alpert identified some key inconsistencies in Appendix G. Several pages of fine print needed to change, and of course I'm happy to have the true story finally nailed down.

Other significant amendments include more precise syntax regarding things like discretionaries, hyphens, and patterns. Many enhancements have also been made to the index. Altogether, it has turned out that at least 93 of *The TeXbook*'s 483 pages have been improved in some way (about 19%).

And *The METAFONTbook* has improved even more — on 128 of its 361 pages (35%). A typo was even found in its Table of Contents! Two of the leading contributors to this bug hunt, Hu Yajie and Udo Wermuth, must surely rank among the absolutely top proofreaders of the world, possibly of all time. In particular, Yajie not only suggested many mutually orthogonal ways to apply spit and polish to this multidisciplinary book, but also helped me to straighten out the formal syntax of METAFONT's expressions.

Computers & Typesetting Jubilee

One of the highlights of my life took place on 21 May 1986, when Addison–Wesley arranged for an all-day event [10] at Boston's Computer Museum, to celebrate the completion of TEX and METAFONT. It was the first time I'd gotten a glimpse of the books [A, B, C, D, E], which were literally "hot off the press." And my fondest recollection from that day was the beaming face of A–W's cofounder, Mel Cummings, as he held those five volumes in his hands with obvious pride and satisfaction. He had spent his life in the printing industry, and devoted it to producing technical books of the finest quality; so I was delighted to see his delight.

Having just looked again at each of the 2668 pages in those volumes, I can't help but feel a reflected glow of pride from being associated with this extraordinary collaborative undertaking, especially now that it has reached a new peak of perfection. It seems fair to say that these books represent a significant milestone in the history of typography, as they self-describe every detail of the computations that went into their own composition. "If copies of these books were sent to Mars, the Martians would be able to use them to recreate the patterns of 0s and 1s that were used in the typesetting." [10]

Therefore I'm extremely pleased to announce that Addison–Wesley has just published brand new printings of Volumes A, B, C, and D, dated February 2021, a "35th Jubilee Edition" that contains all of the refinements that were introduced during this tuneup. At last the i's have all been really properly dotted and the t's have all been really properly crossed! (The 2017 printing of Volume E remains up to date.)

Conclusion

The TEX family of programs seems to be nice and healthy as it continues to approach perfection. Chances are nil that any documents produced by previous versions of TEX or METAFONT will be affected by the changes in the new versions. Volunteers have been stalwart contributors to this success in optimum ways.

Stay tuned for The TEX Tuneup of 2029?!

References

[1] The Bank of San Serriffe, account balances. `https://www-cs-faculty.stanford.edu/~knuth/boss.html`

[2] Donald E. Knuth, *Digital Typography* (Stanford, California: Center for the Study of Language and Information, 1999), xvi + 685 pages. (CSLI Lecture Notes, no. 78.) The second printing (2012) contains numerous corrections.

[3] Donald Knuth, "The TEX tuneup of 2008," *TUGboat* **29**:2 (2008), 233–238. `tug.org/TUGboat/tb29-2/tb92knut.pdf`.

[4] Donald E. Knuth, "The TEX tuneup of 2014," *TUGboat* **35**:1 (2014), 5–8. `tug.org/TUGboat/tb35-1/tb109knut.pdf`.

[5] 潇洒张, `https://tex.stackexchange.com/questions/551313` (27 June 2020).

[6] 潇洒张, `https://tex.stackexchange.com/questions/552166` (03 July 2020).

[7] CTAN: Comprehensive TEX Archive Network, `https://ctan.org/`.

[8] Donald E. Knuth, *Literate Programming* (Stanford, California: Center for the Study of Language and Information, 1992), xvi + 368 pages. (CSLI Lecture Notes, no. 27.)

[9] *Computers & Typesetting*, `https://www-cs-faculty.stanford.edu/~knuth/abcde.html`.

[10] Barbara Beeton, Peter Gordon, and Donald Knuth, "*Computers & Typesetting* coming out party," *TUGboat* **7**:2 (June 1986), 93–98. `tug.org/TUGboat/tb07-2/tb15knut.pdf`. (My remarks have also been reprinted, with amendments, as Chapter 28 of [2].)

[A] Donald E. Knuth, *The TEXbook* (Reading, Mass.: Addison–Wesley, 1984), x + 483 pages. Also published as *Computers & Typesetting*, Volume A. Currently in its 35th printing (paperback, 2017) and 23rd printing (hardcover, 2021).

[B] Donald E. Knuth, *Computers & Typesetting*, Volume B, *TEX: The Program* (Reading, Mass.: Addison–Wesley, 1986), xvi+594 pages; fifth printing (1994), xvi + 600 pages. Currently in its 11th printing (hardcover, 2021).

[C] Donald E. Knuth, *The METAFONTbook* (Reading, Mass.: Addison–Wesley, 1986), xii + 361 pages. Also published as *Computers & Typesetting*, Volume C. Currently in its 14th printing (paperback, 2017) and 10th printing (hardcover, 2021).

[D] Donald E. Knuth, *Computers & Typesetting*, Volume D, *METAFONT: The Program* (Reading, Mass.: Addison–Wesley, 1986), xvi + 560 pages; third printing (1991), xvi + 566 pages. Currently in its 9th printing (hardcover, 2021).

[E] Donald E. Knuth, *Computers & Typesetting*, Volume E, *Computer Modern Typefaces* (Reading, Mass.: Addison–Wesley, 1986), xvi + 588 pages. Currently in its 8th printing (hardcover, 2017).

◇ Donald Knuth
`www-cs-faculty.stanford.edu/~knuth`

TeX entomology in 2021

Karl Berry

Our peerless Ur-Wizard asked me to write a few words about the filtering and testing process for this year's TeX tuneup, which I am glad to do.

First, I am beyond grateful to the many other people who volunteered to help with the job, and without whom it would have been an impossible task: for TeX, Donald Arseneau and David Fuchs; for METAFONT, Bogusław Jackowski, Piotr Strzelczyk, and Jerzy Ludwichowski; and for everything, Barbara Beeton, who was Don's entomologist "from the beginning" until this cycle. She still read every bug that came in, and provided much useful history and advice. For many years, Peter Breitenlohner, who sadly passed away in 2015, and Chris Thompson were also key members of the group of vetters.

As some sort of bonus to the hundreds of bug reports for TeX & METAFONT that I sent to Don this year, I also sent a dozen or so reports on CWEB, for which I relied on Andreas Scherer to confirm and elaborate. Andreas is the creator and maintainer of the `cwebbin` descendant of CWEB now used in TeX Live; as has been more widely announced elsewhere, Don asked Andreas to take over as the maintainer of the official CWEB as well.

How bugs are reported: Anyone can send reports either to the public mailing list `tex-k@tug.org` (`lists.tug.org/tex-k`), or to me personally if they want to retain privacy. This is described both on Don's web page (`www-cs-faculty.stanford.edu/~knuth/abcde.html#bugs`) and on the general information page I created for bugs in TeX & METAFONT, `tug.org/texmfbug`. This worked out well, with a number of incoming reports on `tex-k` being answered by people other than me, a most welcome outcome.

After some initial triage by me (some reports had been fixed in later printings, etc.), I sent reports to the appropriate vetters for discussion. Many could be easily confirmed, such as obvious typos — but even there, one of us typically searched all the books and other `WEB` sources for the same typo, since so much material is copied around the TeX system. Occasionally these searches resulted in bug reports for the few `.web` files not maintained by Don, notably Oren Patashnik's BibTeX and Tom Rokicki's PKtype and PKtoGF, which I directed accordingly.

Other reports took hours of analysis to determine the root cause, and/or the details of what we could sensibly suggest to Don, if anything. The champion in this regard was perhaps Sophie Alpert's bug about the treatment of Inner atoms in *The TeXbook*, which Don mentioned in his report. Here, we enlisted Udo Wermuth's expert help in checking if we were on the right track with the myriad issues that arose.

One of the most useful checks for all code (as opposed to textual) bugs was possible due to David Fuchs, who has implemented a complete build system, including a runtime library, which can compile and run all of the original `WEB` sources without any change (files) whatsoever. This made it possible to definitively determine whether some particular strange behavior was due to the Web2C or other infrastructure, or was truly in the original source. DRF discusses the system in his article published in *TUGboat* 41:1, also available at `tug.org/texmfbug/fuchs-knuthbug.html`.

Thanks to Robin Laakso in the TUG office and long-time TeX colleague Oleg Katsidatze, the original printed reports with Don's handwritten comments have recently been sent back to everyone who provided physical addresses. In case of physical mail being lost, we have a scan of all the paper and can resend copies.

Looking ahead, I plan to record new confirmed incoming bugs, to be reviewed in the next tuneup, at `tug.org/texmfbug/newbug.html`, so potential reporters can more easily check for known bugs. A sibling page, `tug.org/texmfbug/nobug.html`, lists a few especially noteworthy non-bugs; it's not practical to list every declined report, but publicly listing the most significant ones seems worthwhile.

In the `\end`, coordinating these bug reports has given me a whole new level of appreciation for the TeX system. Don, thank you for creating it, and giving it to the world.

◇ Karl Berry
`karl (at) freefriends dot org`
`https://tug.org/texmfbug`

doi.org/10.47397/tb/42-1/tb130berry-filter

Hyphenation exception log

Barbara Beeton

This is a brief update of the list of words that TEX fails to hyphenate properly for U.S. English. The full list last appeared in *TUGboat* 16:1, starting on page 12, with periodic updates, most recently in *TUGboat* 39:2, p. 152. The complete list is given at `tug.org/TUGboat/Contents/listtitle.html #hyphenationexceptionlog`.

The full list, updated through the end of 2020, is posted on CTAN at `ctan.org/pkg/hyphenex`, and in the TEX Live package `hyphenex`. The TEX-usable output is the file `ushyphex.tex`.

The full list has been rearranged, adding a third group, devoted to chemistry and related disciplines. The rationale for this revision is that the vocabulary for this area is common internationally, and the hyphenation (since it is based on etymology) is likely to be the same or very similar regardless of language, allowing this material to be reused easily.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, unabridged, although we very occasionally omit or add breaks to the dictionary's on semantic grounds. Boundless thanks to Win Treese, lover of dictionaries, and owner of the paper edition of the *Third*, who has checked items since I lost access to the copy at AMS on my retirement and then to those at libraries closed on account of the coronavirus.

Hyphenation for other languages

Hyphenation patterns and rules used by `babel` and `polyglossia` are maintained by a group of volunteers who are receptive to problem reports and new submissions. Please see `tug.org/tex-hyphen` for information and contact.

The list — English words

`ar-eas`	areas
`cauliflower`	cau-li-flow-er
`colophon`	col-o-phon
`cricket`	cric-ket
`eggshell`	egg-shell
`elec-troen-cephalo-gram`	elec-tro-en-cephalo-gram
`epigraphs`	epi-graphs
`grapheme`	graph-eme
`graphemic`	gra-phe-mic
`graphetic`	gra-phe-tic
`grapholin-guis-tic`	grapho-lin-guis-tic
`home-o-mor-phi-cally`	ho-meo-mor-phi-cally
`in-terele-ment`	inter-ele-ment
`legacy(ies)`	le-ga-cy(ies)
`legate`	leg-ate
`lega-tion`	le-ga-tion
`Lu-a-TeX`	Lua-TeX
`Lu-aLa-TeX`	Lua-LaTeX
`macros`	mac-ros
`MakeIn-dex`	Make-Index
`markup`	mark-up
`passover`	pass-over
`polyvinyl`	poly-vinyl
`potato`	po-ta-to
`prepend(s,-ed,-ing)`	pre-pend(s,-ed,-ing)
`provider`	pro-vi-der
`scriptscript`	script-script
`steril-ity`	ste-ril-i-ty
`stylesheet`	style-sheet
`the-sis(es)`	thesis(es)
`ther-mal`	ther-mal
`tiger`	ti-ger
`tran-spile(s,d)`	trans-pile(s,d)
`tran-spiler`	trans-pi-ler
`tran-spiling`	trans-pil-ing

Names and non-English words used in English text

`Ar-bor-Text`	Arbor-Text
`Athens`	Ath-ens
`Granjon`	Gran-jon
`Hewlett`	Hew-lett
`Joseph`	Jo-seph
`Max-i-m-il-ian`	Max-i-mil-ian
`Mesopotamia(n)`	Mes-o-po-ta-mia(n)
`Monaco`	Mon-aco
`Mon-tag-nard`	Mon-ta-gnard
`Packard`	Pack-ard
`Pak-istan`	Paki-stan
`Phineas`	Phin-eas
`Pho-to-shop`	Photo-shop
`Robe-spierre`	Robes-pierre
`Schol-ar-TeX`	Scholar-TeX
`ToUni-code`	To-Unicode
`Venezuela`	Ven-e-zu-e-la
`Wingdings`	Wing-dings
`Xe-rox`	Xerox
`polyvinyl`	poly-vinyl

Words used in chemistry and similar fields

`ac-etaminophen`	acet-a-mi-no-phen
`acetyl-cholinesterase`	ace-tyl-cho-lin-ester-ase
`acetyl-choline`	ace-tyl-cho-line
`acetyl-glu-cosamine`	ace-tyl-glu-cos-amine
`adeno-syl-me-thio-n-ine`	ade-no-syl-me-thi-o-nine
`paradichlorodiphenyl-trichloroethane`	para-dichloro-diphenyl-trichloro-ethane

◇ Barbara Beeton
`https://tug.org/TUGboat`
`TUGboat (at) tug dot org`

doi.org/10.47397/tb/42-1/tb130hyf

Year 2020 at GUTenberg

Jérémy Just

Abstract

Starting in January 2020, the French-speaking TeX users group underwent a crisis that could have led to its dissolution. A vote rejected the idea of dismantling the association, but the past months left scars.

1 Context

The year 2020 has been stressful for many out-of-human-control reasons, due to the Covid-19 pandemics. Since the beginning of this period, I've been pleased to see people investing their forced spare time in new team projects, proactively looking for the satisfaction of pursuing a common objective with their fellows, remotely. I did get involved in such new team projects myself, translating documentation, providing help to LaTeX newcomers and solving problems with Ti*k*Z. I personally feel this involvement, this feeling of "working together", has been instrumental in getting through this period with a healthy mind.

I've been taking care of GUTenberg, the French-speaking TUG, for quite a long time now, but surprisingly, my work for GUTenberg, this year, is not part of this satisfying involvement, even though it kept me busy for most of my spare time, and even more. The GUTenberg association underwent a crisis triggered by a small group of people. It is now mostly solved in itself, but it has apparently put an end to the friendly working atmosphere that had been the rule for years in the association.

2 The crisis

In mid-January 2020, five persons paid their membership fees on the same day, and then gathered with three others to send an email to our public discussion list, asking for the dissolution of the association. Five were inactive members of the board; some of them had not paid any membership fees since 2008 or 2012.

There followed a storm of messages on our mailing list, some of them very constructive, others containing near-insults.

In spite of the urgent need for a General Meeting (GM), it was as difficult as in previous years to agree on a date to hold it. The GM was finally held on November 14$^{\text{th}}$, by teleconference. 35 persons attended it.

This meeting was a long-awaited opportunity to chat about our views for GUTenberg. We had some great moments of sincerity during the meeting, for example when some members of the board explained that they had constituted a "shadow cabinet" (these are their words) and had been coordinating to block *on purpose* any project. As the former president who had to face this situation, I really appreciated finally having an explanation for all those blocks.

The question of the dissolution had been asked in advance to our 104 current members, through an electronic vote. Out of 54 persons who voted, 5 were in favour of the dissolution of GUTenberg, and 49 were against it.

We also had to renew the whole board of administrators. Elections at GUTenberg have long been a mere formality, as there are usually fewer candidates than the twelve open seats. People volunteer because they want to help on a specific project led by the association, or just because they're interested in helping in the general management of the association. But this year, seven candidates applied months before the GM and constituted an electoral list, asking voters not to vote for anyone other than them (some had co-signed the email that triggered the crisis, and had set up the "shadow cabinet").

In the end, we managed to fill the twelve seats of the board and, more importantly, to have new persons elected. That's a personal point of view, but I consider it very important to regularly welcome newcomers onto the board, for the association to stay representative of the current community.

A more extensive report of our GM can be found in *La Lettre GUTenberg* n°41 (in French): `www.gutenberg.eu.org/Lettre-GUTenberg`

3 And now?

A few months have now passed since the General Meeting. I must admit that the atmosphere among the board is far from the friendly one that we had when the board was working smoothly.

I feel (personal point of view again) that newcomers haven't been integrated, and that too many discussions take place outside of our discussion list. An association cannot work based on exclusion.

I'm really worried about the future of GUTenberg. It's one thing to see 49 persons voting against a dissolution. It's a completely different problem to have the association *doing things* in the interest of the LaTeX community.

The best I can do is urge the community to gather around projects, and see how GUTenberg can provide support.

That being said, I can already see one project for which our General Meeting has been beneficial: we've been hosting a French-speaking LaTeX FAQ on our server for years (the project started in 2011). The

Jérémy Just

doi.org/10.47397/tb/42-1/tb130just-gut

wiki was running at a low pace, with limited feedback from its users. But since last fall, we're seeing a dramatic increase in its activity plots, with more active contributors, and more significant updates to the contents (see the plots below)!

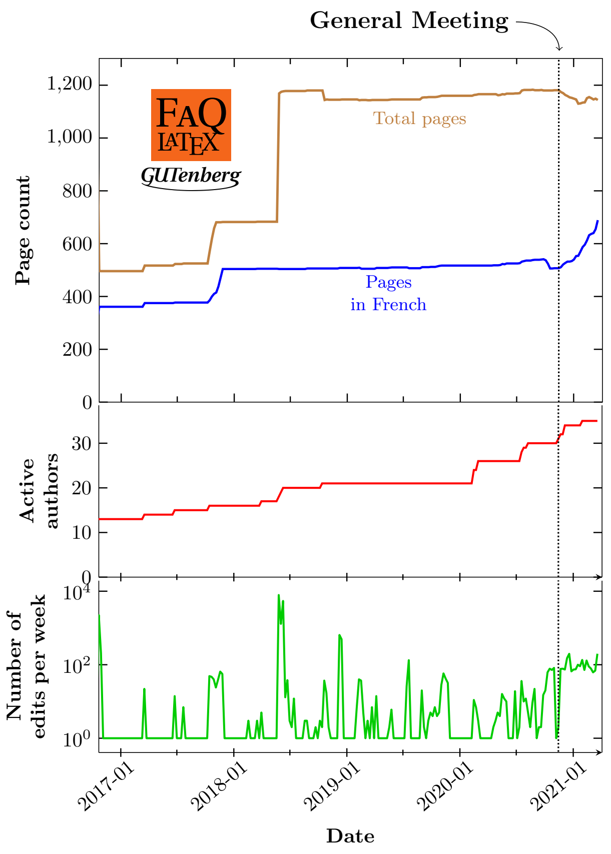
For those interested, the URL of the FAQ is: `faq.gutenberg.eu.org`

I sincerely hope this project will be followed by others!

4 Acknowledgements

I would like to thank all the members of the international LaTeX community who have asked for some news from GUTenberg, all year long. And more specifically, I gratefully acknowledge the support we received from Arthur Rosendahl, Robin Laakso and Barbara Beeton (Barbara also kindly asked me to write this report for *TUGboat*).

◇ Jérémy Just
Lyon, France
`jeremy (at) jejust dot fr`
`https://www.gutenberg.eu.org`
ORCID 0000-0003-0842-9808



Lapses in TeX — a look backward

Barbara Beeton

TeX has now been used "in the wild" for over 40 years, so it's possible to look back and examine this tool and its ecosystem to see what decisions might have been made differently to avoid problems that still exist today. Some deficiencies are the result of limited hardware or facilities (such as Unicode) that did not exist at the time when TeX was created (1978–79). However, others could reasonably have been implemented differently within the existing limitations, and these are what will be examined here.

By what authority?

Why can I claim authority to examine this topic?

I was sent to Stanford in the summer of 1979 to learn TeX under the tutelage of the TeX Project. In preparation for this assignment, I collected a number of "good bad examples", problems that had actually occurred in publication production at the American Mathematical Society (AMS). In the event, these examples proved to be well chosen; a number of them appear in Appendix D of *The TeXbook* [5], and the command `\firstmark` was newly created to address an unmet need, evidenced by the insertion of its syntax handwritten by Don Knuth in my copy of the ur-TeX manual.

For many years, I was Don's bug collector (or "TeX entomologist" as I preferred), distributing reports to individuals whom Don recognized as sufficiently knowledgeable to determine whether a problem was or was not a bug, communicating their analysis to the submitter, organizing the reports for communicating to Don on his predetermined schedule, and in turn communicating Don's response to the submitter of the first report. The bug collecting function was turned over to Karl Berry with the completion of the 2014 review cycle.

Initial conditions and philosophy

The rationale behind the creation of TeX has been thoroughly covered in Knuth's 1978 Gibbs Lecture [4]. The first implementation in SAIL was eagerly adopted by some local academics and visitors to Stanford who had access to the requisite hardware. But it soon became apparent that a portable implementation would serve a much larger audience who needed this tool, so a widely available subset of Pascal was chosen for TeX82, and a new tool, WEB, was developed that would make it possible to code the program and publish it in an intelligible "public" form that could be read and understood by a technically literate audience (*TeX: The Program* [6]).

doi.org/10.47397/tb/42-1/tb130beeton-lapses

We must recognize that the original target for TEX's output was *print*. TEX predates the World Wide Web, Unicode, and the high-resolution screens attached to very fast processors that have made it possible for someone to read a document directly from the electronic representation. In order to meet these "new" requirements most effectively, the document structure needs to be preserved in the final electronic form. Several elements important to this goal will be evident in what follows.

8-bit limitation

It's a bit unfair to fault this limitation, since Unicode didn't exist until 1987, long after Knuth had returned to his work on *The Art of Computer Programming.* TEX78 was based on 7-bit fonts; the basic ASCII arrangement was carried into TEX82, still with seven bits "live", although eight bits were built into the code structure, and full eight-bit input support was added in 1989.

A 256-character font encoding was devised in 1990 at a TEX meeting in Cork [2]. This arrangement didn't match any of the standard 8-bit European encodings; however, an attempt was made to accommodate all the accented letters, variants and digraphs required for Western European languages. Input encodings were then developed to permit direct input of these alphabetic characters from keyboards that provided them.

It was possible within this limitation to implement Cyrillic for most Slavic languages, with input based on the Mathematical Reviews transliteration, a rather complex ligaturing mechanism, and requiring only a few control sequences [1, p. 17]. Other alphabetic fonts created for this "unextended" version (as reported in *TUGboat*) were Vietnamese, Hebrew, modern Greek, Arabic, Croatian glagolitic, Ethiopic, and the International Phonetic Alphabet (IPA).

With the introduction of XeTEX in 2005 and LuaTEX at about the same time, input was opened up to accept Unicode natively, and support was provided for OpenType and TrueType fonts. For the basic TEX engine (with LaTEX preloaded) `\inputenc{utf8}` was implemented, but input of accented letters is converted to the `\cs`⟨*letter*⟩ form for processing, and TEX fonts are still limited to 256 characters.

For languages where the input order of characters does not necessarily match the display order, 8-bit input is insufficient. Supporting this would have required extensive (breaking) changes to the program, exceeding Don's requirements. An "early" extension to TEX, Omega [3], was first presented in 1994, but has since been abandoned. Nonlinear composition requirements are now implemented by font-shaping mechanisms, not by the main composition engine (cf. [8]).

Limitation of the character box

The "shape" of a TEX character is defined as a rectangle (or a parallelogram for sloped characters) with the origin at the baseline on the left side (for left-to-right scripts) and a width measured at the baseline. TEX uses only the metrics. While this simplicity permits efficient calculation of necessary values for line and paragraph breaking, without adjustments the spacing of adjacent characters may not be optimal, regardless of the quality of their design.

This model is most appropriate for Western alphabetic languages, which typically have reliably "restricted" shapes, even taking diacritics into consideration. Font-shaping mechanisms developed to handle more complex scripts have been mentioned in the previous section.

Within the existing design, two adjustments are provided, recorded in the `.tfm` files used by TEX82: explicit kerning and the "italic correction", a value indicating the overhang of a tall sloped letter. There is no corresponding adjustment for the left-hand side, which results in the following suboptimal appearance, depending on the letter beginning a new line:

Watch the left margin.
This is one example.
The normal flush left margin.
What about this?
One last line.

Hermann Zapf's microtypography addressed this.

In the Computer Modern fonts no kerning is specified between any lowercase letter and a following uppercase. Since CamelCase was not in heavy use when TEX was created, we ignore this omission here. But kerning between an uppercase letter and a following lowercase is also nearly nonexistent, leading to the unfortunate spacing of the combination "Av", among others, which could have been avoided.

Let's look at the situation where punctuation follows uppercase. This too is unkerned, and can yield particularly unsightly results in bibliographies, which are often overrun with initials (and not easily managed). Some examples, showing manual adjustments that have been used in this issue:

P.O. Box	`P.O. Box`
P. O. Box	`P.\thinspace O.~Box`
W.J. Martin	`W.J. Martin`
W.J. Martin	`W\kern-.05em.\kern.07em J\kern.01em. Martin`

In math, the situation is somewhat different. While in text the spacing of adjacent letters is set based only on their origins and width, in math the italic correction is always applied, increasing intercharacter spacing. This is particularly noticeable in sub- and superscripts:

$$A_xB \quad A_fB \quad P_xQ \quad P_fQ$$

This can be adjusted manually by applying a negative or positive thin space, and Knuth in *The TEXbook* recommends manual attention. But this can get tiring, and some instances can be missed in proofreading, leading to inconsistent appearance.

A recent post on `tex.stackexchange.com`[1] contained a repetitious example of bad spacing that was easily addressed by an ad hoc definition.

$$\biggl({\partial f^0 \over \partial \eta_t} \biggr) \qquad\qquad \biggl({\partial\mkern-2mu f^0 \over \partial \eta_t} \biggr)$$

This definition is applied in the obvious location.

```
\def\partialf{\partial\mkern-2mu f}
$$ \biggl(
  { \partial f^0 \over \partial \eta_t }
  \biggr)
$$
```

Another `tex.stackexchange` post[2] asks for "optically balanced space" in expressions such as the following.

$$\mathbf{e}_i \cdot \mathbf{e}_j \neq \mathbf{e}_i \times \mathbf{e}_j \neq \mathbf{e}_i \wedge \mathbf{e}_j \neq \mathbf{e}_i \otimes \mathbf{e}_j$$

Compare the more uniform spacing of this expression, where subscripts don't disturb the "line".

$$\mathbf{a} \cdot \mathbf{b} \neq \mathbf{a} \times \mathbf{b} \neq \mathbf{a} \wedge \mathbf{b} \neq \mathbf{a} \otimes \mathbf{b}$$

The requested spacing is not possible without knowing more details about the shapes of all characters that can appear in math expressions. (Whether or not such a request is reasonable or desirable has been asked in a comment to the request. This question will be ignored here.)

Limitations imposed by the line-breaking algorithm

Baselines aren't "frozen" until the end of a paragraph, and it's bad style to break a page between text and a display, so in effect, the display is part of the preceding paragraph. If a display is set in a font size different from (usually smaller than) that of the preceding paragraph, the wrong baselines are applied to that text.

It's possible to adjust this manually, but many (most?) people are unaware of it, and scrunched paragraphs can be seen in otherwise fine math papers. Look at this paragraph; the display that follows is set in `\footnotesize`.

$$\begin{aligned} a + b &= c \\ d + e &= f \end{aligned}$$

[1] `https://tex.stackexchange.com/q/592191`

[2] `https://tex.stackexchange.com/q/581045`

Even if the text following the different-sized display doesn't start a new paragraph, the normal baselines are restored. This code produced the example:

```
\begingroup \footnotesize
$$
\eqalign{
  a + b &= c \cr d + e &= f \cr }
$$
\endgroup
```

Line breaking by paragraph places some limitations on desirable formatting.

- If a paragraph is broken at the end of a page, and a different page width is wanted on the next page, that change can't be applied automatically.
- Sometimes, a by-line view is preferable to a by-paragraph view, for example, to facilitate communication of editorial suggestions. A change in line width affects line numbers. (Line numbers are required, for example, for some legal documents.)
- It's not easy to reflow material for, e.g., screen presentations.

Two-dimensional material vs. "the grid"

In addition to the baseline anomaly shown above, the design of TEX makes it difficult to maintain uniform baselines throughout a document. Historically, printers have tried to achieve layouts in which lines of type match up on the front and back of a page; this was particularly important on thin paper, where what's on the other side might "read through" if it is set between the lines on the reading side. This uniform spacing is known as "grid typesetting".

Uniform baselines aren't difficult to achieve with straight text, but several forms of printed material are by definition two dimensional — most notably math, chemical structures and music. Forcing them onto a grid can result in either squeezing or inserting excess space, degrading comprehensibility.

A few TEX practitioners have devised means to achieve grid layout, but in the presence of especially complex math structures, the problem may be intractable.

A fraction anomaly

In math processing, the gap between a fraction line and the numerator or denominator is defined to be the same height as the thickness of the fraction line; this is governed by the setting of font dimension 8 in the math extension (family 3) `.tfm` file (rule 10 in Appendix G of *The TEXbook*. with its application to fractions explained in rule 15). If for some reason the fraction line is made thicker, the gap quickly becomes too large.

The (primitive) command `\abovewithdelims` demonstrates the problem. An alternative is implied by the description of the command `\above`, which accepts just a ⟨*dimen*⟩. The example in *The TEXbook* shows `1pt` as the dimension, and this looks promising, but as it turns out, this is treated in the same way as `\abovewithdelims`, so if a larger dimension is specified, the gap expands accordingly.

$$\frac{af}{fa} \qquad {af \abovewithdelims.. 3pt fa} \qquad {af \above 1pt fa} \qquad {af \above 4pt fa}$$

Ideally, the gap should either increase more slowly, or require an explicit setting. It's likely that the need to accommodate such a situation was never predicted; it's quite rare. But when it does occur, the result is a distinct surprise, and a search for documentation doesn't find any.

This code produces the example shown above:

```
$$
 {af \over fa} \qquad
 {af \abovewithdelims.. 3pt fa} \qquad
 {af \above1pt fa} \qquad {af \above4pt fa}
$$
```

Hyphenation discrimination

All permissible hyphenation points are weighted the same, but in English (as in many languages), it is often better to preferentially hyphenate a compound word at the junction of its lexical elements. This is especially important in chemical and similar terms.

For example, let's adopt a convention that a hyphen shows the position of a normal hyphen, while an equals sign shows the location of a lexical (preferable) breakpoint, separating elements of a compound.

```
  pho-to=syn-the-sis
  pa-ra=di-chlo-ro=ben-zene
```

(Aside: All proper breakpoints in "photosynthesis" are identified by TEX's (U.S.) hyphenation algorithm, but only the last in "paradichloroben-zene".)

The dictionary used for developing the U.S. patterns has hyphenation indicated at only one level, so this limitation is not surprising. But the dictionary underlying the British patterns records two levels, based primarily on etymology, but also on syllabification. (The U.S. patterns, based on pronunciation, sometimes coincide with etymology, but it's not guaranteed. Technical terms are more likely to be hyphenated according to etymology.)

A two-level mechanism is even more desirable for agglutinative languages like German, and alternative mechanisms have been devised where necessary, but this would have been simpler had a two-level mechanism been included in the design.

Finally, if a text is to be properly reconstructed from the printed output, it must be possible to distinguish between hyphens that are inherent in the text and those introduced by the hyphenation routine. This information is lost after composition, and failure to restore it properly may change the meaning. In addition, inclusion of language markers in the output would be a useful adjunct here.

Missing spaces

In the original design of TEX output, only characters and their (relative) positions are present. The space character is absent; what is seen on a page is the illusion of a space, provided by the gap separating not-quite-adjacent glyphs. This is not remedied by PDF, and spaces can be lost when text is cut-and-pasted, unless the gap exceeds a certain minimum width.

But a different approach might have been taken, namely the marking of word boundaries. Nelson Beebe states that this "could have been trivially included in the original SAIL version with no significant memory increase." The presence of such markers could support checking for delimiter balancing, spelling, grammar and syntax, all of which are best done on the typeset form, not the input.

One way! Do not back up!

TEX input is processed in a one-way stream, with no provision made for backtracking. This means that it may be impossible to trap and save the last character or token in a string without parsing the whole string or otherwise predefining some particular feature that can be used to isolate it. (It *is* possible to apply special processing to the last line of a paragraph, calling on `\lastskip`, `\lastbox`, etc.)

One situation requiring special treatment of a single terminal character is the different shape of a terminal sigma in Greek (ς vs. σ). In one approach, the letter 'c' is input following a final sigma, and the two letters are ligatured to produce the desired shape. A different mechanism, ⟨*boundarychar*⟩, was added with the 8-bit update, but its application is not entirely trivial, and most users understandably prefer a more explicit solution.

Where is the origin?

TEX itself completely avoids discussing page dimensions. The imaging software assumes `\hoffset=0pt`, `\voffset=0pt`, extending downward. This is opposite from what is assumed for traditional printing, where the origin is at the bottom left and extends upward. The printing devices available when TeX was developed were (and most devices today still are) unable to print at the absolute edge of the output medium; this was often blocked by the gripping mechanism, and although it was only a small fraction of an inch, it might not have been the same for all devices, so the origin couldn't be set at zero. Instead, an easy to remember value for a position inside of this limit was assigned: `1in,1in` from the upper left corner, a setting that is a source of consternation for the myriad TeX users located in regions using the metric system.

What is the origin of this one-inch origin?

In the U.S., paper was usually assumed to be lettersize, 8.5 by 11 inches, and a common size for the text block was `\hsize=6.5in` by `\vsize=8.9in`, which results in one-inch side and top margins and a 1.1 inch bottom margin (into which a page number is often dropped) when a full page is centered. This value was selected for the fixed origin, in order to provide a consistent value for the software. David Fuchs, developer of the earliest output device driver, was (to the best of my memory) the person who established the value.

The inherent ability of laser printers to print on different paper sizes is limited by driver support, and nonstandard dimensions are blocked or lost. `pdftex` introduced primitives for page dimensions and several different page boxes required by PDF, but even when laser printers became generally available, the 1-inch value was retained for the sake of backward compatibility.

One last thing to think about

When TeX was created, the only way to read a TeX document was on paper or from the source file. But since then, considerable software has emerged for PDF text analysis, optical recognition from scanned text, etc. In order to reliably locate and recover raw text from such "final" electronic documents, it's necessary to be able to disambiguate different columns on a page, recognize page numbers, and record similar identifying information. TeX contains nothing to make such recognition easy, or even in some cases possible. It was undoubtedly premature to think of such details in 1980, but they should be considered for the future.

Acknowledgments

Several people who were there at the beginning, or at least have been involved with TeX for a very long time, have contributed their knowledge to this effort, both by providing missing information and by otherwise keeping me honest.

Chief among these contributors is David Fuchs. Nelson Beebe added points that were overlooked in the limited environment of 1980, but which could have been implemented within those limitations had they been foreseen. Karl Berry, as ever, politely called attention to my logical inconsistencies. Phil Taylor commented from a British point of view, helping to tidy up bits that might not be understood the same way on opposite sides of the pond.

Finally, Don Knuth was asked to read and comment. (If Don hadn't created TeX, there would have been no reason for this essay.) His comment was: "All these things and more will be fixed in iTeX* as soon as the implementation team is ready." [7]

References

[1] B. Beeton. Mathematical symbols and Cyrillic fonts ready for distribution (revised). *TUGboat* 6(3):124–126, 1985. `tug.org/TUGboat/tb06-3/tb13beetcyr.pdf`

[2] Extended TeX font encoding scheme — Latin. *TUGboat* 11(4):516, 1990. `tug.org/TUGboat/tb11-4/tb30ferguson.pdf`

[3] Y. Haralambous, J. Plaice. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO 10646/UNICODE, etc. *TUGboat* 15(3):344–353, 1994. `tug.org/TUGboat/tb15-3/tb44haralambous-omega.pdf`

[4] D.E. Knuth. Mathematical typography. *Bull. Amer. Math. Soc* 1(2):337–372, March 1979. `https://www.ams.org/journals/bull/1979-01-02/S0273-0979-1979-14598-1/S0273-0979-1979-14598-1.pdf`

[5] D.E. Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1984. Volume A of *Computers & Typesetting*.

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

[7] D.E. Knuth. An earthshaking announcement. *TUGboat* 31(2):121–124, 2010. `tug.org/TUGboat/tb31-2/tb98knut.pdf`

[8] S. Matteson. The road to Noto. *TUGboat* 41(2):145–154, 2020. `tug.org/TUGboat/tb41-2/tb128matteson-noto.pdf`

◇ Barbara Beeton
`https://tug.org/TUGboat`
`tugboat (at) tug dot org`

* [A bell rings at this point.]

Typographers' Inn

Peter Flynn

Page numbering revisited

In my last column [2] I mentioned retrofitting page numbers from the PDF back into a web version of a document, and said it was relatively trivial with TeX. Daniel Nemenyi of KCL emailed me to ask how, so I had to dig out the code and see.

I know others have done this, but I don't know if anyone has documented it anywhere. What we implemented was for a client using XML, generating one transformation to XeLaTeX for creating PDF, and another to HTML for their in-house web site. The code is not proprietary but I can't extract it directly without exposing a lot of their in-house naming, so I rewrote a short version for Daniel.

The implementation was done using the fwlw package (which makes catchwords available for each page: the first and last words on the page plus the first word of the next page). With this, we modified the `\pagestyle` provided to typeset the catchwords at the bottom of the page, in white, so they were not visible, and separated them by an otherwise unused delimiter so we could extract them reliably: we used the ASCII decimal 172 (0xAC) character (¬) or NOT symbol (see Figure 1).

```
\documentclass{article}
\usepackage{lipsum,fwlw}
\usepackage{xcolor}
\makeatletter
\def\ps@pagerange{\let\@mkboth\@gobbletwo
 \let\@oddhead\@empty\let\@evenhead\@oddhead
 \def\@oddfoot{\rlap{\color{white}%
     Page=\thepage¬First=\usebox\FirstWordBox¬%
     Last=\usebox\LastWordBox¬%
     Next=\usebox\NextWordBox¬}%
       \hfil\thepage\hfil}%
 \let\@evenfoot\@empty
 \let\chaptermark\@gobble
 \let\sectionmark\@gobble
 \let\subsectionmark\@gobble
}
\makeatother
\pagestyle{pagerange}
\begin{document}
\lipsum[1-100]
\end{document}
```

Figure 1: Minimum worked example to expose catchwords for retrieval.

For the extraction we used *pdftotext*, a freely-available utility which creates a plaintext version of a PDF document. In this, page-breaks are signalled with an ASCII decimal 12 (0x0C) character, which is the Control-L or FF (FormFeed). In this example, the few lines immediately above each of the page-breaks contains the page number preceded by the delimited string we defined in `\ps@pagerange` in Figure 1.

In Figure 2 you can see two fragments of the output, the first from page 1 and the second from page 16 showing a problem where the page number occasionally gets imbrangled in the 'Next' catchword. This has not been resolved.

```
lorem lorem, interdum eu, tincidunt sit amet,
Page=1¬First=¬Last=amet,¬Next=laoreet¬
1

^Llaoreet vitae, arcu. Aenean faucibus pede eu
ante, Praesent enim elit, rutrum
```

```
tempus magna. Aliquam ut purus. Proin tellus.
Page=16¬First=amet,¬Last=tellus.¬Next=
16

Vestibulum¬

^LVestibulum ante ipsum primis in faucibus
```

Figure 2: Text fragments of output from Figure 1 at pages 1 and 16.

Now that the data is plaintext, you can use the standard *grep* and *awk* text utilities (or Perl, or Python, or Lua, or whatever is your favourite scripting language *du jour*) to pull out the lines with the delimited page number, first, last, and next words. You can then programmatically step through each page number and locate the span of text delimited by the First and Last words, using the Next word as a cross-check.

The tricky bit is application-dependent: you then need to be able to reliably read your source text programmatically, find the first word on a page, scan forward to the last, check the following word is the next value, and then do whatever is needed to insert the page number at whatever point is appropriate for your document.[1]

In the case in point, the production text was stored as XML, so the delimiters they used for the line of data embedded in the PDF were actually `<` and `>` characters, so the extracted fragments were already XML. That way the lines extracted from the text file were used in XSLT to identify each location in the XML source, push the page numbers into

[1] Daniel did suggest it might be more tractable to write the page-break data to a separate external file rather than embedding it: I'd be interested to hear from anyone implementing this.

a Processing Instruction, and cyclically check the accuracy each time the file was processed. If the source is LaTeX, it might be more problematic to process.

It's not 100%, of course: it will be thrown by figures, tables, and math occurring at a page boundary, which our client didn't use. But the small number of corrections beats doing it by hand.

Type 1 (PostScript) fonts

Some of you may by now have seen Adobe's announcement[2] that it will end support for Type 1 (PostScript) fonts on 31st January 2023 in all its products (e.g. *InDesign*, *Illustrator*, *Photoshop*, etc.).

If you open a document containing unembedded Type 1 fonts with an Adobe product after that date, the fonts will not be recognised, and will be classed as 'Missing' even if you have the font files installed in your operating system. In addition, your installed Type 1 fonts will no longer appear in the Fonts menu and there will be no way to use them in Adobe's software.

However, PostScript and PDF documents with embedded Type 1 fonts will continue to display as normal, so they will still be readable with Acrobat Reader, but they will not be editable and will not work in other Adobe products.

So what's this all about? To be fair, Type 1 fonts are Adobe's invention (back in 1984), so they can call the shots. When PostScript printers arrived, they came with the built-in Adobe '35' popular fonts[3] that have dogged DTP ever since (a much wider choice was distributed later). Those 35 fonts became so ingrained that software producing PostScript (and later, PDF) only needed to reference the font by name, with no need to embed it in the document, because it could be guaranteed to be available on all printers via drivers like *Ghostscript*. It's also why so many packages that create formatted output, like statistical and numerical analysis programs, generate PostScript and PDF output without the need to embed the fonts. Plus they were seen as 'free' — in a world where font piracy is rampant, many users became accustomed to the idea that [these] fonts 'just came with' every operating system and software suite.

But the world has moved on since then, and font technology has advanced hugely. TeX has moved on too, from providing only Computer Modern and other METAFONT fonts (Type 3), to updated Type 1 versions as well as the Adobe '35' and other Type 1 fonts generously donated to the cause. You could buy or download additional Type 1 fonts and install them for use with TeX. And now we can use any OpenType or TrueType font via XeTeX, LuaTeX, and friends, including the TeX Gyre fonts (open source equivalents of the Adobe '35').

Does it matter to us? Well, yes ... some. We need to be aware that in the long term, Type 1 is going to become a dead end. For now, if you have old PostScript or PDF documents for which no source is available, they will continue to display. If you want to continue generating PostScript or PDF documents using the Type 1 fonts that come with TeX distributions, or others you have bought or downloaded, feel free to do so: your output documents will continue to be displayable. TeX itself is unaffected, and so far as I have been able to find out, neither is software from other suppliers, so you can continue using Type 1 fonts in TeX, and in many other non-Adobe systems.

So what's to do? The easy answer is, switch to XeTeX or XeLaTeX or LuaTeX or one of the other variants that support OpenType or TrueType fonts. I made that switch a couple of years ago and have not regretted it [1]. But there are still a lot of LaTeX packages that depend on PostScript fonts or graphics for other reasons, and if you use them, you may need to stick with *pdflatex* for a while yet.

Afterthought

I couldn't trace the quotation 'There is not in existence a page with a rule on it that cannot be instantly and obviously improved by taking the rule out.' [3] but Karl Berry pointed me at *The TeXbook* (end of Chapter 21). I should have looked there first!

References

[1] P. Flynn. Typographers' Inn — XeLaTeX. *TUGboat* 37(3), Sep 2016. `tug.org/TUGboat/tb37-3/tb117inn.pdf`

[2] P. Flynn. Typographers' Inn — To print or not to print. *TUGboat* 41(3), Dec 2020. `tug.org/TUGboat/tb41-3/tb129inn.pdf`

[3] G.B. Shaw. On Modern Typography. *The Dolphin: A Journal of the Making of Books* 4(1):80–81, Fall 1940.

◇ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
`peter (at) silmaril dot ie`
`blogs.silmaril.ie/peter`

[2] `https://helpx.adobe.com/ie/fonts/kb/postscript-type-1-fonts-end-of-support.html`

[3] These are: Avant Garde, Bookman, `Courier`, Helvetica, New Century Schoolbook, Palatino, Σψμβολ (Symbol), Times New Roman, *Zapf Chancery*, and Zapf ✤✻■✳✪❁▼▲ (Dingbats).

Interview with Amelia Hugill-Fontanel

David Walden

Amelia Hugill-Fontanel is the Associate Curator of the Cary Graphic Arts Collection at the Rochester Institute of Technology. Many TEX people first "met" her through her TUG 2020 conference video presentation on the evolution of type specimen books (`youtu.be/7Cm2AcQiUuk`).

The interview took place via Zoom between Walden in his home in East Sandwich, Massachusetts, and Hugill-Fontanel in her home in Victor, New York.

David Walden, interviewer: Please tell me about your youth.

Amelia Hugill-Fontanel, interviewee: I am from Trenton, New Jersey. I was born in 1975, and I grew up around Trenton. My parents divorced when I was quite young. So, I'm very close to my mom, who raised me and we lived in Trenton until about 1990. She got remarried and her husband was from the Rochester, New York, area. So we moved up to a suburb of Rochester. I always thought I would go back to the downstate area to be nearer to my mom's side of the family, but I always found a wonderful place in Rochester, including in college. So I have now lived the bulk of my life in the Rochester area.

D: Was art always a part of your life as a child or did that come to you later, say in high school?

A: There are these great Dr. Seuss books. One was called *My Book About Me* and it was a book that you were actually encouraged to fill in. It asked you questions. It had pictures like "Tell us who are the people in your neighborhood?" and you'd go around and interview people and write their names in the blanks. There was one page that was "What do you want to be when you grow up?", and then there were pictures of all these different professions like nurse and doctor and mailman and secretary, teacher, all these things, and then there was a write-in and you were supposed to circle who you wanted to be and I wrote — I came across this book not too long ago — and I wrote in "artist". So I was about eight years old.

I never realized that it went that deep, but I've always been drawing pictures ever since I was a kid, and my mom was very encouraging — she took me to a few extracurricular art classes. And then I think when I got to high school, I realized that even though I was academically strong in different fields, I always gravitated towards art — different kinds of clubs, like doing different kinds of set design for the theater. I wasn't interested in being on stage. I wanted to draw the backdrops or design the T-shirt for the environmental club.

So, I guess I always gravitated towards art. I remember when I went to college — I went to SUNY New Paltz, which is a state university in New York — and the registration person asked me "What do you want to major in?", and I had never thought of it before. Nowadays I think students are really keyed in to what they want to major, but I just said "Art", and they put me down for studio art.

D: Did you have a good, or encouraging art teacher in high school?

A: Yes. I went to a suburban high school in Rochester, Fairport High School, and they had great art programs. One of my favorites was an artist-in-residence program. So, it wasn't just the art teacher who was monitoring, but she brought in several working artists from the community who made their livings from doing different kinds of art. One was an airbrush artist. Another was a jeweler, another a cartoonist. And so, there were different blocks of classroom assignments where you would work with these artists to create work in different media that the high school teachers themselves didn't have experience with. This was great exposure to seeing how people negotiated a career in different media. It was very good.

D: Did you have other high school activities, music, sports? You mentioned drama.

A: Well, like the kids in the back, like the stagehands and stuff like that. I did not enjoy dancing and singing. They were usually musicals.

The other big formative experience in high school was that I was good at the French language, and so I did an exchange program. We had an exchange student come and stay with us, my family. I was able to go to France in my junior year in high school. Ever since freshman year, I have always loved French, and I'm still fluent in French. I took it all through college and I've traveled there extensively.

D: Where did you go for your exchange program in France?

A: We went to Normandy. It was a small town on the seaboard near Caen, which is one of the cities that's up north in France. subsequently, I married a Frenchman.

D: Let's move on to your postsecondary education. You mentioned SUNY New Paltz. From looking at your CV you weren't there very long.

A: I went for a semester to SUNY New Paltz and discovered it wasn't quite for me at that time in my life. It was downstate. I should have — hindsight is 20/20 — and I should have given it a better chance but my 18-year old self went back home after one semester and I went to a community college, Monroe Community College, in Rochester and I thought that was going to be just one semester until I figured things out. As it turned out, I stayed there and got my associate's degree. It was a really good place, an excellent community college. They had great studio art classes, good instructors. After that, I transferred into Nazareth College.

I work in academia now, where the places where you got your degrees or fellowships become important, but I can't say better things about my community college experience. If I hadn't gone there, I wouldn't have been exposed to a slice of our population in Rochester that I probably would have never experienced at Nazareth College, which is a small liberal arts college. I got to meet mothers who were trying to go part time and get their degree. I got to meet people who served in the military and were working at college as part of their GI Bill benefits, all different people, immigrants — there was a woman from China who had married an American man and she was trying to get better at English. It was a great experience for me.

D: I can empathize. I went to UC Berkeley for one term, tried to study architecture, discovered I wasn't going to be good at that. Like you, I went back home and went to junior college for another year and a half so that I could get qualified to go to the four-year college.

A: Perfect. You kind of grow up just a little bit in that spot.

D: Yes.

A: I worked hard. I worked a lot of different jobs. I had a job as a cashier at a hardware store. I worked in the bookstore. I worked in the counseling office at the community college. So, I did a lot of work — work in addition to trying to go to school.

D: Your CV says you got a BA from Nazareth College. While you were there, were you already interning at Eastman House?

A: Yes. I had originally gone into … I can't remember now what the degree was, the associate degree, but it was mostly studio art classes. But as part of that, we had an art history class at the community college and I fell in love with art history. For everybody else in the entire auditorium: the lights go off, the slide projector goes up, and everybody is sleeping. But I was rapt, learning about ancient Greece and different kinds of cathedrals. I was completely in love with art history and the professor was great too. So, even though I still did a few studio classes at Nazareth College, I focused more on art history.

I came out with an art history degree, with a minor in studio art, because art history was just transformative for me. It was a very small college and there were only a few professors who were teaching art history, but I took every single one of them because I liked it so much. I realized that I got more enjoyment from studying the history of art and the causes, like the cultural causes, and writing about art than I did from studio art. I couldn't see myself making a living from making things, making art; having to support myself on that would be stressful for me.

D: How did that lead to Eastman House?

A: One of the classes I took was something like contemporary art criticism, and the woman who was the professor — her name was Judy Natal — was a photographer. She was a really interesting photographer in her own personal practice, where she did photography and a special process called photogravure, which is a wonderful printing process. She would transfer photographs to marble and sculpture. She was a collected artist at Eastman Museum, and knew the curator there. She recommended me for an internship. I wound up staying there more than a semester and, wonderfully, got hired part-time right after my bachelor's degree to work on a cataloguing project in their photography collection.

I was still working other jobs at the same time as Eastman House because I couldn't support myself working only in the museum. I remember that at one of those part-time jobs somebody asked me if I wanted to go full-time; and I said "No, I need to stick with Eastman as long as I can and see what I can do to get in there." Eventually, the curator of photography hired me as her full-time assistant curator.

D: Nice … some excellent training for the future.

A: Yes.

D: How did you end up going to RIT, and were you already doing printing and that sort of thing, or just art and cataloging?

A: Those transformative moments like the time at Monroe Community College when I took the art history class and knew I loved that ... well, another one of those times was at Nazareth in my senior year. It was spring, and I was to be out of there in a couple months. I took a digital art class. This was in 1997. I think we were running a bunch of different software, Photoshop 1.0, QuarkXPress 2, vector-based Adobe Illustrator, and others. It was a class to learn those software elements and learn just a little bit about graphic design.

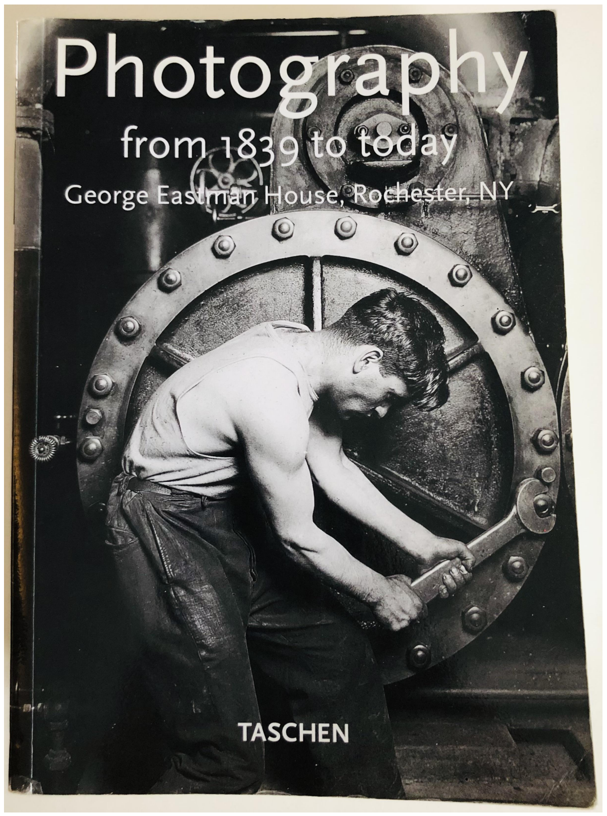
I used those skills at Eastman Museum to prepare exhibitions. Hardly anyone else, just a very few people, knew how to use QuarkXPress or knew how to use Photoshop; and one of those people was the graphic designer at the museum and the publication staff. Because I had those skills, I could be a sort of liaison between the graphic designers who were designing books and the curatorial staff who were doing the writing and actually selecting photos. I began to see that it was going to be a long time and a lot more education if I wanted to be a curator at a museum, and I didn't think that was for me at that time.

So, I kind of threw in my cards: "I'd like to do some graphic design or I'd like to study about art book publication," because at Eastman Museum, it seemed like most of our work was focused on "Let's get the photographs ready for the exhibition and write the labels and create the narrative of an exhibition." But along with that, you have to do all the marketing of it. You have to photograph those images and make sure they print correctly on the poster and the book and the invitation. It seemed like we were putting as much mental energy into the reproduction of these museum artifacts, and I thought that was really interesting.

Luckily RIT had a program called Graphic Arts Publishing. In it you would learn not only the different software page layout programs but also the reproduction processes and the typographic history about graphic design and layout, and it was perfect for me. So, that's how I got into that.

D: At Eastman, did you also do catalogues for the exhibits?

A: Yes, sometimes. I think the tipping point was when we worked with the art book publisher Taschen. It was the 50th anniversary of the museum. This is the book, and I was the lowest on the totem pole. Just about every single page has images upon images.



It was my job to get the physical photographs out of the archive and take them to the photographer to make four by five-inch transparencies, color transparencies, and then when we got the proofs back from the actual printer, we would compare the proofs with the transparencies. It was this amazing on-the-job learning. That whole process was formalized in my studies at RIT — that kind of art book reproduction. It wasn't just "Oh, we're just going to do this one-off." I wanted to be part of more productions like this.

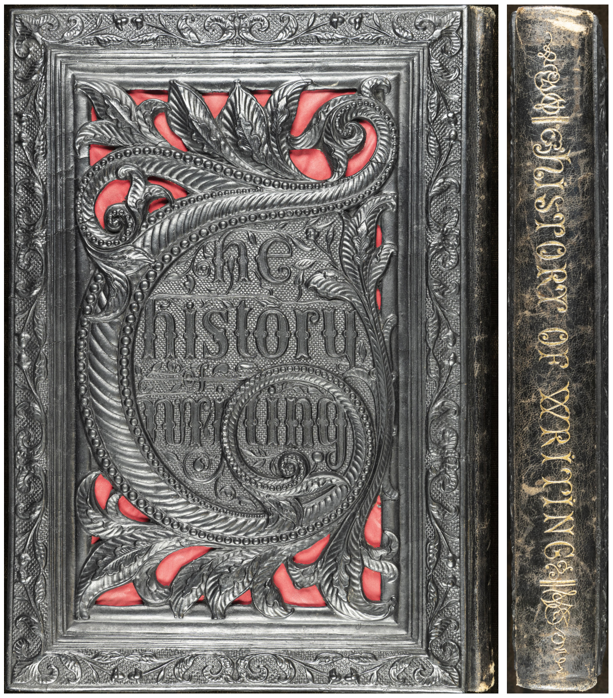
D: Wonderful. You got an MS at Rochester. How did you go from being a student to an employee?

A: All these happy accidents, I think. Hold on. I'm just checking for my dog.

D: Are you, in fact, at RIT or are you at your house?

A: I'm at my house. We're working two days on campus, three days remote.

So, the question was, how did I stay at RIT? Well, great opportunities, being the right person at the right time, I guess. As a graduate student, I applied for a position as a graduate assistant at the Cary Collection because I had been there on a tour before I matriculated as a graduate student and I thought then, "I would love to work here," and I think I presented myself as someone who had



Henry Noel Humphreys, *The Origin and Progress of the Art of Writing*..., London: Day and Son, 1855. An example of a fragile 19th century papier-mâché binding from the Bernard Middleton Collection of books on the History and Practice of Bookbinding.

museum experience, which would be useful to the Cary Collection.

The curator then, David Pankow, hired me. I have been an employee of the Cary since 1999, when I finally matriculated as a grad student. David had planned two big conferences. I don't know why he planned two conferences in a single year, the year 2000. I was the graduate assistant, and worked hard on those conferences to help pull them off. One was called Bookbinding 2000, and it was to mark the acquisition of Bernard Middleton's book binding collection which we still have. It's a real jewel in the collection. A lot of people came from all over the world for that conference. The second was the American Printing History Association's conference called Printing on the Digital Brink. David saw how hard I worked, I think.

In the middle of this, David had always expressed that he had wanted to start a university press at RIT because, looking at the programs, there's a wonderful program in photography; there was a wonderful one in printing; there was another one in graphic design. There really shouldn't be any reason why with all these resources and talents that the university shouldn't have a publishing program, a formal one. So, he was able to do a proof of concept. He was given permission by the Vice President of Finance for two years to publish a few books. I was hired as the production editor right out of grad school to be the person to put those books into production and to publish them. So, what is now called RIT Press has been around since 2001, when I was first hired. It was originally called RIT Cary Graphic Arts Press because we focused on graphic arts publications that directly related to the Cary Collection subject matter.

D: As the production editor, did you help with the direction of the press or was it more you were doing the mechanics and operations of the press?

A: It was such a small endeavor that it was a Jane-of-all-trades situation. It was David Pankow, who was the Director, who would choose most of the content, but in terms of liaising with the authors and the designers, getting print quotes and even shipping, that's what I did. Luckily, I had student assistants to help.

In those early days, up until about 2004, we only had one and a half full-time employees to run this endeavor, and my director was the half time because he was still running the Cary Collection too. I learned so much — about how to do everything related to the publication production, in terms of filing for ISBNs and cataloguing data, how to work with copy editors and authors and negotiate printing costs.

It was also an exciting time, because a lot of the new developments in digital printing of books were coming along in those early 2000s. We were able to do some innovative production that was on the edge of how print-on-demand took off — trying to push the envelope and see if we could get truly good quality in terms of color digital images, printing compared with traditional offset lithography.

D: Were you also helping with the marketing?

A: Yes. Sure. We would do like a prospectus sheet. We would do mailings. I think I wrote most of the original website.

D: Were you using some kind of website generation program or did you write HTML in a plain text editor?

A: Part of the allure of this university press is that some of the services could be provided by the library. So, we did have a web designer who was a library web designer, but then he taught me how to load new products on to the site and so, I was doing plain text editing of HTML to get those images and items up and uploading new PDFs every time the order form changed. We didn't have any kind of online ordering at that time. You had to call and give me your credit card or send in a check by mail.

D: You were the production editor for how long?

A: Until 2008.

I had my first son in 2004 and I had my second son in 2008 and I realized that it was too much. It was too much to be a mom and to work full time. It's a credit to my former boss, David Pankow, that when I said "I can't continue on this way with this new baby on the way. I need to quit," and he said "Hold it. Hold it. Let's find another solution." He very kindly did a search for my replacement as production editor, which really needed to be a full-time position, and I became more of just a financial manager, business manager, which was a part-time position. So, while my second son was a baby, for about a year, I continued in that way, which was wonderful. It was great that he made that happen.

D: When your son got older, did you became even more part-time or less part-time?

A: More full-time. In 2009 the library gave David a new part-time assistant curator position; and he asked me to be in that position.

D: When did you start being on the adjunct faculty and teaching courses?

A: Let's see … I think that was about 2003.

D: While you were still with the press?

A: Yes. That was great — a way to apply what I had learned in terms of production and design and software. I was teaching photography students essentially how to learn these software tools and present their work. I remember a lot of arguments. They're photo students. They love Photoshop, and I'd say "Okay, I need your resume," and I'd show them how to use Adobe InDesign, and they'd argue with me: "Why use this? We know Photoshop. Photoshop can handle text. We're going to submit it in this," and I'd say "Okay, you do that."

D: That's not good. Photoshop doesn't handle text well.

A: Right.

I'd get them and I'd mark them up because of course, if you're a 19-year old student, there are probably going to be a couple edits that need to happen in your resume and they'd say "Now, I have to go back in Photoshop and change everything," and I'd say "Do it in InDesign," and they would. So, that kind of nice conversion, this convincing them it's okay to use the right tool for the project at hand. You need to use vector-based software for some things and you need to use bitmap-based software for others.

D: You were an adjunct professor and then you had a curatorial position and that's basically what you're doing today.

A: Yes. Over the course of time, now it's associate curator. You know how these academic things work. But what's very nice is that I'm still a teaching adjunct, except for this year in the pandemic. Now I teach a letterpress printing class in the School of Art. It's just fantastic. I get not only art students — it's open enrollment with no prerequisites. I've had engineers and software programming students take my class.

D: Is your experience "drifting" — I don't mean that in a pejorative way — from starting college to incrementally building this career useful experience in guiding students in what they might do? Do they come to you for such guidance?

A: I do write a lot of recommendations for students. Usually they are students who are interested in museum-based jobs or librarian-based jobs, going for library school or library science positions.

I do think that the drift or the kind of haphazard — the not-straight line — path is a good model. There are very few people who can be absolutely assured that what they start on when they're eighteen years old is going to be where they're going to end up when they're forty-five. We have a lot of undergraduate students who finish and then are kind of in a quandary about what to do next, and I say, "You need to just work for a year before you go right into grad school." That's what I did. I worked for about two and a half years, and it gave me a really better viewpoint of what adult life is and where I want to put myself in that adult life. I think it's okay that I moved around.

D: My view is that it's never too late to figure out what you should be doing. New things will come up.

A: Yes. I love working on book production projects. I really like graphic design, and I like to write, and I like the process of putting a book together. And I still do it all the time in my job as a curator, even though I don't do it for the ultimate goal of selling a book. I prepare articles or I contribute to books. I even print things as part of our programming at RIT and the Cary Collection. So it never got away from me. But I think that intense view of printing something that is a thousand copies offset really helps to plan even a small job that's on a letterpress machine.

D: Do you teach offset or roto?

A: At the School of Printing Management and Sciences, where I got my master's degree, they did teach that. However, that school has undergone a lot of change over the last twenty years. Part of it is that it's hard to convince people to spend a lot of money on a private school education to become a printer. Printing kind of suffers from being perceived as a blue collar field. I don't think it is. I think it's fascinating to be a printer.

The School of Printing was very well-attended for many, many years, but when a lot of printing went offshore because it became too expensive to produce many books and various printed articles in America, the students went away. So the School of Printing has become a lot smaller at RIT, but it's also transformed. They are now aligned with the Packaging Science Department, because we all know that packaging is the most ubiquitous printing. We see it all the time in the grocery store. And there are fewer students, but they all do well in the marketplace because they're cross-trained in actual production and in programming in terms of web media and different kinds of graphic reproduction processes. So there's a lot more concentration on digital scanning and analysis of digital imaging.

You asked me did they still teach offset and roto. They teach offset. We used to have a gravure press at RIT, but they don't teach rotogravure anymore. We have a flexographic press. They teach offset lithography, screen printing, and a huge gamut of digital printing.

D: You mentioned food packaging. I want to digress a moment.

A: Sure.

D: I worked my way through college, four summers, in a big printing plant which had a seventy-six-inch (I think) Miehle four-color offset press. I was a helper. I re-piled paper to go into the feeder and re-piled printed sheets coming out and took them to a place in the warehouse for the ink to dry. Another summer I worked in the cutting shop next door where they stamped a big sheet into, for instance, eight different food-package boxes. I had a pneumatic chisel for peeling apart the individual six-pack-of-beer cartons which could then be glued next door in the gluing plant. It was very interesting.

A: Amazing. I love that.

D: I loved it, and I thought, "If this college stuff doesn't work out, I'm going to spend my twenty years here and become the lead pressman. It's a fascinating craft."

A: That's so interesting. In the past, the students at the RIT School of Printing got jobs as plant managers and scanning managers. RIT's School of Printing is an interesting story. It's been around for almost a hundred years, and it originally started as a trade school, so it wasn't always a degree-granting program. Around the fifties and sixties, they started doing degrees. There was always work for those School of Printing grads, because part of the program was funded by the Gannett Corporation. Gannett's headquarters are here in Rochester, so an RIT School of Printing grad was assured that they could get a job at any Gannett newspaper around the country.

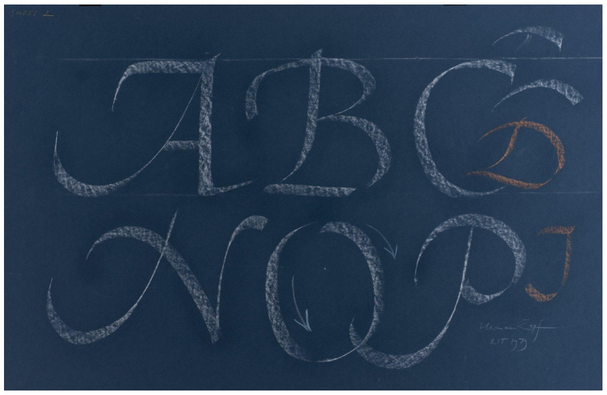
It's very much changed now to be software-based, multimedia-based. And that's okay. A lot of people have hard feelings about that, but I embrace that change because I think still they're teaching some great things there, even if they no longer get to do all the physical printing. I got to, as a student, be on a web press and take the web press class where you'd watch the splice, and it was terrifying but exhilarating and fascinating. I love that.

D: Since this is an interview for the TEX Users Group (`tug.org`), let me ask you a little bit about TEX at this point. Did you know about TEX and LATEX before you participated in TUG 2020?

A: Not really. I had heard of it from our mutual friends, Chuck Bigelow and Kris Holmes of Lucida fame (`lucidafonts.com`). But when we were preparing for the TUG conference that didn't happen in summer 2020 in Rochester, I had reached out to colleagues who were in the Program of Imaging Science and found out that they teach their students — there's undergrad, grad, and doctoral students — everything is run in LATEX. They focus on that. I think it's a quiet thing that is not accentuated, but when you start probing deep into a program you find out, "Oh, it really is used widely," especially in their program.

D: Kris and Chuck have been well-known in the TEX world ever since they were out there helping Knuth start TEX back in, I guess, the late seventies. What connection do you have with them at RIT?

A: I don't remember what year it was. But Professor Bigelow, Chuck Bigelow, was hired as the Melbert B. Cary Professor at the School of Printing at RIT. He and Kris moved to the Rochester area, and that was our first introduction. That professorship has always worked very closely with the Cary Collection, the Cary professorship — we both get our endowments from the same estate of Melbert B. Cary, Jr., and his wife, Mary Flagler Cary. And we're both dedicated to printing history and typography, etc.


Hermann Zapf, Calligraphic teaching sheet from his summer classes at RIT, 1979. Demonstration of stroke sequence, stress angles, and forms of calligraphic letters. Chalk on blue paper. One of many Zapf items in the Cary Collection.

So Professor Bigelow would bring his classes to Cary and be such a wonderful collaborative partner. Kris Holmes was also teaching calligraphy at RIT. We have an extensive calligraphy collection at the Cary, so she would bring her students for different kinds of guest visits so that they could see original calligraphy, especially from Hermann Zapf who, of course, was the instructor of both Chuck and Kris at RIT in the 1970s. They took Professor Zapf's class in calligraphy.

So that's how I got to know them and know them well. Then, in 2010, the Cary Collection, Professor Bigelow, and a couple other departments on campus put on The Future of Reading conference. Kris was a speaker and Chuck was part of the planning committee. Ever since then we've been such lovely colleagues and good friends.

D: Zapf, of course, was another person who helped Knuth create TEX. He helped Knuth get his type designs right.

More generally, what do you see as the value to RIT or to Cary or yourself of having the sequence of Alexander Lawson, Zapf, Frank Romano, Bigelow and the other Cary Professors?

A: It's always been this opportunity for a renowned practitioner in publishing or typography or graphic arts to be a professor at RIT and influence the students.

The way we operate in terms of acquisitions and programming — in my position at the Cary Collection, if a book comes up for possible acquisition, the first question we ask is, "How are we going to teach with it? How are the students going to learn from it?" It's a very student-centered way of acquiring materials for educational purposes. I think having the Cary Professor in that position at RIT is in the same spirit. How can students benefit from having such a prolific or successful practitioner be a leader and a model for them in their classes? That's how I would view the Cary Professorship.

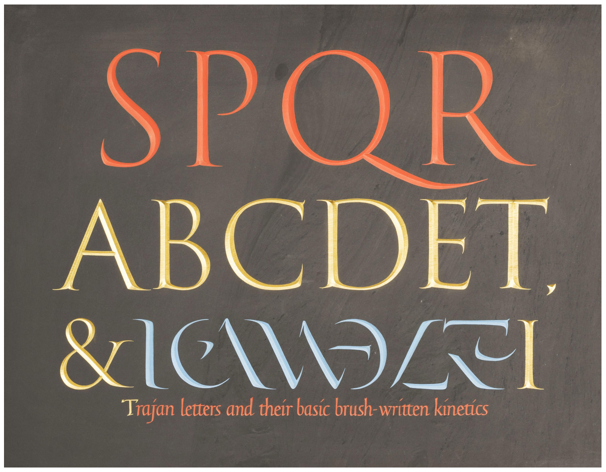
D: I want to cover a bit more about the Cary Collection (`tug.org/TUGboat/tb39-3/tb123walden-cary.pdf`) and this may be a good time. Might you recount the story of Lawson's involvement in the creation of the Cary?

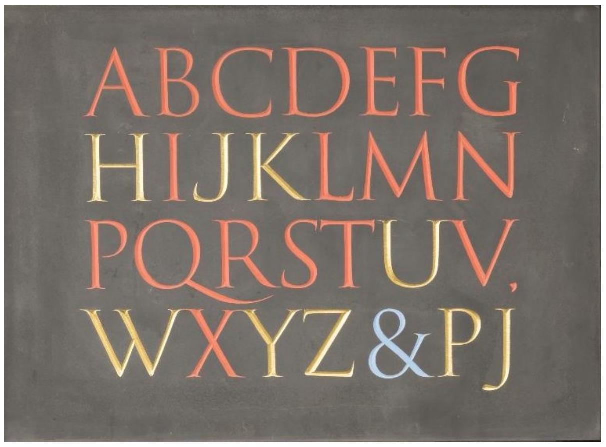
A: Oh, sure. It's a great story.

The Cary Collection just celebrated its fiftieth anniversary in 2019. So 1969 is when the collection was officially deposited at RIT. But the great story is that, a few years previous to that, the estate of Mary Flagler Cary was announced that there was going to be fifty million dollars available for New York State educational purposes. One of the RIT School of Printing alums, Herbert Johnson, who later became one of the Cary Professors — he was a book designer for Alfred Knopf — saw that announcement in the *New York Times*, that this Cary Estate was going to come up and interested parties should apply for grants from this foundation. Johnson sent that notice to Alexander Lawson, who was his former professor in typography at the RIT School of Printing.

Johnson wrote to Lawson because RIT, in the early sixties, had acquired a collection related to Frederic Goudy called the Coggeshall Memorial Workshop. Coggeshall was a friend of Frederic Goudy who had acquired a lot of his type and was planning to write a biography, which unfortunately never got done. So this Coggeshall/Goudy Collection was already at RIT, and using that collection, Professor Lawson taught many students, including Herbert Johnson. Goudy also had an association with Melbert Cary, who was the husband of Mary Flagler Cary. So, an RIT alum from the School of Printing gets notice of this foundation, sends it to his former professor, and then Lawson jumps on this. He talks to one of the development vice presidents at RIT and says, "Maybe we should make a proposal for some of this foundation's money, because we do have a Cary association in the School of Printing."

I have talked to some of the former trustees of the Cary Foundation. One of them said that they did a site visit to RIT and they were very impressed with how the students who worked at the library were so courteous and kind to them and directed them appropriately. So that gives me pride, as a library employee, that the library was looked upon favorably for what would be a really substantial





Left: Trajan letters and their basic brush written kinetics. Right: Trajan alphabet. Both by Father Edward Catich.

gift from this foundation of funds to establish the Cary Graphic Arts Collection. The library acquired Melbert B. Cary's book collection, which is about 2,500 books, but then also an endowment for the Cary Professorship, and then another small endowment for the Goudy Award, which is the annual award on typography (`en.wikipedia.org/wiki/Frederic_W._Goudy_Award`).

That's the long meandering way the collection came to RIT. I like how it involves an alum, a current professor, and this association that Melbert Cary had with one of our substantial holdings, namely materials about Frederic W. Goudy, the type designer. One thing I always mention about this in tours with visitors is that I wish I could say that Melbert B. Cary had made his millions and millions of dollars from type founding; he was an importer of metal type from Europe to the United States. He died in 1941, and he did a lot to incorporate novel and interesting avant-garde typefaces in the American market from different foundries in Europe. But Cary didn't make his money there; he was married to an heiress. Her last name was Flagler.

D: Flagler . . . like the places in Florida.

A: Yes, exactly. Her family was involved with oil, real estate, and railroads. If you go to St. Augustine, there's a Flagler College. There's a Flagler County in Florida. Mrs. Cary was a Flagler, and that's where that foundation's money came from. It was a very interesting foundation. It was interested in the Cary Collections which collected things about printing; that came to RIT. They also collected medieval playing cards that went to Yale University's Beinecke Library. And they also collected musical scores by famous composers, like a score written by Mozart, and that went to the Morgan Library in New York City. The Mary Flagler Cary Charitable Trust also gave a lot to music composition and philharmonia and nature conservancy. So a lot of the different wetlands preservations have some kind of Cary grants associated with them.

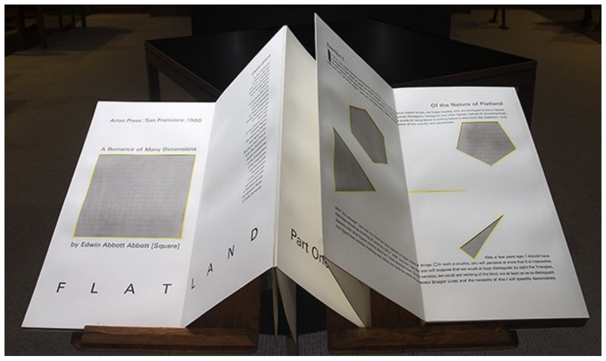
D: Another question about acquisitions of the Cary. You have a set of Father Edward Catich's alphabet stones at the Cary. How did those come to be at the Cary and what is their significance to the Cary or the college?

A: Father Catich was a priest from Iowa. He taught calligraphy there and was a scholar of the Latin alphabet. In 1976 he was invited, as the Goudy Award winner, to come to RIT to give the Goudy award lecture, and also to do classes with students. Because of that association — I'd have to look up whether he donated or if RIT bought them — he provided several significant things, including three slate-carved alphabet stones. Two of them are based on the kinetics of the Trajan Column in Rome, which is the first century inscription.

D: What do you mean by "kinetics" of the column?

A: Catich studied the ductus, or sequence, of the strokes made by hand, and then chisel, to carve the Trajan inscription. So the kinetics I believe refers to the human movements needed to make the letters.

The Trajan Column is considered to be one of the most beautiful inscriptional alphabets that was ever made, from the Roman Empire. Father Catich carved two stones that were based on the Trajan Column. There's also a third stone, of his own alphabet, which he designed, called Petrarch, which is also a Roman inscriptional style alphabet (all can be viewed at `digitalcollections.rit.edu/luna/servlet/s/b430ql`).


Arion Press · San Francisco · 1990
A Romance of Many Dimensions
by Edwin Abbott Abbott [Square]
F L A T L A N D
Part One
Of the Nature of Flatland

Left: a book from the Vincent FitzGerald / Rumi exhibition.
Right: *Flatland* by Edwin Abbott, from the Landmarks of Printing History exhibition, in collaboration with the RIT photography department.

Another marvelous artifact that we acquired from Father Catich was an original rubbing of the Trajan Column. Father Catich, because he was a priest, was able to get permissions in Rome that other people wouldn't be able to leverage. He was able to make several rubbings of the Trajan Column, on scaffoldings up there, and they're in different collections around the country.

RIT has one of them, and I teach with it every month, it seems. In a typical year, we see all the sections of students who are studying typography, and it is an essential resource that I gesture to on the wall and ask students to deeply analyze the letter forms that are evident in that rubbing. It's so essential to us, how we teach. It's so much better than a slide of it.

D: You have a long list of exhibitions you have been involved with. *TUGboat* production editor Karl Berry especially noted your exhibition called "The Light of the Sublime: The Works of Rumi". Will you please speak a little about how that exhibit came about, perhaps, as an example of how exhibits come about more generally.

A: Sure. We like to frame our exhibitions in ways that would be educational for our students, and I think the idea of that one came about because we had been collecting at the Cary — David Pankow initiated it — all the works of a publisher in New York City called Vincent FitzGerald & Company. It was kind of a livre d'artiste publications production where you would invite an artist to respond to a classic text in some way, create artworks for a book, and then other artisans would work on the typography and the illustration and the design and the bookbinding. So every single book is very different in a limited edition. They're works of art, every single one. Vincent FitzGerald was the orchestrator and publisher of this.

So we have all these wonderful books. When I started to think about this exhibition, I knew I wanted to highlight the Vincent FitzGerald books, because we had this great collection that we had already been showing students. But I wanted to respond to and physically focus on the works that he published that were translations, or sometimes bilingual publications, of the twelfth century mystic, Rumi, who was Persian. Vincent FitzGerald is dedicated to the works of Rumi, and he had a collaborator in an Iranian woman named Zahra Partovi, who was the translator of the Rumi works. So it was nice that when we started to think about this exhibition to not only show off the publisher and the visual works, but also focus in on an author that was a little bit more diverse than a typical European author — perhaps appealing in ways that the Cary Collection hadn't reached out before to the community, to offer something that wasn't so mainstream. That's how that exhibition came about.

I was proud of that exhibition because Vincent FitzGerald and Zahra Partovi are still practicing artists, and I got to visit them in New York City. They even debuted one of their final works at the Cary Collection during the exhibition opening (`rit.edu/carycollection/light-sublime`).

By the time I had started working with the material, they were more or less done with the book projects, and Zahra had turned now to music composition, so we were able to have a visual light installation and we also had a harpist who performed Zahra's original composition that was based on another quote by Rumi.

I found it exciting and a little bit terrifying, because for most of our previous exhibitions, the makers and the authors were all passed away. To work with current artists is a whole other dimension of collaboration, trying to make sure that the exhibition was appropriate for what they would like.

D: If that exhibition is representative, it seems that creating an exhibition is a dynamic process and you go wherever the materials and situation take you.

What do you see as the benefit of exhibitions either to scholarship or the university or yourself? You certainly seem to be involved in a lot of them.

A: We're such a small staff in the Cary Collection, that even if I'm not the primary curator for an exhibition, I usually act as the preparator for my colleagues who do the intellectual work and the arrangement. I have the hand skills to do a lot of matting and framing. I learned a lot of that at Eastman Museum, in fact.

I always feel that exhibitions are a springboard for a whole bunch of different creative things that could happen. You create an exhibition and you give a tour to students about it and get them excited about the things that are in the collection. Or you create an exhibition and then you can invite a lecture series about the different themes. When we commit to an exhibition, it's not just putting the things on display for a short period of time. It brings into being a place where we could talk about these concepts and include in other programming. It could also be publications that result in exhibitions.

Right now, I just had an exhibition meeting with Steve Galbraith, who's my supervisor now. We were talking about how even in this time where there are not a lot of people in our library because of the pandemic restrictions of quarantine and social distancing, we're still moving forward with exhibitions. We're going to shift in the next year and have them be more online exhibitions. That's a blessing and a curse because I'd love to have people actually look at the objects. On the other hand, the online exhibitions are usually evergreen. So, I can turn to them again and again in the future; for instance, point a class to them, or build upon them in other ways. Say I curate an exhibition about Goudy and then in two years I do an interview with somebody about Goudy; I can add that interview to that exhibition. It could grow in a way that those ephemeral or timely exhibitions can't. So, I view them as part of our outreach and part of our educational program.

D: Online exhibits are terrific for those of us who can't visit the Cary.

A: Yes. I'm a bit sad because the next exhibition that I was supposed to work on heavily was one of our graphic design archive designers. His name was George Giusti. He was a graphic artist who did a lot of editorial design, covers for magazines for example, but he also was a sculptor. We have about 50 of his sculptures, and I was envisioning this fantastic exhibit, where we're usually putting up pictures or books and prints and things, very flat things, this was going to be a sculptural exhibition. After talking with Steve just now, I think it's going to be online, at least for this next year and then hopefully in 2022 we can have a physical exhibition.

D: I hope in 2022 I can travel to the Cary again. I was looking forward to going to the Cary this past summer for the TUG conference.

A: Yes. I planned another conference in the interim here for the Hamilton Wood Type and Printing Museum in Wisconsin, and it was a joint conference with the American Printing History Association, and we had it just this last weekend, November 5th through the 8th, and even though it was online, we had more participation than we ever could have in the physical space. So, there's a little bit of a silver lining and, again, everything was recorded.[1]

D: I'll look forward to it. In a message you said you were embedded in the American Printing History Association and the Hamilton Wood Type and Printing Museum. What do you mean by embedded?

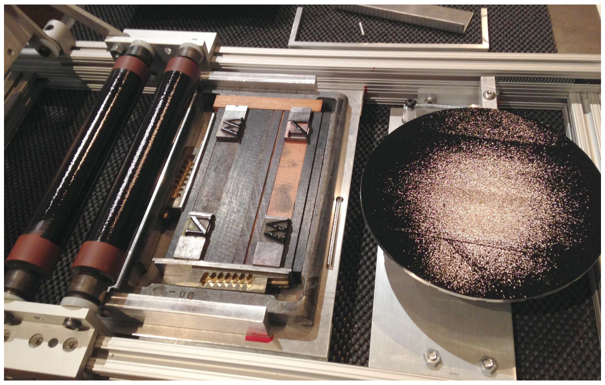
A: It seems like all my volunteering time goes there. And that's okay because I think both organizations serve the Cary Collection well too. We're all good collaborators, all interested in preserving, printing artifacts and graphic design.

D: What kind of volunteering do you do for them?

A: I'm the Vice President of Programs for the American Printing History Association (APHA); their web site is `printinghistory.org`. I organize the conferences and also a lecture series that we have every year called the Lieberman Lecture. I'm also on the board of the Hamilton Wood Type and Printing Museum (`woodtype.org`).

As a board member, we have a chance to choose what best suits us in terms of our talents, how to serve on that board, serve the museum, and so. A year ago or a year and a half ago, I said, "I want to be completely selfish and put all the people that I really care about and I find interesting in printing history in the same room. So, we should have a joint conference." So, that's how that association between the two organizations got started and we had hoped that it would be in-person, but that didn't happen, but still, I think it was a strong alliance between the two places.

[1] Videos are online at `woodtype.org/pages/wayzgoose`.



Left: 18th century-style "common press". Right: Inked type on the small aluminum press. Both built by RIT students.

D: Earlier you also mentioned "Hands-on letterpress teaching and learning across curricula". What do you mean by "learning across curricula"?

A: A good friend of mine, who was previously on the APHA board, said that book arts education is really like a foundation of liberal arts; there is something to be acquired in almost every field, from learning about how to make a book, how to write a book, how to edit a book, and by extrapolation I think you can move that on to the earliest productions of books and that's the process of letterpress printing, the oldest commercial printing process in our culture. Today, a lot of contemporary letterpress work is very art-centered, but from Gutenberg's time up through the 20th century, letterpress was used to print in every discipline. If you were looking at a math book, it was letterpress printed.

The impact that this printing process had had on every discipline of human intellectual thought is, I think, rather astounding; when I couch it that way, we see a lot of different classes in the Cary. It's not just the graphic designers who are studying typography. We see mechanical engineers who want to know how the printing presses work. They're completely invested in how the compound lever structures work in a hand press. Steve, my colleague, regularly sees the History of Math students in his teaching because we have such great books with mathematical tenets in them from the Renaissance era. We see a lot of students from the History of Music class because we have musical scores and calligraphic things. In one of the videos that I recently produced to teach this online, I make a statement that printing history is our shared history. It can be integrated in all curricula in all disciplines (`youtu.be/05RUA1ScXH0`).

D: I first heard your name in 2018 when I visited the Cary, and Steve or Kris said, "Here's our press that Amelia renovated, and out here in the hall is a wooden Gutenberg-like press that our students built, and here's this little aluminum press that you can make for something like $30." How did you learn the mechanics of all of this, restoring presses, guiding students building new presses and so on?

A: The foundation was laid with my former boss David Pankow; he's such a polymath. He's a great librarian and curator and editor and teacher; but when he was hired to be the librarian for the Cary Collection in the 1970s, he applied himself immediately as an employee of RIT and took printing classes. He learned printing while on the job, and then as presses were donated or discarded by the School of Printing because they weren't high-tech enough, David would often take those technology items into our collection — he began acquiring printing presses for the Cary Collection. So in the same way as our collection has grown since 1969 from 2,500 books

Antiphonarium fragment, ca. 1430.

to 45,000 in many different disciplines, the printing technology collection has also grown.

Learning how to take care of the press, the correct way to print with some of our irreplaceable type, that foundation was laid with David. I also learned so much from him about restoration. While I was an employee, he restored several presses, and then I learned from other people in our field, and the field has grown and especially through all the online forums we can gain access to people all over the place; that has really helped. If I have a question about the mechanics of something, I can cast wide to people who will have known or have experience with that kind of object.

D: In the *Fine Books* interview that you gave,[2] I read that you have a personal printmaking practice that uses vintage printing presses. Do you still have that, and what kind of presses do you have?

A: Yes, I do. I have enough presses for it to be annoying when we have to move them. It's like my vocation has become my avocation; even when we're on vacation, we seek out printing presses and things like that or printing places. At any rate, I personally have a Chandler & Price old style printing press, which is a foot treadle platen press from the late 1800s, and I also have a Vandercook 99, which is a little proof press. And then a stand of wood type and a stand of metal type — not too much. But my current partner, my life partner Richard Kegler, the head of P22 Type Foundry (`p22.com`), has a much more substantial letterpress studio. So, I'm able to go in his area and use his automatic presses and type too.

[2] `finebooksmagazine.com/blog/bright-young-librarians-amelia-hugill-fontanel`

D: The interview reported this was a business. Is it art printing or commercial printing?

A: It's more artistic, and I don't sell things very often at all. If I do make something available, often I donate it because I have such a good livelihood at RIT. I'd rather donate time or energy to other associations. I have sold a couple. For one I did sell recently, I donated the proceeds to GirlTrek, a non-profit that cultivates healthy lifestyles for Black women (`girltrek.org`).

D: You mentioned that you had a couple of children. Are they old enough to help you in this printing business or do they have any interest in printing?

A: My older son just turned 17. They are old enough, and they'll do something if I cajole them enough. One son I have trained since he was about eight years old to do wood type restoration; he knows how to clean wood type and just this last year through the pandemic, we did some printing of it and he seemed to be semi-interested in it, but not with a full heart like me.

D: You mentioned that you look up printing presses while on vacation. What printing museums do you particularly recommend that someone should go to?

A: One big one is the Museum of Printing in Haverhill, Massachusetts, that's run by Frank Romano. It is an excellent place to go, especially if you want to learn about like that unsung period of phototypesetting that people forget about.

Oh, hi. Here's my dog.

D: What's the dog's name?

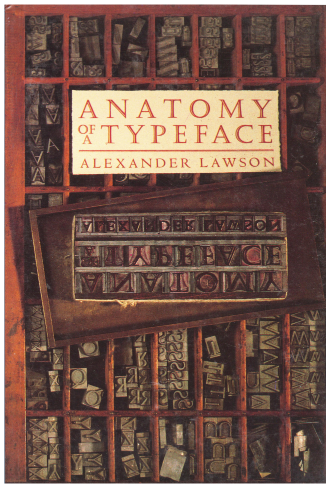
A: The dog's name is Phin, which is short for Phineas Gordon. George Phineas Gordon was the inventor of the platen press.

Another museum is in LA — the International Printing Museum — and that one is run by Mark Barbour, again, a huge, comprehensive, really amazing place. There's another in Toronto that is a sales floor and museum called the Howard Iron Works. And then, of course, the place that I really love is Hamilton Wood Type and Printing Museum, in Two Rivers, Wisconsin. If you're a Green Bay fan, it's about 45 minutes south of Green Bay. It has the largest collection of wood type in North America.

D: For a person who gets interested in typography or printing but wants to learn more about it, not become an expert like you are, just know a bit more, are there books or short courses or online courses or videos? How would you recommend somebody go about learning more about printing and printing history and typography?

A: Perhaps the easiest and probably the most accessible way right now, if you don't want to purchase books, is to get on the mailing lists of a few places around the country that are doing some very nice online lectures that relate to graphic arts and typography. For instance, the Hoffmitz Milken Center, which is at Cal Arts in California. They have a center dedicated to typography and some of the lectures coming out of there are top notch. The Type-at-Cooper Program, which is at Cooper Union in New York City, has also been having some great lectures. And now that we've been through a suite of online conferences, I expect a lot of YouTube-like channels to open up with content from the conferences. There was recently the ATypI Conference that was online, so that probably will be coming up. That's one way to dip your toe without making a big commitment in terms of expanding your library.

Of course there are great books out there on those topics. One of the books that a new student of typography might read — it might feel a little outdated, but I still like it — is Alexander Lawson's *Anatomy of a Typeface.*



I like it because, of course, Professor Lawson did most of his research at RIT, and he was an important faculty member in our history. The book also contains nice concise chapters about the classic typefaces that we see revived on our computers. There's a chapter about Garamond and there's one about Bodoni. If you want just a little snippet about those, I think Lawson's book is a good place to start. There's

Wells Book Arts Summer Institute, July 14, 2014, Kris Holmes' class "Calligraphy and Digital Type Design".

plenty more current scholarship on any one of those type designers, but I think that he definitely covers a lot of the bases in terms of classical typography through history.

D: Have you done type design yourself?

A: No. I was so happy in 2015 to be able to take a class with Kris Holmes at Wells College Book Arts Summer Institute. It was a great class because we learned calligraphy in the morning from Kris and then it was type design in the afternoon from her. I knew going into that class that I was there to soak it up because although I teach students of calligraphy, I had never done any calligraphy and similarly I teach students who are studying type design and typography, but hadn't done it myself. So, I needed to put my toe in that water in order to have a little bit more understanding about what I was teaching and it was fantastic. It was such a treat to absorb Kris' amazing knowledge and her skill in both fields. I'm sure if I practice the way she does, I would have a chance, but I'm not great.

D: The digital humanities you mentioned, what are those?

A: It's an evolving field. It has a deference to classic fields like history and sociology and English, I think, and different kinds of social sciences, but wants to enmesh the power of digital technologies so that they can do kinds of new visualizations or new kinds of publication that were not possible before without this kind of tech coming into it. That's not the official definition, just my perspective on it. We've been privileged to be able to think about it at RIT because a couple years ago, the university started a digital humanities program and we've had some great professors come in. I think the Cary is one of the places that is fodder for digital humanities students to do their projects on. So, that's a really nice thing, to see our collections viewed in a different way.

D: The next time someone can visit the Cary, are there particular parts to the collection you think that they should seek out? Do you have favorite parts yourself?

A: Each time somebody walks in, we do a little reference interview, like "What do you like? What can I show you?" and it's different for every single person. But one place we lead people to because it has such great visual impact is our press room — it happens that I'm the manager of that collection — to talk about letterpress printing technologies because it's so impressive. These are big machines that are often also decorative. So, people get impressed by that.

At the same time, I can fall in love with a tiny little miniature book too. It just depends on what somebody wants to look at. Often we get a visit from a student who walks in, an undergraduate student, who doesn't necessarily know what they want to look at. They're just beginning their education. They're just beginning to formulate the questions for the things that they want to study. Often they say, "Oh, I just want to look around." Well, they don't necessarily know what they're looking for, right? So, depending on what's on the table at any given moment, I'll try to engage them and say "Oh, my gosh. We just got this collection of posters from this collective that solicited posters from artists from all over the country. Would you like to look at that?" So, that's a nice, nonconfrontational way to get students engaged with our collections. On any given day, there might be something new on the table that somebody could look at.

D: What is the point of view of the Cary or RIT about digitizing things and then putting it online? You have a massive website. Some places historically don't want to put their stuff on the web. They want to somehow charge for access. What is the philosophy at your place?

A: Open access as much as we can. I'm so thankful to work with such a talented team in our library, library-wide. I'm part of this digital initiatives team, and over the last two years we've formulated a workflow to efficiently digitize stuff. This has served us so well in terms of education in the pandemic. If our students couldn't get in to see the original things, at least we could point them to a digital surrogate. So, that has worked well and it's a way that we can engage with people that can't come to the Cary Collection, even if we weren't in this limited patron model right now during quarantine. For instance, I could send a link off to somebody who's in England and engage with them in a conversation about our items from afar.

D: Do you have plans for coming scholarly or graphical projects you want to tell us about in advance?

A: Just today my supervisor, Steve Galbraith, who is the head curator, and I, decided to start planning an online exhibition with a lot of digital humanities kind of integration about the Kelmscott-Goudy printing press. That's the printing press that was first owned by William Morris and printed "The Kelmscott Chaucer". It was used for that book. Next year is the 125th anniversary of the book. As a result of this anniversary and some collaborative work with the American William Morris Society, we're hopefully going to have a very robust digital exhibition about the Kelmscott. Steve and I were saying today that the moss doesn't grow under our feet ever. Even with fewer students on campus, there's a lot going on.

I'm also going to contribute to a book. There was a conference called Post-Digital Letterpress Printing that was hosted in Portugal in January 2020 (`pdlp.fba.up.pt`), and I was able to participate in that conference[3] and they're doing a proceedings. So, I'm excited to contribute to the proceedings for that conference. It was a nice small conference where you were able to talk to every single person there. Sometimes big conferences are overwhelming, but this was a good one. It was before we could not travel anymore; we just squeaked it in.

D: Maybe next year or the year after that, the TeX Users Group Conference can plan again to come to RIT, and we can see each other in person.

A: That would be lovely. And in the meantime, please feel free to reach out if you have a question about anything in the Cary. We'd love to hear from *TUGboat* readers (`rit.edu/carycollection/contact-and-visit`).

D: For now let me say thank you so very, very much for taking the time to do this interview — for being willing to talk to the TeX users' world.

◇ David Walden
`tug.org/interviews`

[Editor's note: Images in this interview are courtesy of the RIT Cary Graphics Arts Collection or Amelia Hugill-Fontanel. Some images have been cropped for presentation.]

[3] Amelia's talk is online now at `youtu.be/S1ldj5LMv8Y`.

The DuckBoat — Beginners' Pond: Crazy Little Thing Called Glue

Herr Professor Paulinho van Duck

Abstract

In this installment, Prof. van Duck will explain what glue is and will suggest some tips and tricks to avoid bad spacing in your document.

1 Covid is mean!

Hi, (La)TeX friends,

As you well know, 2020 was a painful year. The virus has brought too much sadness worldwide, and the economic consequences of the pandemic are serious.

It is not easy to talk to you in the usual joyful mood, but *The Show Must Go On*, so let us try to find some not-so-negative aspects of the situation.

Firstly, while you (humans) were confined to your houses, we (animals) could visit places before unreachable for us. My fellow ducks of Sempione Park in Milan, for example, took a walk into Cadorna Station, a commuter station usually very crowded (if you are wondering if they named the station after the general who lost the battle of Caporetto: yes, it is; strange things happen in Italy, quack!).

Secondly, people had time for reading. During the first lockdown, all shops but pharmacies and grocery stores were closed. After it, the first commercial activities people wanted to be reopened were children's clothing stores, hairdressers, and *bookshops!* I hope this habit will remain in the future.

Lastly, many events happened online. Needless to say, it is not the same as being at them live, but it gave you the possibility to attend them even if they were far away. I was happy to follow the 2020 TeX Users Group meeting. It would not have been possible if it had not been online.

This time, I will show you how to avoid awful spacing between your documents' words, lines, or elements.

Barbara Beeton suggested this topic to me, inspired by the TeX.SE question *How can I visualize glue?*[1] and its brilliant answer by Donald Arseneau.

The subject is vast and complex, I will be neither rigorous nor exhaustive, but I would like to give some little tips & tricks to newbies.

I also take this opportunity to thank Barbara for her advice and corrections. Of course, all remaining errors are my own.

[1] `https://tex.stackexchange.com/questions/552527/how-can-i-visualize-glue`

2 Quack Guide No. 6: The glue

The basic elements of a (La)TeX document are *boxes* and *glue*. Every character or image is a box, but also a line of text, a table, or a minipage are boxes (which contain other boxes). The glue is the empty space between them: between words, between lines, between the text and an image or a table, and so on.

This spacing is not fixed; it may vary, according to the situation, to form a line or a page more pleasing to the eye.

Indeed, every glue is defined by three dimensions: a natural size; a stretchability, the basis for how much the natural size can be enlarged; and a shrinkability, how much it can be compressed.

For this reason, Prof. Knuth [2] said it would be more accurate to use the term "spring" instead of "glue", but the latter is a long-established tradition, and he did not change it.

TeX does not treat stretchability and shrinkability in the same way. While shrinkability represents a limit, that is, TeX will not shrink more than what is specified, this is not true for stretchability.

For instance, in the case of a vertical glue, if `\flushbottom` is in effect (see below), and if applying the maximum specified stretchability a page ends up being too short, then the places where stretch is allowed will be stretched in proportion to what is specified.

Let us see a practical example. Imagine you are writing your text on straight lines, like children at primary school do. The vertical distance between two of these lines within a paragraph is set by:

```
\baselineskip = ⟨size⟩ plus ⟨stretchability⟩
  minus ⟨shrinkability⟩
```

The previous is plain TeX syntax. You can also use it in LaTeX directly or through the command:

```
\setlength{\baselineskip}{⟨size⟩
  plus ⟨stretchability⟩
  minus ⟨shrinkability⟩}
```

In both cases, the plus and minus parts are optional.

If you do:

```
\setlength{\baselineskip}{20pt
  plus 6pt minus 4pt}
```

the distance between your lines will normally be 20pt. If necessary, TeX will reduce the spacing to a minimum of 16pt. Stretching the spacing is similar, but there is no limit to how far the glue will be stretched. Both stretching and shrinking are calculated taking into account other glue on the page. For instance, if there is one glue item with `plus 6pt` and another with `plus 2pt`, the former will expand three times as fast as the latter.

The stretching and shrinking does not normally apply to the last page of an article or chapter, which (in practice) is always ended with `\vfill` (which adds a vertical filling space till the end of the page).

If you are interested in the subject, check the TEX.SE post *What is glue stretching?*[2] for more.

For a list of the LATEX default lengths, see the LATEX/Lengths page of Wikibooks,[3] and the TEX.SE post *Lengths and when to use them.*[4]

However, you do not usually need to set these lengths; your documentclass already does it for you.

TEX is one of the best programs — if not *the* best — to manage the glue. But sometimes the automatic result may not be what you would like. Let us see how to remedy this.[5]

As general advice, I recommend making your refinements when you have completely finished your writing. Otherwise, you may have to redo them. It can be an excellent way to procrastinate, quack!

2.1 Horizontal glue

The classical situations where you need to adjust the horizontal spacing are when you get the warnings `Underfull \hbox` or `Overfull \hbox`.

Newbies often consider these messages a sort of mystery and ignore them.

On the contrary, their meaning is evident if you look at a line as a box: an underfull horizontal box is a line that is too empty; an overfull horizontal box warns you that your text goes beyond the margin (of your page, or your column table, you name it).

The `Underfull \hbox` usually appears when your paragraph is too narrow and TEX cannot correctly distribute the words in one or more lines in the paragraph.

To improve this, it is usually best to use either `\raggedright` or (from the `ragged2e` package) `\RaggedRight`.

For example, compare the columns of the following table. The first one is an ordinary `p` column; since it is too narrow, there is too much space between the words, and an `Underfull \hbox` appears. The second and the third are ragged right; note that `\RaggedRight` allows for hyphenation whereas `\raggedright` does not.

Choose the one your text fits better with.

Ordinary `p` column	With `\raggedright`	With `\RaggedRight`
Most Italians do not like pineapple pizza	Most Italians do not like pineapple pizza	Most Italians do not like pineapple pizza

```
...
\usepackage{ragged2e}
...
\begin{tabular}{p{2.2cm}
  >{\raggedright\arraybackslash}p{2.2cm}
  >{\RaggedRight\arraybackslash}p{2.2cm}}
  \toprule
  Ordinary \verb|p|~column &
  With \cs{raggedright} &
  With \cs{RaggedRight} \\
  \midrule
  Most Italians do not like pineapple pizza&
  Most Italians do not like pineapple pizza&
  Most Italians do not like pineapple pizza\\
  \bottomrule
\end{tabular}
```

To cope with overfull horizontal boxes, firstly, you have to identify where they are. With the draft option, they are highlighted by black rectangles.

The fixing depends on the kind of bad box. Let us examine some common cases.

It may happen your text contains a word LATEX is not able to hyphenate. If so, you can specify how to divide it with the command `\-`.

In the following example, the minipage width is too narrow to contain “Paulinho”, and since LATEX does not know how to hyphenate it, the first case gives an `Overfull \hbox`. If you indicate how to do it, the warning disappears.

```
\fbox{\begin{minipage}{9mm}\RaggedRight
  Paulinho is a drake
  \end{minipage}}\hspace{1cm}
\fbox{\begin{minipage}{9mm}\RaggedRight
  Pau\-li\-nho is a drake
  \end{minipage}}
```

Paulinho
is a
drake

Pau-
linho
is a
drake

Other common examples of unbreakable elements are urls. If you would like to divide them, you could use the package `xurl` (or `url` or `hyperref`), see also the TEX.SE post *Forcing linebreaks in* `\url`.[6]

[2] `https://tex.stackexchange.com/questions/64756/what-is-glue-stretching`

[3] `https://en.wikibooks.org/wiki/LaTeX/Lengths` or also other sites: `http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/squeeze.html`.

[4] `https://tex.stackexchange.com/questions/41476/lengths-and-when-to-use-them`

[5] I found some of the following explanations and suggestions on `https://latexref.xyz/` and `https://www.guitex.org/home/it/forum/5-tex-e-latex/102632-spazi-verticali-in-eccesso`.

[6] `https://tex.stackexchange.com/questions/3033/forcing-linebreaks-in-url`

Also inline math or verbatim expression may cause overfull horizontal boxes.

In general, you may try to adjust your spacing using `\sloppy`, which practically allows the space within words to be infinitely stretched. However, since that command may likely ruin the word spacing in the already well-distributed paragraphs, it is not advisable to use it for all your document. You may use the environment `sloppypar`, which acts only on the text within it.

A more convenient parameter, added in TEX3, to modify the stretching is
`\emergencystretch =` ⟨*dimen*⟩
which is similar to `\sloppy` but does not ruin good paragraphs.

You may try setting `\looseness=1` to distribute the words of a paragraph more loosely (it may expand the paragraph one line) or `\looseness=-1` to squeeze them.

You can also force a line break manually with the command `\newline`. The following example shows how it differs from `\linebreak`:

```
This line is broken\newline with
\verb|\newline|.\par
This line is broken\linebreak with
\verb|\linebreak|.
```

This line is broken
with `\newline`.
This line is broken
with `\linebreak`.

Remember never to use `\\` to break a line. You should use `\\` only in `tabular`, `array`, or similar environments to end the rows or in some math environments with alignments.

In the previous code, I also used `\par` to create a new paragraph, but the usual way is to leave a blank line.

If your problem is a too-large image, you can resize it with the appropriate options of the macro `\includegraphics` from the `graphicx` package.

You can use `width=`⟨*wdim*⟩ and `height=`⟨*hdim*⟩ (if only one of them is set, the other is scaled to keep the aspect ratio) or you can scale your image by a given factor, using `scale=`⟨*factor*⟩.

If you are creating your image yourself, using Ti*k*Z or another graphics package, the best practice should be to draw it directly with the correct dimensions.

The `graphicx` package provides `\resizebox` (resize to a width) and `\scalebox` (resize to a scaling factor). You can use them if you have a too-large table, but it is not advisable. In this case, it's usually better to choose a smaller font, starting with `\small`.

Sometimes, you may think to redesign your table. The following, for example, is too large for the minipage where it is placed:

```
\fbox{\begin{minipage}{6cm}\centering
  \begin{tabular}{cc}
  A very long title & Another very long title\\
  \midrule
  d & u\\
  c & k\\
  \end{tabular}
  \end{minipage}}
```

A very long title	Another very long title
d	u
c	k

but, instead of resizing it, you can change the column definition to allow for a manual line break:

```
\newcolumntype{M}[1]{%
  >{\centering\arraybackslash}m{#1}}
...
\fbox{\begin{minipage}{6cm}\centering
  \begin{tabular}{*2{M{2.5cm}}}
  A very long title & Another very long title\\
  \midrule
  d & u\\
  c & k\\
  \end{tabular}
  \end{minipage}
```

A very long title	Another very long title
d	u
c	k

There are also situations where there are no errors or warnings but the spacing is not correct anyway.

For instance, compare

```
Prof. van Duck with\par
Prof.\ van Duck.
```

Prof. van Duck with
Prof. van Duck.

In the first line the spacing between "Prof." and "van" is too much because TEX interprets the dot as the end of a sentence, not as an abbreviation. To correct it, just put a `\␣` (a backslash and a space) after the dot.

By the way, the space after the period is set according to the US typesetting rule. If you do not like it or there is a different rule in your country, you could use `\frenchspacing`. Look at the following example:

```
Note the space after the dot. Here\par
{\frenchspacing
Note the space after the dot. Here}
```

```
Note the space after the dot.  Here
Note the space after the dot. Here
```

In addition to `\␣`, there are many commands for explicitly setting horizontal spacing. For a detailed list, see the TEX.SE post *What commands are there for horizontal spacing?*[7]

Last but not least, I would like to talk about spurious spaces. They are unwanted spaces that appear when you least expect them, but it usually is your fault, quack!

Even the most expert TEXnicians may fall into the trap. There is an instructive article by prof. Enrico Gregorio on this topic [1].[8]

Suppose you are writing a thesis about queen Elizabeth I, and you would like to create a macro for her name, with the symbol ~ between "Elizabeth" and "I", to avoid them being separated onto different lines. In the following simple example, the macro `\eliwrong` creates a spurious space, clearly visible if you compare it with `\eligood`.

```
   Someone says the greatest kings of England
were queens,  Elizabeth I and Victoria.
   Someone says the greatest kings of England
were queens, Elizabeth I and Victoria.
```

```
\newcommand\eliwrong{
  Elizabeth~I}
\newcommand\eligood{% <-- note the %
  Elizabeth~I}
Someone says the greatest kings of England
  were queens, \eliwrong\ and Victoria.\par
Someone says the greatest kings of England
  were queens, \eligood\ and Victoria.
```

Note also the `\␣` after the macros (for example, `\eligood\␣`), which is needed to add a space if no punctuation mark is following. But if `eligood` ends a sentence (e.g., if we had written "`Victoria and \eligood.`"), the capital "I" would cause the space to be only a normal interword space instead of recognized as ending a sentence. To fix it, add `\@` before the period: "`\eligood\@.`".

[7] `https://tex.stackexchange.com/questions/74353/what-commands-are-there-for-horizontal-spacing`

[8] And a funny video: `https://youtu.be/jGAiF2LBLuY`

Remember, when you go to a new line in a `.tex` file, you are putting a space in your final output document. People accustomed to ordinary word editors or programming languages usually find it strange and forget to add the `%` sign when needed.

TEX ignores what is written after a percentage sign and also ignores the leading spaces of a line.

Therefore, pay attention when you put a comment at the end of a line; you may make a space disappear (of course, there are cases where this behavior is convenient, as above, in the definition of the macro `\eligood`).

```
Here:% This is a comment
   a space is missing\par
Here: % This is a comment
   a space is present
```

```
Here:a space is missing
Here: a space is present
```

2.2 Vertical glue

Having adjusted the horizontal glue, let us move on to the vertical.

Many LATEX document classes use the declaration `\flushbottom` when the text is two-sided. Indeed, it makes all the pages of the same height, adding some vertical space if necessary. Usually, this is pleasant to the eye because the left page looks equal to the right one.

Nevertheless, if your pages have many math formulae, tables, figures, and not only plain text, `\flushbottom` may add too much vertical glue.

If this exceeds the allowed stretchability, you may get the warning `Underfull \vbox`.

Sometimes, you may have *widows* and *orphans*, typographical terms to indicate respectively a single ending line at the beginning of a page or a single opening line at the end of a page. There is an interesting article by Frank Mittelbach [4] on this topic.

Let us see how to solve the problem.

First of all, you can use `\raggedbottom`. It does not add extra vertical space to your page. Of course, the downside is that your pages will have different heights, and they could look bad if they are facing pages of a two-sided document.

You may use `\raggedbottom`, also temporarily, to see if any of your pages are too empty and then adjust them. There are many ways to do it.

You may modify some vertical glues, refining their stretchability or shrinkability. For instance, you can adjust the space between paragraphs with `\parskip`.

You may also increase (or decrease if the parameter is negative) the space after a specific paragraph with `\vspace{`⟨*length*⟩`}`. If you place this command in the middle of a paragraph, it is deferred until after the paragraph ends. The space is not added if it is at the beginning of the page, among other places. There is also a starred version of the command, `\vspace*{`⟨*length*⟩`}`, which adds the space also if it is at the top of the page, or would otherwise be ignored.

Of course, there are many more commands for vertical spacing; for a list, see `https://latexref.xyz/Spaces.html`.

If you are not constrained to some typesetting rules, you may try to (slightly) increase or decrease your `textheight`; `geometry` is the recommended package for this.

You can even change the height of a single page with `\enlargethispage`⟨*size*⟩. The size could also be negative, for example

```
\enlargethispage{-\baselineskip}
```

reduces the current page by a line. There is also a starred version of the command that shrinks the glue between your page elements, if possible, to make room for the indicated size.

Moreover, you can suggest to (LA)TEX where a page break should be encouraged or discouraged, respectively, with the macros `\pagebreak`[⟨*num*⟩] and `\nopagebreak`[⟨*num*⟩].

The optional parameter ⟨*num*⟩ is the suggestion's strength. It may vary from 1 (weak) to 4 (mandatory, the default).

Note that these commands apply to the next new line with respect to the point where they are positioned.

```
For example, suppose this is a paragraph at the
end of a page. If you put a \verb|\pagebreak|
here: \pagebreak it will not break the page
there but after the end of the line where
it appears.
```

End of the current page:

> For example, suppose this is a paragraph at the end of a page. If you put a `\pagebreak` here: it will not break the page there but after

Beginning of the next page:

> the end of the line where it appears.

A more abrupt command is `\newpage`, which ends the current page immediately and does not vertically stretch it. Note that a new paragraph is started on the new page. Hence, you should use the command at the end of a paragraph. If you would like to use it in the middle, as in the following example, you should add `\noindent`, if needed, to avoid the possible paragraph indentation.

```
If you put a \verb|\newpage| here:\newpage
\noindent the page breaks at once.
```

> If you put a `\newpage` here:

> the page breaks at once.

Analogously to `\newpage`, also `\clearpage` and `\cleardoublepage` end the current page at once. The latter creates a blank page, if needed, to start the new page always as an odd page; it is used in two-sided documents. If your document has two columns per page, they eventually leave an empty column, whereas `\newpage` does not.

Another difference with `\newpage` is that they also typeset all the floating environments present in the float queue (see below). Hence, you can use these two commands to print all your floats before a certain point if you need this.

If we had to make a ranking of things that shock beginners, floating environments undoubtedly win the gold medal.

The standard ones are `table` and `figure`, but there are others, and you can also create your own floating environment, for example, with the package `float`.

The adjective "floating" always reminds me of a delicious French dessert, poison for diabetics, named *île flottante*. It is an iceberg of meringue that sails on an ocean of custard.

The newbie-astonishing property of these environments is just that they wander about in your document as the meringue glides on the custard. With ordinary word processors, tables and figures usually remain where you put them.

Of course, this is a plus of (LA)TEX, because it places floating elements professionally. Look at any textbook of yours. You will see that figures and tables are generally not positioned where they are referenced, but at a specific place on the page. The text references them with a number (one of the reasons why I love (LA)TEX is exactly its efficient and straightforward way of referencing objects).

The positioning mechanism is quite complex and usually not immediately grasped by beginners. On this topic, I recommend you to read a detailed and clarifying article by Frank Mittelbach [3].

In brief, the floating environments have an optional parameter that specifies where to place the float. It may contain the characters: `h` for *here*; `t` and `b` for *top* and *bottom* of the page; `p` for a float *page*; and a `!` that indicates some positioning restrictions should be ignored.

LaTeX tries to place the float where specified. If it does not succeed, the float is put in a queue which is disposed of on the next page, if possible, or at the end of the document, or when a `\clearpage` is encountered.

The first "strange" thing (to newbies) is that the position specifiers' order is irrelevant. If you write `ht` or `th` it is the same. It does *not* mean that, in the first case, "here" is tried first and, in the second, "top" is the first option. It only means that (La)TeX will not place the floats at the "bottom" or in a "page", at least not at first.

The second is that `h` does *not* mean "absolutely here" but more or less "the float will be placed at the end of the paragraph where it is mentioned, if possible."

After getting over the shock, newbies try with every means to make the floats not floating, for example, using the position specifier `H` from the `float` package. But if you do not want your figures or tables to float, simply do not use *floating* environments, quack!

The command
`\captionof`⟨*float type*⟩ `[`⟨*list entry*⟩`]{`⟨*heading*⟩`}`
from package `caption` (or `capt-of`) allows you to have consistent numbering and labeling without using floats. The parameter ⟨*float type*⟩, that is `figure`, `table`, etc., is mandatory; ⟨*list entry*⟩ is optional and represents what you want to put in the list of figures, tables, etc.

In the following example, I use a `center` environment, which does not float. You may use a `minipage` with an appropriate alignment as well.

By the way, remember never to use a `center` environment *within* a floating environment, because it would add extra vertical space, use `\centering` instead.[9]

```
...
\usepackage{caption}
...
\listoffigures
See Figure \ref{fig:my}, for example:
\begin{center}
\includegraphics[width=3cm]{example-image}
\captionof{figure}[An image]
  {An example image}\label{fig:my}
\end{center}
```

List of Figures

1 An image 39

See Figure 1, for example:

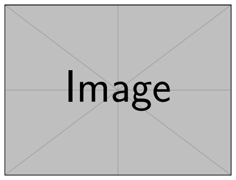


Figure 1: An example image

3 Conclusion

While I was writing this article (January 2021), other bad news reached me. With sadness in my heart, let me remember Gustavo Mezzetti, a brilliant TeXnician and TeX.SE friend, author of the gorgeous `halloweenmath` package. He generously helped many people also on the GuIT Forum, with the nickname of `letteracdp` (taken from a documentclass he wrote years ago).

Ciao GuM, we will miss you!

References

[1] E. Gregorio. Recollections of a spurious space catcher. *TUGboat* 36(2):149–161, 2015. `https://tug.org/TUGboat/tb36-2/tb113gregorio.pdf`

[2] D.E. Knuth. *The TeXbook*, vol. A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[3] F. Mittelbach. How to influence the position of float environments like figure and table in LaTeX? *TUGboat* 35(3):248–254, 2014. `https://tug.org/TUGboat/tb35-3/tb111mitt-float.pdf`

[4] F. Mittelbach. Managing forlorn paragraph lines (a.k.a. widows and orphans) in LaTeX. *TUGboat* 39(3):246–251, 2018. `https://tug.org/TUGboat/tb39-3/tb123mitt-widows.pdf`

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond
Milano, Italy
`paulinho dot vanduck (at) gmail dot com`

[9] `https://tex.stackexchange.com/questions/23650/when-should-we-use-begincenter-instead-of-centering/23653`

Creating document commands: The good, the bad and the ugly

Joseph Wright

Creating document commands in LaTeX has traditionally involved a mix of `\newcommand`, semi-internal kernel commands (like `\@ifnextchar` and `\@ifstar`) and low-level TeX programming using `\def`. As part of wider efforts to improve LaTeX, the team have over the past few years developed ideas for creating document commands in the package xparse. In a parallel article (on `\NewDocumentCommand`, on the following pages), I've looked at how the xparse ideas compare to the abilities of other packages.

The aims of xparse have always been two-fold: to provide a clear way to create *new* commands, and to provide a language to describe *existing* ones. It is also intended to be as flexible as possible, so it doesn't impose artificial restrictions on syntax. That comes at a cost, however: it can be (ab)used to create commands that do not fit into the standard LaTeX pattern.

The LaTeX kernel now integrates most, though not all, of xparse into the kernel. That means the core ideas are available out-of-the-box. This seems like a good time, therefore, to look at the best ways to use the abilities of xparse in making document commands. I won't look at the full detail, but rather pick out how to, and how not to, create good document commands.

1 The Good

The LaTeX kernel is very careful to have consistent syntax for document commands. It uses only a small number of the possible argument types, which I'll describe in xparse terms:

- Mandatory (`m`) arguments in braces.
- Optional (`o`/`O{<default>}`) arguments in `[]`, which may have a default; in xparse terms we can tell the difference between a missing optional argument and one given with an empty `[]` pair.
- An optional star (`s`).
- Picture co-ordinates (`r()`), which are split into x and y, so in xparse terms subject to `\SplitArgument`.

Most of the time, the LaTeX kernel makes arguments *long*, which is shown as `+` in xparse syntax.

A star is *always* used as the first argument after a command, so in some ways it looks like part of the command name itself. Optional arguments are *almost* always given before mandatory ones, and most of the time there is *only one*. Where two are used, for example with `\makebox`, it's because the second is *strictly dependent* on the presence of the first.

Following the kernel, signatures (argument descriptions) such as:

```
s o m
s O{<default>} m m
o m m
```

are 'good'. You *can* use something like

```
s +m O{0} +o +m
```

(with optional arguments after a mandatory one; this is the syntax of `\newcommand`!) if you are careful, but *think very carefully.*

There's one syntax that's not from the kernel but is recommended where it applies: the beamer overlay syntax, which is `d<>` in xparse terms. This always comes first (other than a star), and is best reserved for the 'on X slides of Y' idea in presentations (doesn't have to be using beamer).

xparse lets us create arguments using `_` and `^`, similar to TeX's core math mode syntax. Most of the time, this should be reserved for math mode where you need to emulate the TeX syntax but for some reason need to grab the arguments yourself. This is done using `e{^_}`.

2 The Bad

The above already shows we have quite a few combinations available. Things go bad when too many combinations are used. For example:

- Multiple optional arguments where the second or subsequent ones don't strictly depend on the earlier ones.
- Optional arguments using tokens other than `[]` (or `<>` for overlays).
- Testing for tokens other than `*` as 'a special case' (think things like `+`).

Almost always, complex setups using these types of combination mean *you need to rethink the syntax.* In particular, multiple optional arguments tend to be much better replaced by using a keyval approach.

3 The Ugly

Some ideas in xparse won't be making it to the kernel: these are definitely the Ugly. They'll stay in a stub xparse for historical reasons, and as they do describe some syntax choices people have made, but in truth, they should be avoided:

- *Optional* groups (`g`) in braces; breaks the LaTeX conventions badly.
- Arguments *up to* a left brace (`l`); useful at a low level, but not in a document command.

Joseph Wright

doi.org/10.47397/tb/42-1/tb130wright-goodbad

- Arguments *up to* a token (`u`); widely used in programming, but again not in document commands.

You might wonder why they are all there in the first place: these were part of the more experimental work in xparse, and those particular experiments have shown we don't want to enable such syntaxes even for emulating existing commands.

4 A Fistful of Tokens

There are of course places where you need to go outside of the xparse structures, particularly when parsing specialist data. The popular Ti*k*Z graphics system is one example; linguistic glosses are another. But these are restricted contexts, normally used within a dedicated environment where it is clear that the 'usual' rules do not apply. Basically, if you do this, you are on your own, so be sure to check the balance of consistency *versus* compactness.

5 For a Few Tokens More

Using xparse syntax makes it *much* easier to have a clear break between interface and implementation. As such, the fact that it's got more going on 'beneath the hood' is worth it: it's a lot easier to track what's happening. The move into the kernel will make using xparse descriptions even easier to exploit, so it's important that users defining their own commands give a little thought to the syntax they choose.

◇ Joseph Wright
Northampton, United Kingdom
`joseph dot wright (at) morningstar2.co.uk`

`\NewDocumentCommand` *versus* `\newcommand` *versus* ...

Joseph Wright

Creating new document commands in LaTeX has traditionally been the job of `\newcommand`. This lets you create a command with mandatory arguments, and also support a first optional argument. However, it can't create more complex commands: LaTeX uses stars, multiple optional arguments, and plenty more. To define commands using such syntaxes, the kernel itself uses lower-level TeX programming. But this is opaque to many users, and a variety of packages have been created to ease the burden.

Over the last decade, the LaTeX team have developed xparse, a generic document command parser, as a way to unify many ideas and provide a single consistent way to create document commands. The bulk of that code has now been moved to the LaTeX kernel, and in a parallel article (starts on the preceding page) I've provided some ideas about how best to exploit that.

In this article, I want to look at a related issue: why to use this 'xparse' approach, and how it compares to existing solutions, both in the LaTeX kernel and the wider package sphere. Here, I'm going to avoid talking about 'simple' shortcuts (things such as `\newcommand\myname{Joseph}`): these are best left to `\newcommand`. Instead, I want to deal with commands which take arguments and have some element of 'programming' to them.

What I'll seek to highlight here is that using `\NewDocumentCommand`, we get a single consistent and reliable way to create a variety of commands. There's no need to worry about clashes between approaches, and it all 'just works'.

1 Preliminaries: Protected commands and optional arguments

Before we start, a couple of things are worth mentioning. First, there is the idea of 'protected' commands. In some places, we need commands not to 'expand' (turn into their definition). With a modern TeX system, that can be arranged by the engine itself (pdfTeX or similar), using ε-TeX's `\protected` primitive (built-in). The LaTeX kernel doesn't use that mechanism in `\newcommand`, but lots of other tools do. I'm going to assume that we want to make protected commands unless I mention otherwise. Almost always, unless you are creating a 'shortcut' for some text, you want your commands to be protected.

The second thing to note is that TeX itself has no concept of optional arguments, so they are always arranged using some clever look-ahead code. In xparse, nested optional arguments are handled automatically, but again, `\newcommand` and similar do not do that.

2 The kernel: *versus* `\newcommand`

The kernel's `\newcommand` can, as I've said, create commands with multiple mandatory arguments but with only one optional one. A simple example:

```
\newcommand\foo[3][default]{%
    Code perhaps using #1 and
    definitely using #2 and #3%
}
```

We can of course create an equivalent command using `\NewDocumentCommand`:

```
\NewDocumentCommand\foo{+O{default} +m +m}{%
    Code perhaps using #1 and
    definitely using #2 and #3%
}
```

doi.org/10.47397/tb/42-1/tb130wright-newdoccmd

You may notice that I've used `+m` for both of the mandatory arguments, as that matches `\newcommand`: the arguments can accept paragraphs (is `\long`, in TeX terms). With `\newcommand`, all arguments either accept `\par` or do not: with `\NewDocumentCommand` we can select on a per-argument level what happens.

The optional argument with a default works using `O{default}`, and the result will be the same functionality as `\newcommand`. We gain the idea that nested optional arguments are parsed properly, some better error messages if we use `\foo` incorrectly, and an engine-robust definition of `\foo`.

We can't do a lot more with `\newcommand`, so rather than try to show other `\NewDocumentCommand` features here, we'll first consider how we might make more complex syntaxes using just the classical LaTeX kernel.

3 ... *versus* `\def`: The primitive

Using the TeX primitive `\def`, plus the kernel internal commands `\@ifstar` and `\@ifnextchar`, we can construct more complex syntaxes. For example, let's create the syntax for `\section`: a star, an optional argument and a mandatory one. I'll assume we have `@` defined as a letter here. I'm also going to pass the presence of a star as the text `true` or `false`, as it makes things clearer.

```
\newcommand\section{%
    \@ifstar
      {\section@auxi{true}}
      {\section@auxi{false}}%
}
\def\section@starred#1{%
    \@ifnextchar[%]
      {\section@auxii{#1}}
      {\section@auxii{#1}[]}%
}
\long\def\section@auxii#1[#2]#3{%
    % Here:
    % #1 is "true"/"false" for a star
    % #2 is the optional argument
    % #3 is the mandatory argument
}
```

As you can see, this is a bit tricky already, and it doesn't cover the case where we want to have the optional argument default to the mandatory one, when it's not given. It also doesn't allow for nested optional arguments, and it's not engine-robust. We might of course use more complex paths for the star: we could have independent routes.

Using `\NewDocumentCommand`, things are much easier:

```
\NewDocumentCommand\section{s +O{#3} +m}{%
  % Here:
  % #1 is "true"/"false" for a star
  % #2 is the optional argument
  % #3 is the mandatory argument
}
```

The minor difference now is that `#1` is a special token that we can test for truth using `\IFBooleanTF`. I've also allowed for the optional argument picking up the mandatory one (`#3`), when it's not given.

We could make more complex examples, but the bottom line is: using `\NewDocumentCommand`, we are going to have simple one-line interface descriptions, and the behind-the-scenes TeX argument parsing is hidden away.

4 ... *versus* `\newrobustcmd`: etoolbox

The etoolboxpackage offers `\newrobustcmd` as a complement to `\newcommand`. It provides *exactly the same* interface as `\newcommand`, except it uses ε-TeX to make engine-protected commands. Here's an interface point of view, there's nothing new here.

5 ... *versus* `\newcommandtwoopt`: twoopt

The twoopt package supports a syntax similar to `\newcommand` but for creating two optional arguments. We'll take an example from its documentation:

```
\newcommandtwoopt\bsp[3][AA][BB]{%
    \typeout{\string\bsp: #1,#2,#3}%
}
```

This is reasonably clear: we have an optional argument `#1`, and optional argument `#2` and a mandatory argument `#3`. The two optional arguments each here have a default.

How does this look with `\NewDocumentCommand`?

```
\NewDocumentCommand\bsp{+O{AA} +O{BB} +m}{%
    \typeout{\string\bsp: #1,#2,#3}%
}
```

You'll see that we stay consistent here: the same syntax is used to create one, two or even more optional arguments. I wouldn't recommend using multiple optional arguments in most cases, but when we do, it's a lot easier using `\NewDocumentCommand`.

One thing that `\NewDocumentCommand` can do, but twoopt *cannot*, is create optional arguments that are not in the first or second positions. With twoopt, that would require either the TeX coding we've already seen, or using a different tool again.

6 ... *versus* `\withsuffix`: **suffix**

The suffix package allows one to extend an existing command to look for an optional token ('suffix') immediately after the command name. Taking a simple example from StackExchange (`https://tex.stackexchange.com/a/4388`), we start with

```
\newcommand\foo{blah}
\WithSuffix\newcommand\foo*{blahblah}
```

which translates to

```
\NewDocumentCommand\foo{s}{%
   \IFBooleanTF{#1}
     {blah}
     {blahblah}
}
```

This means we only need one line for the interface set up, and don't need, for example, to split up grabbing optional arguments into two different places (as in the previous example with `\section`).

7 ... *versus* `\newcommandx`: **xargs**

The xargs package is perhaps the most complete approach to extending `\newcommand` as far as optional arguments are concerned. It provides `\newcommandx`, which has the same syntax as `\newcommand` but where the second optional argument is a key–value list, which then describes which arguments are optional, and what their defaults are. Taking an example from the documentation:

```
\newcommandx*\coord[3][2=1,3=n]{%
  (#2_{#1},\ldots,#2_{#3})}
```

would create a command with two optional arguments, `#2` and `#3` (each with defaults), leaving `#1` mandatory. Translating into `\NewDocumentCommand` syntax might make that clearer!

```
\NewDocumentCommand\coord{m O{1} O{n}}{%
    (#2_{#1},\ldots,#2_{#3})%
}
```

The xargs package has the idea of `usedefault`, which allows `[]` to be the same as `[default]`. That's not something xparse does, as it is pretty confusing: what happens when you want an empty optional argument? This links to something I've said before: avoid consecutive optional arguments *unless* the second is dependent on the first.

8 ... *versus* `newcommand.py`: **newcommand**

Stepping outside of TeX itself, Scott Pakin's Python script `newcommand.py` provides a description language somewhat like xparse, and converts this into a 'template' of TeX code, allowing a 'fill in the blanks' approach to creating commands. It can cover several of the ideas that xparse can, including a few that will *not* be migrated to the LaTeX kernel. It can also set up a command taking more than 9 arguments, but that's always going to be tricky as a user.

What is important is that using a script means we have to work in two steps, and it's hard to see what's happening from the TeX source. It also doesn't offer anything that the kernel doesn't already do: no protected commands, no nested optional arguments, no improved error messages. So in many ways this is using techniques we've already seen, just made a little more accessible, at least if you have Python installed.

9 ... *versus* `\NewEnviron`: **environ**

As well as document commands, the xparse syntax can be used to create document *environments*: the same relationship we have between `\newcommand` and `\newenvironment`. What people sometimes want to do is grab an entire document environment body and use it like a command argument. Classically, one does that using the environ package. Again, taking an example from the documentation:

```
\NewEnviron{test}{%
  \fbox{\parbox{1.5cm}{\BODY}}\color{red}
  \fbox{\parbox{1.5cm}{\BODY}}%
}
```

would grab all of the body of the environment `test` and typeset it twice, the first time in red. That is, the environment body is saved as `\BODY`.

Using `\NewDocumentEnvironment`, we have a syntax similar to `\newenvironment`

```
\NewDocumentEnvironment{test}{+b}{%
  \fbox{\parbox{1.5cm}{#1}}\color{red}
  \fbox{\parbox{1.5cm}{#1}}%
}{}
```

with the argument grabbed in the normal way as (here) `#1`. We can therefore have 'real' arguments first, then grab the body.

10 Summary

Using the tools set up in `\NewDocumentCommand`, we can have a consistent way of creating a wide range of document commands. Rather than use a mixture of tools, from the kernel, the TeX engine, and the package sphere, it is far preferable to use the single interface of `\NewDocumentCommand` for defining new commands today.

◇ Joseph Wright
Northampton, United Kingdom
`joseph dot wright (at) morningstar2.co.uk`

Comparison of OpTEX with other formats: LATEX and ConTEXt

Petr Olšák

Introduction

OpTEX [1] was introduced in an article [2] in the previous issue of *TUGboat*. It is a macro package that creates a format for LuaTEX. Its features are comparable with other formats like LATEX or ConTEXt. One may ask why use a new format, particularly when it requires a different markup syntax? I try to answer this question here. I present a comparison among the LATEX, ConTEXt, and OpTEX formats, from various points of view.

Basic concept

LATEX. It was created in the 1980s as a real *format*, i.e. the implementation of visual and typographical aspects of the TEX output. Moreover, it provides a markup language given in [3] which was intended for the authors of (typical) scientific publications. Authors are instructed: use this markup and don't worry about the typographical look of the output. This look is implemented for you in the format.

A very important feature of LATEX is its modularity. There are macro packages that solve particular problems with the typesetting of documents which can be loaded when the document is processed. This concept has grown to a size which nobody could have expected in the 1980s. Now, some packages give authors an interface to set various typographical parameters of the typesetting too. So, there is no single format of the output. But the possibility of controlling the visual aspect of the output has no uniform strategy. It is spread among various packages created by various authors.

LATEX introduces a new level of terminology, syntax, etc., over the TEX primitive level. We are not using *control sequences* (meaning macros, registers, ...) in LATEX. There are *commands* and newly introduced *functions* and *variables* here. But the interpreter is TEX, so it reports (for example) the message "`undefined control sequence`" and the LATEX users may not know this term and they may not understand such messages. For example, a typical TEX message "`missing \cr inserted`" is not understandable for average LATEX users because they are using `\\`, not `\cr`.

The different LATEX syntax can be shown in the following example. The setting to the register which controls the width of the typesetting area is documented as

```
\setlength{\textwidth}{13cm}
```

which would be `\hsize=13cm` at the TEX primitive level. The primitive level is allowed in LATEX documents too, so we often see a mix of primitive syntax and LATEX syntax in real-world documents.

In the last ten years, a new language level over the primitive level has been developed, used, and propagated by the LATEX team: expl3. It is intended to be used by macro/package writers for LATEX. It is even further from the TEX primitive level. For example:

```
\tl_set:Nn \l_pkgname_hello_tl { Hello! }
```

is comparable to `\def\hello{Hello!}` from the primitive point of view. Very special naming conventions must be used here. And different terminology is used: the `\l_pkgname_hello_tl` is not a macro without parameters, but rather a *variable*. The `\tl_set:Nn` is not a macro that expands to the `\def` primitive, but rather a *function*.

ConTEXt. It was created and is still developed by Hans Hagen (and colleagues). The first released version was in 1994 (in Dutch) and the ConTEXt name was given to the package in 1996. Now, we have a development version 'ConTEXt lmtx' based on the LuaMetaTEX engine (not included in TEX Live) and the stable 'ConTEXt mkIV' based on the LuaTEX engine. When I use the word ConTEXt in this article, I mean the stable ConTEXt, because I don't have experience with the development version.

ConTEXt is not only a format, it is a tool that enables one to set the typesetting parameters consistently and process the document. All features used in typical present-day documents are supported in one place without the need to load external and third-party packages.

The settings of the typesetting parameters are done with `\setup...` commands in key-value syntax. So, new syntax over primitive TEX syntax is created here. And the distance from this level of syntax to the primitive level seems to be quite big. For example, the primitive `\hsize` is set when a parameter `width` in a special context is used in ConTEXt.

ConTEXt is closely associated with MetaPost for creating vector graphics which can be programmed and which "cooperate" with typesetting material. LATEX and OpTEX more commonly use Ti*k*Z for this, although all the formats support both graphics packages, among others.

doi.org/10.47397/tb/42-1/tb130olsak-fmtcmp

OpTEX. It was released in 2020 and the first stable version dates from February 2021. It is the successor of the OPmac macros [4, 5] specially designed as a format for the LuaTEX engine.

OpTEX is similar to plain TEX but provides myriad additional features needed when preparing typical documents in PDF format. The list of features is presented in the next section.

OpTEX does not try to define a new level of language over the primitive level of TEX. It is intended that if something is not supported in OpTEX macros then the user works at the TEX primitive level (or plain TEX or OpTEX basic macro API). For example, when you want to set the width of the typesetting area, use simply `\hsize=13cm`.

The macros are straightforward, they solve only what is explicitly needed. They do not scan nor manipulate with lots of parameters for setting typography. OpTEX generates a "default typography" if nothing more is done. The main concept is: if you want different typography or different behavior, then copy appropriate macros from the OpTEX kernel and make changes to them in your macro file. For example, suppose you want to give a different look to section titles. Then copy the macro `\_printsec` from OpTEX and modify it. You can see that the macro uses primitive commands for typesetting: `\hbox`, `\vbox`, `\kern`, `\vskip`, `\hskip`, `\penalty`, etc. You can use this box-penalty-glue concept directly without any inserted inter-layer of language. This is very natural in TEX and you can use the full power of TEX.

We can summarize the basic concept of OpTEX in two sentences: (1) We can return to the original TEX principles. (2) Simplicity is power.

Kernel versus packages

LATEX. The features implemented directly in the LATEX kernel (i.e. in the `.fmt` file) correspond to the time of LATEX's origin. There is no color support, no support for graphics insertions, no hyperlink support, no Unicode font support, etc. All these additional features are solved using external packages. So, document preambles contain plenty of `\usepackage` commands to load additional macros. Without them, you cannot solve typical problems when processing today's documents.

ConTEXt. It has an almost monolithic kernel that implements all necessary features. There is a possibility of "modules" in ConTEXt (something comparable to LATEX packages) but they are not typically used because virtually all features are present in the kernel.

OpTEX. The kernel implements:

- all macros from plain TEX,
- font selection system for Unicode fonts,
- Unicode math,
- color support including color mixing,
- graphic insertions,
- creating simple graphic elements,
- typesetting at absolute or relative positions,
- external and internal references and hyperlinks,
- automatic generation of a table of contents,
- generation of `\cite` references from `.bib` files,
- creating alphabetically sorted indexes,
- hyphenation for all available languages,
- switching between language-dependent phrases,
- footnotes and marginal notes,
- verbatim listings including syntax highlighting,
- PDF outlines in Unicode,
- creating tables with a new `\table` macro,
- creating slides for presentations,
- simple predefined styles "letter" and "report",
- comfortable setting of page layout,
- printing "lorem ipsum dolor sit",
- simple API for macro writers,
- loops and key-value syntax for macro writers,
- namespaces for users and macro writers.

Many other features can be implemented in a small number of macro lines. They are listed in the OpTEX tricks web page [6]. Other such features will be added here if a user asks me to solve a new problem. A user can copy the macro lines from this web page to his/her macro file and (possibly) modify them and use them. Almost every feature listed here is typically comparable with using some LATEX package, but is solved with more straightforward macros. For example, the feature comparable with the `import` package is implemented by three lines of macro code in OpTEX trick 0035. The LATEX package `import` itself has 120 lines of macro code.

Some features take more than a few lines of code. OpTEX supports loading macro packages too, with a `\load` macro. For example, there are packages:

- `qrcode` calculates QR codes and prints them,
- `vlna` handles non-breakable spaces after Czech/Slovak prepositions and other similar typographical features, using the `luavlna` package
- `emoji` enables printing of a large number of colorized emoticons from the special Unicode font.

Documentation

LaTeX. The LaTeX project page [7] lists 8 files as "general documentation" with 160 pages in total, but this is not all. The mentioned documentation describes LaTeX kernel features. But users need to use dozens of packages when an average document is prepared. Each package has its documentation. This represents hundreds or thousands of additional documentation pages from various authors. The documentation is of different ages in different styles. It is difficult to recognize what is relevant and what is obsolete. There exists a book [8] that summarizes features of LaTeX and of all typically used LaTeX packages but not every LaTeX user has access to this book. And features of recent versions of packages may differ from those described in this book.

There is the LaTeX `doc` system: the macro programmer can write code and technical comments together. You need to pre-process these sources to get macro files usable when the format is generated or documents are processed. This system is widely used by LaTeX package programmers.

ConTeXt. There are about 180 PDF files with various ConTeXt documentation. It is not within the power of the average user to know such a huge amount of information and be able to select the most important parts when starting with ConTeXt. On the other hand, this illustrates that ConTeXt covers a very large area of computer typesetting.

OpTeX. The main OpTeX manual [9] is divided into two parts: user and technical documentation. The first part has only 22 pages. There is a summary of OpTeX markup at page 26. You can click on each control sequence listed here and the relevant part of user documentation is shown. You can click again on a control sequence here and your PDF viewer jumps to the second part with technical documentation and with a detailed technical description of the macro and with the macro source printed there.

The manual is created using OpTeX, of course. When the technical part of the documentation is processed, the actual OpTeX files with the macro sources and with detailed comments are read. It is similar to literate programming where technical notes and code sources are together in a common file. You don't have to pre-process this file: the files are ready to be read when OpTeX format is generated and when the documentation is prepared.

Of course, if the user wants information beyond document markup, then he or she must know the basics of TeX itself and plain TeX. This information is summarized in document [10] in 26 pages. More information about TeX math typesetting and Unicode math is summarized in document [11] in 30 pages.

The three documents mentioned above [9, 10, 11] are sufficient to acquire all knowledge about OpTeX and TeX. They are all that you need to know when working with OpTeX documents. Of course, you can get more information about the TeX engine used by OpTeX: [12, 13, 14]. No other documents are needed.

External programs

LaTeX. When creating the reference list of `\cite`d records from a `.bib` file, LaTeX needs an external program (ancient BibTeX or newer Biber). Users have to run this sequence: `latex`, `biber`, `latex`, `latex` to get the correct reference list and `\cite` numbers.

When creating an index, the external program Makeindex or Xindy is used. The user must not forget to run `makeindex` followed by `latex` after final corrections (just before sending to print) are done.

When printing a code listing with syntax highlighting, some LaTeX packages use another external program, such as a python script.

OpTeX. We don't have to use any external programs when creating bibliography references, sorting an index, or printing code listings with syntax highlighting. All these tasks are done at the TeX macro level.

OpTeX reads the `.bib` file directly and creates the appropriate reference list. A bibliography style (a simple macro file) is used to set the rules for printing and sorting records. If all `\cite` commands are before the reference list generated by `\usebib`, then you need to run OpTeX only two times: the first run accumulates all labels used in `\cite` commands, and the records with these labels are read from the `.bib` file. The reference list is created and labels are connected to the generated numbers and saved to a `.ref` file. The second run uses these label-number pairs to print the numbers at the places where `\cite` commands are used.

Index sorting is done by a merge sort algorithm which is very efficient in the TeX macro expansion language. The special two-pass algorithm is used for similar phrases, which is configurable for almost all languages. The sorting rules are applied by the currently selected language.

The syntax highlighting of code listings is done at the TeX macro level. It is configured in macro files `hisyntax-c`, `hisyntax-html`, etc.

Font selection system

LaTeX. In the 1990s, the "New font selection scheme" (NFSS) was designed, along with the LaTeX2ε release. The system allows the selection of family, weight, shape, size, and encoding of the font independently. It was designed for old fonts with a maximum of 256 characters in the font. The NFSS creates a new layer over the `\font` primitive. Special internal names for font families were used and declared in font definition files (`.fd`). Each newly installed font was typically converted to various 8-bit encodings. The virtual fonts technique was frequently used. This required a high level of understanding fonts in TeX, making font installation almost impossible for average users.

When Unicode engines were released then a new `fontspec` package for selecting Unicode fonts was designed. It follows the principles given by NFSS but adds new features to support Unicode text fonts.

OpTeX. The font selection system respects the basic plain TeX principle: you can select font variants typically by `\rm`, `\it`, `\bf` and `\bi` selectors when a font family is loaded by `\fontfam`. The variant is selected with respect to the current "font context" given by the current setting of font size and more features given by "font modifiers". The set of font modifiers (for example `\cond`, `\caps`, `\sans`, `\bolder`) depends on features provided by the selected font family. Users can combine the font modifiers arbitrarily: the appropriate font is selected if the font family provides it. The font families including their modifiers are implemented in "font files". The log file shows the available font modifiers of the selected font family. Users can create a font catalog with simply `\fontfam[catalog]\bye`. All families registered in font files are printed in this catalog; see [15]. You can see all available families, modifiers, and variants here. Font samples for each such combination are shown.

Unicode fonts are preferred in OpTeX. An appropriate Unicode math font is loaded automatically too when `\fontfam` is used. The special font modifier `\setff{...}` sets an arbitrary OpenType font feature if it is supported by the font.

Of course, OpTeX supports usage of the `\font` primitive and allows you to simply incorporate the font selector (declared by `\font`) into your macros to enable scaling this font by the size context used in the document. You need not declare a `\font` for the same font repeatedly for all necessary sizes.

Fragile commands in titles

LaTeX. LaTeX users know the term "fragile command" well. A macro used in the title of a section or chapter can be broken when it is written to internal files used for generating the table of contents. It has been over 20 years that (ε-)TeX has provided the `\detokenize` and `\scantokens` primitives but LaTeX users are still fighting with fragile commands, solving the problem with methods like using the `\protect` macro or `\DeclareRobustCommand` which adds a mysterious space to the end of the control sequence name, making tracing more difficult.

OpTeX. There are no "fragile macros". OpTeX reads the titles for chapters and sections in verbatim mode and re-tokenizes them when they are actually used for printing. Moreover, the verbatim constructions work in titles too:

```
\sec Title with `\this{` matter
```

The in-line verbatim is surrounded by `` `...` `` here. As shown, unbalanced braces `{}` can be in the verbatim text. And this verbatim text is correctly printed in the table of contents, headlines, and PDF outlines. This is difficult in LaTeX because the syntax for section parameters is the text surrounded by balanced braces `{}`. The title parameter in OpTeX is delimited by the end of the line.

ConTeXt. The title parameter is surrounded by braces `{}` similarly as in LaTeX, so the example above with `\this{` cannot work as-is in ConTeXt.

Markup language

LaTeX. Almost all user-level LaTeX macros have undelimited parameters, so users must use braces to specify parameters of more than one token. LaTeX's `\newcommand` does not allow declaring a macro with delimited parameters. Brace pairs `{}` outside the context of macro parameters (i.e. in the meaning begingroup–endgroup) are not preferred in LaTeX markup language.

Writing the LaTeX document often means *coding* the document. There are many nested LaTeX environments (a new syntactic concept in LaTeX) and `{}` braces. For example, Beamer documents are more programming language than source text. The author's ideas (what's intended to be displayed in the Beamer presentation) disappear among the surrounding code in source files.

There is no "standard LaTeX markup language" in the sense that if a program (other than TeX) understands this markup then it knows how to convert from or to this markup. We cannot suppose what packages will be used, and they significantly affect the tagging of the document.

OpTEX. The main credo is: the source files of the document are created (typically) by humans, humans will read these sources and manipulate them. Source files are intended primarily for humans, not for machines. Machines must be programmed to respect these principles.

This is why OpTEX tries to define a more lightweight markup language. There are no highly nested environments, there is a minimal number of braces `{}`. Items in lists begin naturally with a `*` character, titles are terminated by the end of the line.

Many other relatively lightweight markup languages have been devised; Markdown, for example. But we cannot have absolute control of "to PDF processing" in Markdown. Maybe, an author can initially prepare documents in Markdown, but the result must be converted to a `.tex` format, which could use the OpTEX format. A (human) typesetter takes this `.tex` file and starts to program the typography of the document. It means that he or she manipulates the `.tex` sources and adds more information here and adds `.tex` macro files.

The `.tex` sources are "the heart" of the document. They are what's needed to archive documents for future use (in the next edition, for example). They can be processed for printing purposes to PDF (by OpTEX) or they can be converted to other formats (by other converters).

There is an OpTEX markup language standard (OMLS) [16] which gives instructions for potential authors of converters from/to OpTEX format. The standard specifies the syntax and semantics of "known OpTEX tags". Other tags can be ignored. The OMLS covers tags that are used in typical tasks when processing OpTEX documents.

ConTEXt. The markup language is somewhere between LATEX coding and OpTEX writing the document. Tables, though, are created in a special way not typical in TEX. They look like HTML code which is far from a text intended for humans.

Hello world test

Create the following test file

```
\loggingall
\cslang             % Czech hyphenation patterns
\fontfam[Adventor] % Unicode font family Adventor
Ahoj světe!         % Hello world!
\bye
```

and its counterparts in other two formats (LuaLATEX and ConTEXt), i.e. loading the Unicode Adventor family, selecting a language, and printing one sentence. We count the number of lines in the `.log` file when full tracing is activated by `\loggingall` and the time spent when processing this document without `\loggingall` (measured using the author's notebook). The following table summarizes the results:

	OpTEX	LuaLATEX	ConTEXt
lines	1.8 K	3.6 M	231 K
time	0.5 s	1.0 s	1.1 s

We can see that LuaLATEX needs to do about two thousand times more internal operations than OpTEX to process this "Hello world" document. LuaLATEX is slightly better than ConTEXt (although ConTEXt was run with the `--once` option) in the time measurement, because ConTEXt has an exceptionally large `.fmt` file and time is spent loading and uncompressing this file. See the following table which shows the size of `.fmt` files in the GNU/Linux TEX Live distribution:

	OpTEX	LuaLATEX	ConTEXt
`.fmt` size	750 KB	1.2 MB	11 MB

Note that OpTEX has a noticeably smaller format file than LuaLATEX although it implements not only the comparable features of the LATEX kernel but also a number of LATEX packages not included in the kernel: `xcolor`, `hyperref`, `url`, `listings`, `biblatex`, `graphicx`, `geometry`, `amsmath`, `amsymb`, `fontspec`, `unicode-math`, `cprotect`, `eqparbox`, `tabularx`, `booktabs`, `keyval` and more.

Why it is possible? OpTEX keeps its basic concept "simplicity is power".

Moreover, OpTEX does not create any auxiliary file if it is not needed. When the example above was processed, LATEX creates an unnecessary `.aux` file and ConTEXt creates a `.tuc` file, but OpTEX creates only `.pdf` and `.log` files.

When OpTEX needs an auxiliary file, then only a `.ref` file is created where all needed information is saved. We don't need `.toc`, `.lot`, `.lof`, `.nav`, `.glo`, `.idx`, `.ind`, `.bbl` and others that have been used over the years.

Namespaces

LATEX. There is no user namespace. You cannot tell users: "you can define and use an arbitrary control sequence with given naming scheme". There is only the concept: "try it and maybe you will be successful". More precisely, the user can define an arbitrary control sequence using `\newcommand` and this macro ends with the error "control sequence defined already" (in other words: "you are out of your namespace"), or it is successfully defined.

LaTeX packages commonly use internal names containing the `@` character. These names typically begin with `pkg@` where `pkg` is the name of the package. The `expl3` language uses `_pkg_` in the names of control sequences too. This results in code where we repeatedly see `_pkg_`, `_pkg_`, `_pkg_` in practice, despite various language features to attempt to reduce the need for the repetition. This reduces the clarity of the code and reduces the concentration of an eventual reader on the key topic of the code.

OpTeX. We can say to users: Your namespace includes names with letters only. You can define control sequences in this namespace and use them. Moreover, some control sequences in your namespace have pre-declared meanings (primitive, macro from OpTeX, register or anything else). You can use them. If you don't know about the existence of the meaning of a pre-declared control sequence and you never use it with this meaning, then you can re-define it without problems. For example, `\def\fi{finito}` will work as you wish, if you never use `\fi` in its primitive meaning.

How does it work? All primitive control sequences and internal macros in OpTeX have their duplicates in the format `\_foo`. For example, there are control sequences `\hbox` and `\_hbox` with the same meaning. OpTeX macros use exclusively the `\_foo` form. This is the OpTeX namespace. Users can utilize `\_foo` sequences too without problems in "read-only" mode and they can re-define such sequences when they know what they are doing.*

The packages for OpTeX have their own namespace in the naming scheme `\_pkg_foo`. But the package writer doesn't have to write `_pkg_` repeatedly in the internal macros because there is a `\_namespace{pkg}` declaration. If it is used then the macro programmer can write `\.foo`, `\.bar` in the code, and it is transformed to `\_pkg_foo`, `\_pkg_bar` at the input processor level, when the macro file is scanned.

Happy (Op)TeXing!

* Unfortunately, there is one exception to this principle of the user namespace: the control sequence `\par` is hardwired in the TeX implementation. For example, it is generated at each empty line. The OpTeX manual mentions this exception from the user namespace. I wrote a patch to TeX, which enables to set any name to this hardwired control sequence. Unfortunately, I sent this patch after the deadline for TeX Live 2021, so we must wait a while until it will be implemented in the TeX engines. The patch is ready, documented, and tested on my computer. When the patch is applied then this footnote will be rendered obsolete, since there will be no exceptions from the user namespace.

References

1. OpTeX web pages. `petr.olsak.net/optex`
2. P. Olšák: OpTeX — A new generation of Plain TeX. *TUGboat* 41:3, 2020, pp. 348–354. `tug.org/TUGboat/tb41-3/tb129olsak-optex.pdf`
3. L. Lamport: *LaTeX: A document preparation system*. Reading, Mass.: Addison-Wesley, 1994.
4. OPmac web page. `petr.olsak.net/opmac-e.html`
5. P. Olšák: OPmac: Macros for Plain TeX. *TUGboat* 34:1, 2013, pp. 88–96. `tug.org/TUGboat/tb34-1/tb106olsak-opmac.pdf`
6. OpTeX tricks. `petr.olsak.net/optex/optex-tricks.html`
7. LaTeX Project web pages. `https://www.latex-project.org/`
8. F. Mittelbach et al.: *The LaTeX Companion*. Reading, Mass.: Addison-Wesley, 2004.
9. OpTeX manual. `petr.olsak.net/ftp/olsak/optex/optex-doc.pdf`
10. P. Olšák: TeX in a nutshell. 2020, 29 pp. `petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf` `ctan.org/pkg/tex-nutshell`
11. P. Olšák: Typesetting Math with OpTeX. `petr.olsak.net/ftp/olsak/optex/optex-math.pdf`
12. V. Eijkhout: *TeX by Topic*, 2007. `ctan.org/pkg/texbytopic`
13. D. Knuth: *The TeXbook*. Reading, Mass.: Addison-Wesley, 1984.
14. LuaTeX reference manual. `www.pragma-ade.com/general/manuals/luatex.pdf`
15. Font catalog generated by OpTeX. `petr.olsak.net/ftp/olsak/optex/op-catalog.pdf`
16. OpTeX markup language standard (OMLS). `petr.olsak.net/ftp/olsak/optex/omls.pdf`

◇ Petr Olšák
Czech Technical University
in Prague
`https://petr.olsak.net`

GUST e-foundry font projects, closing report 2019–2020

Jerzy Ludwichowski

1 For the record

The GUST e-foundry's set of interrelated projects that are reported on here was conceived in 2015. A leaflet presenting the ideas and asking for financial support was sent out to various TEX LUG boards later that year. Support was offered in 2015 by NTG, in 2016 by CSTUG and ConTEXt Group. DANTE e.V. and TUG joined in 2017.

The "advertising" leaflet mentioned above was turned into a one page summary and published in *TUGboat*, Volume 38 (2017), No. 2 as "GUST e-foundry current font projects" (`tug.org/TUGboat/tb38-2/tb119ludwichowski.pdf`).

The official start of the project was never declared, but it seems that 2017 is a good number. However, work was being done already in 2016.

2 What was planned

The main goal of those projects was to add mathematical, technical and geometrical symbols to all of the TEX Gyre text fonts with the exception of TG Chorus. TG Chorus was excluded as such symbols seem of little use in a chancery font.

Further, several related ideas were coined:

- a sans-serif math OTF font, possibly based on DejaVu, for use in headings;
- a heavy math OTF font, possibly based on TG Termes, also for headings;
- a monospace text font with math symbols, for use in text editors.

Two other goals were also set:

- enhancements to existing math fonts, like math kerns, variant extra alphabets (e.g., calligraphic or double-struck) implemented using the "stylistic set" features `ss01`–`ss20`;
- continuous, yearly maintenance reviews and, if needed, releases of e-foundry's fonts with fixes.

3 Stage 1: What was done through 2018

The outcome of the part of the project that might be called its first stage was described in the paper by B. Jackowski, P. Pianowski, and P. Strzelczyk "TEX Gyre text fonts revisited", published both in *TUGboat*, Volume 39 (2018), No. 3 (`tug.org/TUGboat/tb39-3/tb123jackowski-gyre.pdf`) and *Die TEXnische Komödie*, 30. Jahrgang, Heft 3/2018.

This is a crude summary of what was done (for details see the article):

- devising the enhanced repertoire of glyphs;
- elements of MetaType1 (`en.wikipedia.org/wiki/METATYPE1`) were reimplemented by replacing `t1utils` and some AWK and Perl scripts with Python code interfacing to FontForge — both more portable and easier to maintain;
- the internal structure of the TG fonts became even more OTF-like:
  - — the `ss10` feature allows the use of the original math symbols if replacements are not liked or needed; and
  - — the "anchors" mechanism based on the `ccmp`, `mark` and `mkmk` features is used to place accents over glyphs in a precise way;
- the improved MetaType1 was used to extend the list of glyphs of TG Adventor and TG Pagella by over 850 items, which took the fonts to version 2.501.

4 Stage 2: Algotype, the successor to MetaType1, 2019–2020

After releasing the new versions of TG Adventor and Pagella, the team decided to attempt a full reimplementation of MetaType1.

It is important to notice that up to now, for over 20 years, all of the many e-foundry's fonts were produced with MetaType1. This program began in late nineties of the twentieth century as a no-name engine to create Adobe PostScript Type 1 outline fonts for Janusz M. Nowacki's efforts to revive the traditional Polish type Antykwa Półtawskiego and was reported at the Heidelberg EuroTEX Conference in 1999 ("Antykwa Półtawskiego: a parameterized outline font", `https://jmn.pl/biblio/02ap.pdf`).

A few years later OpenType became an ISO standard (ISO Standard ISO/IEC 14496-22, Part 22: Open Font Format, March 2007; updated 2019). Naturally, the program by the name MetaType1 has been adapted and OpenType versions could be included in the TEX Gyre collection of fonts (released in 2006–2007).

Another adaptation of MetaType1 became necessary with the advent of OpenType Math fonts when in 2010 Microsoft implemented math font support into MS Office. MetaType1 proved itself by generating the TG Math fonts: Bonum, Pagella, Schola, Termes and later DejaVu. The engine was also used by the e-foundry team for Latin Modern fonts in both Type 1 and OpenType formats, along with the LM OpenType math font.

All of these changes, accumulated over so many years, have lead inevitably to MetaType1 becoming rather unwieldy and complex. In particular, porting of the system became a nightmare, which was experienced when Marek Ryćko had to step in for Piotr Strzelczyk who left the team in early 2019 and MetaType1 had to be installed from scratch in a different environment.

This departure of Piotr Strzelczyk was a severe blow and forced a drastic change in priorities: nothing became more important than a reimplementation and redesign of the font production line. At BachoTEX 2019, the article "Redesign of a MetaPost-based font generating system" by Marek Ryćko and Bogusław Jackowski, presented by Marek Ryćko was awarded the W. J. Martin Prize.

MetaType1 was rewritten in such a way that only MetaPost and Python 3 (with some pieces of Python 2 to communicate with the FontForge library) are used. Moreover, a new way of configuring the system was worked out — the configuration is now governed by simple, universal data files (in JSON format). Exactly the same scripts can be run both under GNU/Linux and Windows (no tests with Macintosh were performed so far), which has solved the portability problem.

The new engine is called Algotype. The name tries to stress that fonts are being defined algorithmically. The Python part of Algotype is now available at `pypi.org`.

The team will publish the Algotype system on GitHub.

5 Current and future font works, 2021–

Immediate future:[1]

- Algotype is being used for production work already, but still requires further effort.
- Enhanced (see the "What was planned" section) TG text fonts Schola and Termes, processed with Algotype, are close to being released together with revised versions 2.501 of TG Adventor and TG Pagella.
- A new release of the Latin Modern fonts with corrections proposed by Frank Mittelbach at BachoTEX 2019.
- 2021 should see the rest of the enhancements to the TEX Gyre family, i.e., the new releases of TG Bonum, TG Cursor and TG Heros.

The renewed team with Marek Ryćko hopes to be able to tackle in the near future the remaining tasks listed in section "What was planned", although we cannot make promises.

[1] It should be noted that the first three items would have already happened if it were not for the COVID-19 pandemic and Bogusław Jackowski being hospitalized for over a month for a COVID-19 infection and then heart surgery.

6 Financing (support) up to date

The following donations to the project were received and paid out up-to-date:

- ConTEXt Group: 1,500 EUR in the years 2017–2019;
- CSTUG: 2,000 EUR in the years 2017–2018;
- DANTE e.V.: 7,000 EUR in 2018;
- NTG: 18,000 EUR in the years 2015–2020;
- TUG: 2,903 USD in 2017;
- individual persons: 1,960 PLN in the years 2017–2019.

The total funding has amounted to 28,500 EUR, 2,903 USD and 1,960 PLN. Donations are always welcome at `tug.org/donate`, as well as through the other user groups.

The GUST e-foundry is deeply grateful to all its supporters and promises to continue its best efforts.

7 Unrelated: Other GUST packages updated

Bonus information not related to the e-foundry, although many of the same people are involved: several packages originating with GUST were updated for the TEX Live 2021 release, for the first time in many years: `cc-pl`, `mex`, `pl-mf`, `plhyphen` and `lm`.

The updates clarify the licensing (mostly public domain), update the source encoding to UTF-8, and other housekeeping matters. There are no notable changes in functionality, except for `lm`: a bug, known since 2015, was fixed in the Latin Modern LATEX support.

8 Final remark: Feedback requested

The gentle readers of this report are kindly asked for feedback: do you like/hate/see faults in/ask for enhancements to/propose fixes to/... the works of the GUST e-foundry, or other packages?

Please write! We will do our best to satisfy your request.

◇ Jerzy Ludwichowski
GUST, Poland
`Jerzy.Ludwichowski (at) gust dot org dot pl`

The NewComputerModern font family

Antonis Tsolomitis

Abstract

We present the NewComputerModern font family (NewCM for short), an extension of the default Computer Modern fonts at (currently) 10pt for Unicode TEX engines.

1 Introduction

Back in the mid-1980s you could hardly typeset anything in a non-Latin language. Thanks to the work of many people and the Babel package, TEX was extended so it could typeset Cyrillic, Greek and many other scripts and languages that needed different fonts and typesetting rules. For Greek in the early nineties it was difficult just to type in Greek (it was a pain merely to set up a Greek keyboard under GNU/Linux even in the late nineties).

Things progressed over the years and the TEX world gained the Unicode engines, promising to solve access to thousands of glyphs outside the Latin blocks. However, we are in 2021 and still: You install a TEX system (any), you start a simple document, you run `xelatex` or `lualatex`, you fire up your PDF reader ... And you realize that the old frustration is still there. There is nothing in the PDF except Latin glyphs.

After all these years I find this not satisfactory, to say the least. There is no fallback mechanism (as there is for Office apps) and the default fonts contain only Latin glyphs (plus math). So, the user must make choices. Select a Cyrillic font, a Greek font, a Hebrew one etc. But the user must know how to do it, and it is not trivial because s/he must find fonts that match in style and weight; and if math is needed the task is even more difficult.

NewCM was born with these thoughts in mind. If TEX is Unicode-enabled, where is its default Unicode font? Should a default font support all the planet's languages? Most probably not. But why not support at least the large communities of spoken languages whose members have a proven interest for their language being supported?

NewComputerModern is an attempt to expand the Latin Modern (`lm`) fonts to common non-Latin scripts, while keeping metric compatibility with `lm`.

2 The `fontsetup` package

So if one (i.e., me) merges in the glyphs from existing fonts is he done? Not at all. Why? Because your new font is not the default, and people will not easily switch away from the system's default. Now we are at the same point. It is not easy to properly set up fonts that support math too. What is needed is a simple way to do that. A "one liner". This need gave birth to the `fontsetup` package: An easy way to properly load fonts and matching accompanying mathematics. The line

```
\usepackage[olddefault]{fontsetup}
```

will load all you need, from the font side, to typeset in all languages covered by the `lm` fonts plus Cyrillic, Greek, Hebrew and Cherokee. All with matching math, sans serif and typewriter fonts. It will also provide access to several other Unicode blocks such as Braille patterns and more, to be discussed below. Of course hyphenation and label strings must be loaded for the main language (for example, the `xgreek` package for Greek).

But why "olddefault"? What is "default" and what are other options?

3 A Book weight for NewCM

An old problem of ComputerModern is the fact that it is a light font. And this problem is the same with Claudio Beccari's Greek which was added in NewCM, and Cyrillic from `cmu`, also added, because a goal for those fonts was to match the weight of Knuth's original fonts.

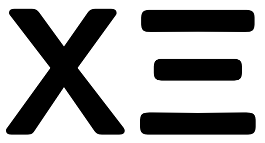
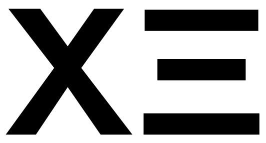
To design heavier fonts from scratch would be a huge undertaking, given the thousands of glyphs involved. To do it automatically with a font editor is known to create problems with the glyphs. But there is a catch with that last sentence. We do not want to create bold versions with the font editor. That would be bad. We just need a little bit of heaviness to be added so that the fonts look good at both low and high resolutions; the existing bold can stay untouched.

So this gave birth to NewCM Book weight. Was it as simple as it sounds? No, because we need math too. So the Book weight math font must carry all the information needed to properly typeset math, and this is many weeks of work for just one font. But in the end, there we have it: this is the "default" version of the `fontsetup` package. That is the Book weight for NewCM, supporting all languages the project supports, and all the features that will be presented below. The amount of added "boldness" is such that it matches in color with the GFSDidot family, which I have used in my books in the past and which looks good at both high resolution printing and low resolution screens.

The `fontsetup` package has more options to easily load many other font families with matching math. Please check its documentation (`ctan.org/pkg/fontsetup`).

4 Bold Sans

The BoldSans in the `lm` and `cmu` fonts is merely a stroke-extension of the Sans, with rounded corners. NewCM provides a true BoldSans:

LM	NewCM
	

This currently covers Latin and Greek, but soon it will cover Cyrillic too.

5 New languages added

Cyrillic has been added from the `cmu` package and Greek monotonic and polytonic from Claudio Beccari's fonts.

Greek:

Θεώρημα 5.1 (Πυθαγόρειον) *Ἐν τοῖς ὀρθογωνίοις τριγώνοις τὸ ἀπὸ τῆς τὴν ὀρθὴν γωνίαν ὑποτεινούσης πλευρᾶς τετράγωνον ἴσον ἐστὶ τοῖς ἀπὸ τῶν τὴν ὀρθὴν γωνίαν περιεχουσῶν πλευρῶν τετραγώνοις.*

Russian:

Я помню чудное мгновенье:
Передо мной явилась ты,
Как мимолетное виденье,
Как гений чистой красоты.
(Пушкинъ)

Hebrew and Cherokee were designed from scratch:

A few letters from Hebrew:

צּלּשׂשׁוהדגבא

A few letters from Cherokee:

ᏧᎤᏃᏯᏰ ᏧᎤᏫᏩᏭᏯ

Back to Greek, Small Caps is included (in Mono font too) and all polytonic accents of Greek. Ypogegrammeni is the default for all characters including Small Caps and prosgegrammeni is offered as an alternative shape in the `ss01` lookup table:

	ypogegrammeni	prosgegrammeni
regular	ᾋ ᾞ ᾯ ᴀͅʜͅῼ	ᾋ ᾞ ᾯ ᾼῌῼ
sans	ᾋ ᾞ ᾯ ᴀͅʜͅῼ	ᾋ ᾞ ᾯ ᾼῌῼ
mono	ᾋ ᾞ ᾯ ᴀͅʜͅῼ	ᾋ ᾞ ᾯ ᾼῌῼ

The prosgegrammeni alternates can be accessed with commands from the `fontsetup` package, either of:

`\textprosgegrammeni{`⟨*text*⟩`}`
`{\prosgegrammeni `⟨*text*⟩`}`

6 More Unicode blocks

Braille, both 6dot (uni2801–uni283F) as well as 8dot (uni2840–uni28FF) patterns are included in two versions. The Regular font provides the characters for sighted persons (such as teachers) so they can easily see which dots are on and which off. The Sans font contains the true Braille characters. I decided to have the sighted version in the Regular font since a blind person does not need the real Braille pattern, as those are produced by embossers. The Braille patterns here are meant for use as fonts to typeset text mainly for sighted persons.

	6dot	8dot
Regular version	⠅ ⠆ ⠇ ⠝ ⠞	⡂ ⡃ ⡄ ⡚ ⡛
Sans version	⠅ ⠆ ⠇ ⠝ ⠞	⡂ ⡃ ⡄ ⡚ ⡛

IPA symbols are included, and following a suggestion of Huanyu Liu the kerning found in the `tipa` package has been added here and further improved. Moreover the letters eth, eng, beta, theta and chi exist in IPA-style in the fonts and are accessible in the ss05 lookup table since they have a different design than the Latin and Greek letters. You can access this lookup table using the `\textipa` and `\textsansipa` commands of the `fontsetup` package.

	Non-IPA	IPA
Regular	ð ŋ β ϑ χ	ð ŋ β θ χ
Sans	ð ŋ β θ χ	ð ŋ β θ χ

7 Ligatures and stylistic alternatives in Latin

The Serif font includes additional ligatures fb ffb ffh ffj ffk fft fh fj ft fk, and the same with long-s instead of f in the default liga table (in addition to the default fi fl ffi ffl ff). It also includes an alternative k (in the cv01 table) and sp ch ck ct st in the dlig table. Finally it also includes "end" versions for the letters a, e, m, n and r in the cv02 table.

Regular	k	a e m n r	sp ch ck ct st
cv01	k		
cv02		a e m n r	
dlig			sp ch ck ct st

8 Archaic Greek writing for scholars and others

The Sans Serif Regular font provides access to 6th century BCE and 4th century BCE Greek capitals in ss04 and ss03 lookups. The `fontsetup` package provides commands such as:

`\textivbce{}`, `\ivbce`, `\textvibce{}`, `\vibce`.

6th century BCE:
ΜΗΔΕΙΣ ΑΓΕΩΜΕΤΡΗΤΟΣ ΕΙΣΙΤΩ

4th century BCE:
ΜΗΔΕΙϟ ΑΓΕΩΜΕΤΡΗΤΟϟ ΕΙϟΙΤΩ

Moreover, all fonts (except Mono and Math) support Ancient Greek Numerals (the full Unicode block of Greek digits u10140–u1018E is supported), with most symbols designed from scratch (those that did not exist in C. Beccari's original fonts). A few of the new symbols:

𐅪𐅫𐅬𐅏𐅭𐅇𐆊

The four numerals that already existed in this range (that is u10144–u10147) in Beccari's fonts have been altered to a new design matching the style of cm but also provide some Ancient Greek flair. The new designs in Serifed and SansSerifed are:

𐅄𐅅𐅆𐅇 𐅄𐅅𐅆𐅇

The `fontsetup` package provides commands for all of the above symbols. The commands follow the Unicode name of each slot (minus the "Greek Acrophonic"). So the Unicode slot u1014F named "Greek Acrophonic Attic Five Staters" can be accessed with the command `\atticfivestaters` and gives 𐅏; and the slot u10182 named "Greek Kyathos Base Sign" can be accessed with the command `\greekkyathosbasesign` and gives 𐆂.

9 Old Italic

The fonts also fully support the Old Italic Unicode block (u10300–u1032F) in the Sans font. For example, the slots u10307, u10310, u10312, u10314, u1031F and u1032F are 𐌇𐌐𐌒𐌔𐌟𐌯.

10 Unicode Math coverage and options

NewCM provides full Unicode math support, that is all mathematics Unicode slots presented in `unicode.org/charts/` in both Math weights, Regular and Book. These blocks are:

Mathematical Symbols
- Arrows (uni2190–uni21FF)
- Supplemental Arrows-A (uni27F0–uni27FF)
- Supplemental Arrows-B (uni2900–uni297F)
- Supplemental Arrows-C (u1F800–u1F8FF)
- Additional Arrows (uni2B00–uni2BFF)
- Miscellaneous Symbols and Arrows (uni2B00–uni2BFF)

Mathematical Alphanumeric Symbols (u1D400–u1D7FF)
- Arabic Mathematical Alphabetic Symbols (u1EE00–u1EEFF)
- Letterlike Symbols (uni2100–uni214F)

Mathematical Operators (uni2200–uni22FF)
- Basic operators: Plus, Factorial (uni0000–uni007F)
- Division, Multiplication (uni0080–uni00FF)
- Supplemental Mathematical Operators (uni2A00–uni2AFF)
- Miscellaneous Mathematical Symbols-A (uni27C0–uni27EF)
- Miscellaneous Mathematical Symbols-B (uni2980–uni29FF)
- Floors and Ceilings (uni2308–uni230B)
- Invisible Operators (uni2061–uni2064)

Geometric Shapes (uni25A0–uni25FF)
- Additional Shapes (uni2B00–uni2BFF)
- Box Drawing (uni2500–uni257F)
- Block Elements (uni2580–uni259F)
- Geometric Shapes Extended (u1F780–u1F7FF)

Unfortunately, the `unicode-math` package does not currently provide commands for the hundreds of extra glyphs that have been added in order to fully cover the above Unicode ranges. The user can consult the Unicode charts at the above link and access the required glyph with `\char"#` where # is the Unicode number of the slot the glyph belongs to. For example, `\char"2BDA` will give the Hygeia symbol (uni2BDA) the Rod of Asclepius as shown above.

10.1 Upright and extensible integrals

The Math fonts (both Regular and Book weights) include upright integrals in the ss02 StylisticSet.

Normal integrals	Upright integrals
$\int \iiint \displaystyle\int \oint_a^b$	$\int \iiint \displaystyle\int \oint_a^b$

Use with

```
\setmathfont[StylisticSet=2]
                              {NewCMMath-Book.otf}
```

or

```
\setmathfont[StylisticSet=2]
                           {NewCMMath-Regular.otf}
```

or use the `upint` option of the `fontsetup` package; for the Book weight:

```
\usepackage[upint,default]{fontsetup}
```

or for the regular weight:

```
\usepackage[upint,olddefault]{fontsetup}
```

Moreover, extensible integrals are supported by the fonts but *not* by the Unicode TeX engines. The following code is a trick so that extensible integrals can be constructed using LuaLaTeX. The result is shown at the end of the article. What the code below does is define the slot uni222B (Integral) as a delimiter. Then, this is extended as a delimiter with the mechanism that the font provides.

```
\documentclass{article}
\usepackage[default]{fontsetup}
\begin{document}
\[
\Uleft\Udelimiter 0 0 "222B
\begin{pmatrix}
  1
\end{pmatrix}
\Uright.
\rightarrow
\Uleft\Udelimiter 0 0 "222B
\begin{pmatrix}
  1\\2
\end{pmatrix}
\Uright.
\rightarrow
\Uleft\Udelimiter 0 0 "222B
\begin{pmatrix}
  1\\2\\3\\4
\end{pmatrix}
\Uright.
\rightarrow
\Uleft\Udelimiter 0 0 "222B
\begin{pmatrix}
  1\\2\\3\\4\\5\\6
\end{pmatrix}
\Uright.
\]
\end{document}
```

10.2 Non-Unicode symbols

It seems that Unicode has forgotten to include slots for the negation of uniform convergence. The fonts include two extra slots for $\not\rightrightarrows$ and $\not\leftleftarrows$ that can be accessed in math mode with the commands `\nrightrightarrows` and `\nleftleftarrows` of the `fontsetup` package.

Unicode seems to have also forgotten to include MathSansGreek. These are included in the Math fonts and they are and accessible with commands such as `\msansAlpha` or `\mitsansAlpha`.

$$\mathsf{AB\gamma\delta} \quad \mathsf{\mathit{AB\gamma\delta}}$$

11 Future work

The immediate plans for NewCM are to provide the fonts at 8pt and to provide support for accent (diacritics) stacking. Work on the 8pt version has already begun. The 8pt size of the `lm` fonts looks lighter at 8pt than the 10pt font. This should not happen in my opinion, so the 8pt design (set at 8pt) will match the weight of the 10pt design in the latin glyphs too. Nonetheless, metric compatibility will be preserved.

12 Thanks

There are many people I would like to thank who have reported bugs of the fonts. Special thanks go to Karl Berry, Claudio Beccari, David Carlisle, Robert Alessi, Huanyu Liu and Manuel Boni for supporting this project with their help and suggestions.

◇ Antonis Tsolomitis
Department of Mathematics
University of the Aegean
Karlovassi, 832 00 Samos
Greece
atsol (at) aegean dot gr
https://myria.math.aegean.gr/~atsol/

$$\int\begin{pmatrix}1\\2\end{pmatrix} \rightarrow \int\begin{pmatrix}1\\2\\3\\4\end{pmatrix} \rightarrow \int\begin{pmatrix}1\\2\\3\\4\\5\\6\end{pmatrix} \rightarrow \int\begin{pmatrix}1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\end{pmatrix} \rightarrow \int\begin{pmatrix}1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\end{pmatrix}$$

An attempt at creating font transitions

S.K. Venkatesan

Abstract

In this short note we discuss the abstract topological aspects of fonts and how a simple homotopy deformation can be created to transition a glyph in one font to that of another font. We demonstrate interesting transitions for some fonts from sans-serif to serif using a JavaScript implementation of a simple algorithm. We also discuss other cases which involve deformations that map non-homeomorphic configurations that involve catastrophic bifurcations.

1 Introduction

Donald Knuth in his METAFONT project [1] wanted to initiate a framework for creating fonts through computer programs. An essay in Douglas Hofstadter's book *Metamagical Themas: Questing for the essence of mind and pattern* [2] challenges this ambitious initiative and discusses it in great detail. A simple question is, can a computer program automatically create hybrid fonts, for example, a new font that is 30% like Helvetica and 70% like Times? We answer this question using a simple algorithm and demonstrate its ramifications for some fonts.

2 Algebraic topology of font glyphs and their deformations

Topology is usually defined as the study of clay geometry, explained through an example of a teacup being topologically equivalent to a ring (the handle of the teacup). The crucial idea is that you can remold the clay without breaking it anywhere through any continuous deformation, unlike in Euclidean geometry where lengths are also preserved.

Font glyphs are, for the most part, connected compact two-dimensional manifolds with simply-connected one-dimensional boundary curves. Although most glyphs are simply-connected, there are also some glyphs such as lowercase "i" and "j" that have two disconnected pieces of manifold, due to the dot that is disjoint from the main shape. In any case, each connected piece is a connected compact two-dimensional manifold with a boundary.

Typically, in the English uppercase alphabet, C,E,F,G,H,I,J,K,L,M,N,S,T,U,V,W,X,Y,Z and in the lowercase, c,f,h,k,l,m,n,r,s,t,u,v,w,x,y,z are glyphs that are two-dimensional manifolds with a simply-connected single boundary which can be homeomorphically mapped to a two-dimensional disc with one-dimensional circle as its boundary (we could have just as well used a square instead of circle, as they are topologically equivalent). In this case, it is not difficult to prove that any two such glyphs can deformed from one into another by a simple homotopy deformation. In any case, we will be constructing an algorithm that does this, so we prove it by example.

In other glyphs there are holes in the manifolds; as in the case of O, Q, A, etc. In some cases, as in B, there are two holes. So the glyphs can have more than one boundary as well. But in all cases the boundary is a simply-connected one-dimensional curve. As a special trivial case of the Poincaré conjecture,[1] we have that any simply-connected one-dimensional curve is homeomorphically equivalent to a circle. In fact, there is only one one-dimensional manifold (without boundary), the circle.

3 Description of the algorithm used for font deformation

We shall here briefly describe the first algorithm used to map the glyph of one font into another.

3.1 Algorithm 1

Step 1 We center a font glyph into a standard font box of 1em height.

Step 2 We identify the boundary curves of the font.

Step 3 We divide the curve into equal segments along the path of the curve and create a fixed number of an array of points.

Step 4 We reorder the points of the array so that the one that is nearest to the origin appears first.

Step 5 We do steps 1–4 for both the font glyphs.

Step 6 We now transition from one set of arrays to another using a linear path in time, to show the transition.

3.2 Algorithm 2

Steps 1–4 Follow as in Algorithm 1

Step 5 We first segment the sans-serif font glyph.

Step 6 We now segment the serif font glyph into finer (typically 10 times more) segments than the sans-serif font glyph.

Step 7 We now look for an orthonormal projection (due to symmetry requirements we may have to use x-projections and y-projections as well) from sans-serif (as sans-serif is more regular-shaped) font glyph onto serif font to locate the one-to-one correspondence.

Step 8 Once the points have been paired we now transition from one set of arrays to another using a linear path in time, to show the transition.

[1] It is no longer a conjecture, as Grigori Perelman has proved the conjecture for the hitherto unproven three-dimensional case [5].

3.3 Some hints on filling the area inside the curve

The curve that has the longest length needs to be filled inside, while the rest of the curves should not be filled inside (except for the case of the dot for "i" and "j"). In order to identify which piece of curve maps into which, we use the length and the centroid of the curves as the important parameters to guide in our choice.

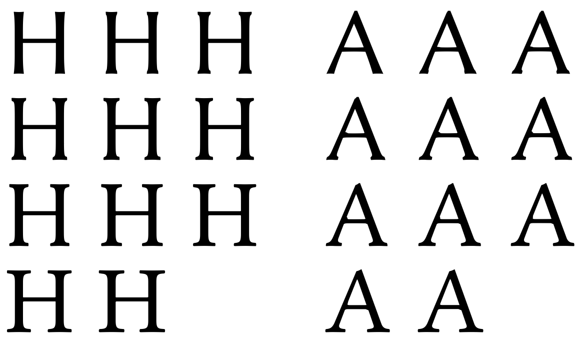
To identify that the dot in "i" and "j" needs to be filled, we need to find out if the other curves are inside the main curve or not, a non-trivial exercise. In fact, it took more than a century to prove the Jordan curve theorem [3], i.e., that a one-dimensional closed curve embedded in a two-dimensional Euclidean plane divides the plane into two disconnected pieces (the inside and outside), but for approximately convex shapes this identification is not that difficult.

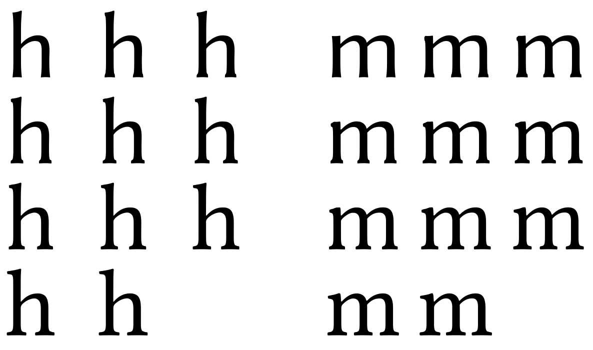
4 The Demo site and JavaScript algorithm

The demo site is here:
`cqrl.in/dev/font-transition-js.html`

The JavaScript code, released under GPLv3, is here: `github.com/Sukii/font-transition`

5 Transition from Linux Biolinum Regular to Linux Libertine Regular





Now let us typeset some text with this transition font to see if it is reasonable:

100% sans-serif:
Quick Brown Fox Jumped High
Over The Lazy Dog

70% sans-serif, 30% serif:
Quick Brown Fox Jumped High
Over The Lazy Dog

30% sans-serif, 70% serif:
Quick Brown Fox Jumped High
Over The Lazy Dog

100% serif:
Quick Brown Fox Jumped High
Over The Lazy Dog

6 The pessimism of Hofstadter and the confidence of Knuth

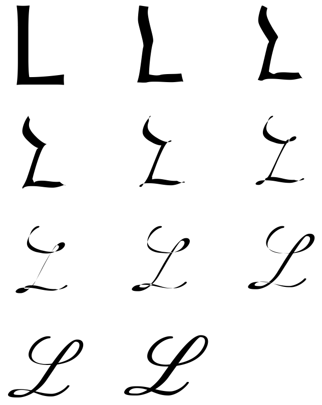
It is true that there are limits to what is feasible; for example, Einstein showed, using the principle of relativity, that the speed of light limits the speed of particles, as it would require infinite energy to cross that barrier. However, we also know that quantum tunnels break through such impossible barriers, as in the case of the Hawking tunneling effect, by which we are able to visualize black holes. More simply, X-rays are not visible to human eyes but they can be discerned through other means such as photographic plates. So limits of human perception and achievements can be extended by other means, as we gain better perception of the problems.

We have shown in this article, both theoretically and through demonstration of font transitions that it is possible in some effective way to achieve what Donald Knuth envisaged. In this article, we restricted ourselves to transition from one font to another only in the case where both are topologically equivalent (i.e., homeomorphic), but it is possible to go beyond that into other complicated designs such as script fonts and gothic (Fraktur, etc.) fonts, i.e., create a transition effect from a sans-serif font glyph to a script or gothic font glyph. This cannot be achieved by a simple homotopy deformation. It requires creation of bifurcation effects and other catastrophes studied by Rene Thom [4], V.I. Arnold [5], and many others.

Instead of meandering into profundity, we shall now discuss a concrete example. Consider the following calligraphic character L:

$$\mathrm{L} \longrightarrow \mathscr{L}$$

generated using the `calrsfs` LaTeX package. Let us see what the transition from the sans-serif "L" in Linux Biolinum to this script L might look like:



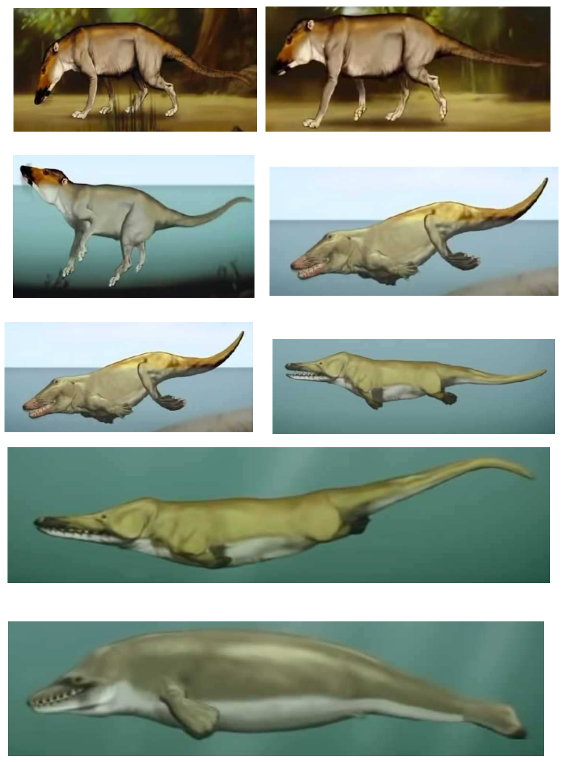
However, creating this transition was not as easy as the previous cases. The "script L" has three boundary curves while "sans-serif L" has only one boundary curve. But, if we carefully observe the pen stroke as it flows through the glyph, we can see it is just one stroke but self-intersecting at two places. This can be considered as a two-dimensional ribbon embedded in (2+1)-dimension (like a twisting and turning fly-over we see in big cities).

Unfortunately, the font files do not store this self-intersecting curve. Instead, the font files stores it as three boundary curves. In order to create the above transition we exported the glyph from the font file as three SVG curves, welded them together as a single curve that self-intersects at two junctions but flows unobstructed through the channels. Unfortunately, if we try to import this back into FontForge as a glyph and export it as a font, it doesn't work.

This describes the flow of the pen in (2+1)-dimension paper (the third dimension being time). To explain this I always mention this joke: There were once a big truck and a bus speeding along a highway at the same physical point on the road but didn't collide. How is it possible? Answer: The bus travelled in the morning, while the truck travelled at night.

7 Evolution of a whale from a mouse-deer

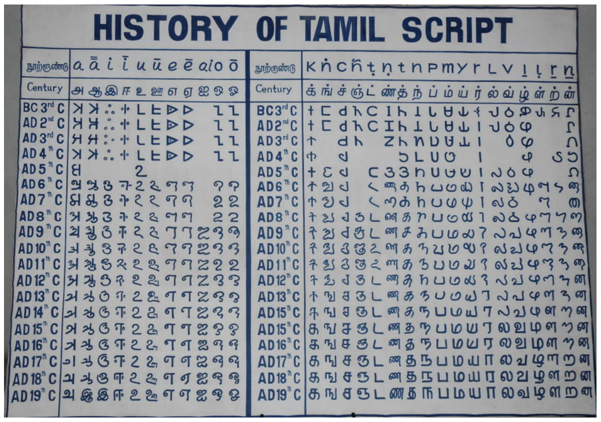
Here we explain the complexity of morphing of one glyph using the evolution of a land mammal (a mouse-deer) into a whale, giving a classical Darwinian angle to it:



In the above transition, one can observe that all animals retain approximate left-right symmetry in their external form. This left-right symmetry is imposed by the Darwinian pressure on the external form by the environment that punishes any left-right asymmetry as it decreases survival fitness due to impaired physical mobility. We shall see in the next section how such symmetry conditions can be imposed on glyphs to preserve approximate left-right symmetry in some characters in the transition states.

Insects possess some kind of left-right symmetry, but not all life forms have left-right symmetry, e.g., amoeba, octopus, jellyfish or for that matter trees and plants. In the zoo of Latin alphabets, only some characters possess approximate left-right symmetry, like H, A, M, W, but they each have their own unique shape and some internal symmetry and aesthetics of their own.

One can also employ these techniques to study the evolution of scripts over time, e.g., the evolution of Tamil script, as shown here (`wikipedia.org/wiki/Tamil_script`):


HISTORY OF TAMIL SCRIPT
Century
BC 3rd C
AD 2nd C
AD 3rd C
AD 4th C
AD 5th C
AD 6th C
AD 7th C
AD 8th C
AD 9th C
AD 10th C
AD 11th C
AD 12th C
AD 13th C
AD 14th C
AD 15th C
AD 16th C
AD 17th C
AD 18th C
AD 19th C

8 Machine learning approaches for fonts

An interesting broad question is whether one can create entirely new fonts out of old ones, using approaches like genetic algorithms. Here we have shown how one can create intermediate transition states from two sans-serif/serif pairs. In theory, it is also possible to create transition states using more than two fonts as well, i.e.,

$$\text{Font}_{\text{new}} = f(\text{Font}_1, \text{Font}_2, \text{Font}_3, \ldots)$$

One interesting problem that has been studied well is the aspect of character recognition (OCR), i.e., recognition of a glyph as an avatar of a particular character of a Latin alphabet. This in a way recognizes in essence what a character is, but it is more in relation to what other characters are and does not capture the essence of what a character is. The question is whether machine learning is capable of doing this. Hofstadter [2] explains how this is a notoriously difficult problem as there many avatars like the Gothic black letters that are so different in form.

As we showed here for the case of “Script L”, it is possible to iron out some of the twists and turns of the flow of the pen by studying them as regular manifolds in (2+1)-dimension, that when projected onto a two-dimensional paper produces self-intersecting curves. We shall endeavor to bring out some of the qualitative features of the alphabets and how they can be defined using some abstract topological concepts in mathematics in (2+1)-dimension in future work.

Recent techniques in one-shot machine learning techniques [7] also hold genuine promise. This uses a function called “triplet loss” that trains on three images: an anchor image, a positive image, and a negative image. The neural network adjusts the parameters so that the features for the anchor and positive image are near while that of the negative image is far off. However, as we mentioned there are internal approximate symmetries (like the left-right symmetry) that need to be discovered and preserved, a genuinely difficult problem.

Of course, not all gyphs produced by an algorithm through machine learning will be usable as glyphs by humans. Some have ugly overlaps, as can be easily discerned, but the interesting question is: Is there always a perfect transition path from sans-serif to serif that avoids those pitfalls? Felix Klein wanted to define concepts of beauty through aspects of geometrical symmetry in an object and recently Roger Penrose also has been discussing these aspects in physics as well. The interesting aspect of human art (thinking of fonts and glyphs as an art form) is that it contains within it seeds of that fundamental question of what it is to be human. This may be a philosophical question that lies outside the realm of science, but our endeavor is to find those aspects that lie within the capacity of a scientific enquiry by trying to transcend human limitations through machines.

References

[1] Knuth, Donald E. (1982) The Concept of a Meta-Font, *Visible Language*, Vol. 16, No. 1, pp. 3–27. `visiblelanguage.herokuapp.com/issue/61`

[2] Hofstadter, D. (1996) *Metamagical Themas: Questing for the essence of mind and pattern.* Chapter 13: Comments on Donald Knuth’s Article ‘The Concept of a Meta-Font’. Basic Books.

[3] Alexander, J.W. (1920) A proof of Jordan’s Theorem about a simple closed curve. *Annals of Mathematics*, Vol. 21, No. 3, pp. 180–184. `doi.org/10.2307/2007256`

[4] Thom, R. (1972) *Structural Stability and Morphogenesis.* W.A. Benjamin.

[5] Arnold, V.I. (1984) *Catastrophe Theory.* Springer-Verlag.

[6] Tao, T. (2006) Perelman’s proof of the Poincaré conjecture: A nonlinear PDE perspective. `arxiv.org/abs/math/0610903` See also: `claymath.org/millennium-problems-poincar%C3%A9-conjecture/perelmans-solution`

[7] Dickson, B. (2020) What is one-shot learning? `bdtechtalks.com/2020/08/12/what-is-one-shot-learning`

◇ S.K. Venkatesan
TNQ Technologies, Chennai
`skvenkat (at) tnqsoftware dot co dot in`

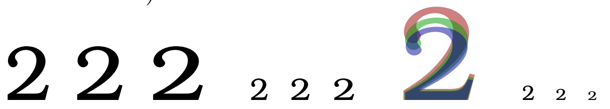
Scaled fonts and glyphs

Hans Hagen

1 History

The infrastructure for fonts makes up a large part of the code of any TEX macro package. We have to go back in time to understand why. When TEX showed up, fonts were collections of bitmaps and measures. There were at most 256 glyphs in a font and in order to do its job, TEX needed to know (and still needs to know) the width, height and depth of glyphs. If you want ligatures it also needs to know how to construct them from the input and when you want kerning there has to be additional information about what neighboring glyphs need a kern in between. Math is yet another subtask that demands extra information, like chains of glyphs that grow in size and if needed even recipes of how to construct large shapes from smaller ones.

Fonts come in sizes. Latin Modern and the original Computer Modern, for instance, have quite a few variants where the shapes are adapted to the size. This means that when you need a 9pt regular shape alongside a 12pt one, two fonts have to be loaded. This is quite visible in math where we have three related sizes: text, script and scriptscript, grouped in so called families. When we scale the digit 2 to the same height you will notice that the text, script and scriptscript sizes look different (the last three are unscaled):



Plenty has been written (in various documents that come with ConTEXt) about how this all works together and how it impacts the design of the system, so here I just give a short summary of what a font system has to deal with.

- In a bodyfont setup different sizes (9pt, 10pt, 12pt) can have their own specific set of fonts. This can result in quite a number of definitions that relate to the style, like regular, bold, italic, bold italic, slanted, bold slanted, etc. When possible loading the fonts is delayed. In ConTEXt often the number of fonts that are actually loaded is not that large.
- Some font designs have different shapes per bodyfont size. A minor complication is that when one is missing some heuristic best-match choice might be needed. Okay, in practice only Latin Modern falls into this category for ConTEXt. Maybe OpenType variable fonts can be seen this way, but, although we supported that right from the start, I haven't noticed much interest in the TEX community.
- Within a bodyfont size we distinguish size variants. We can go smaller (x and xx), for instance when we use sub- and superscripts in text, or we can go larger, for instance in titles (a, b, c, d, ...). Fortunately most of the loading of these can be delayed too.
- When instances are not available, scaling can be used, as happens for instance with 11pt in Computer Modern. Actually, this is why in ConTEXt we default to 12pt, because the scaled versions didn't look as nice as the others (keep in mind that we started in the age of bitmaps).
- Special features, such as smallcaps or oldstyle numerals, can demand their own definitions. More loading and automatic definitions can be triggered by sizes needed in, e.g., scripts and titles.
- A document can have a mixed setup, that is: using different font designs within one document, so some kind of namespace subsystem is needed.
- In an eight-bit font world, we not only have text fonts but also collections of symbols, and even in math there are additional symbol collections. In OpenType symbols end up in text fonts, but there we have tons of emojis and color fonts. All has to be dealt with in an integrated way. And we're not even talking of virtual fonts, (runtime) MetaPost generated fonts, and so on.
- In traditional eight-bit engines, hyphenation depends on a font's encoding, which can require loading a font multiple times in different encodings. This depends on the language mix used. A side point is that defining a European encoding covering most Latin languages was not that hard, especially when one keeps in mind that many eight-bit encodings waste slots on seldom used symbols, but by that time OpenType and Unicode input started to dominate.
- In the more modern OpenType fonts combinations of features can demand additional instances: one can think of language/script combinations, substitutions in base mode, special effects like emboldening, color fonts, etc.
- Math is complicated by the fact that in traditional TEX, alphabets come from different fonts, which is why we have many so-called families; a font can have several alphabets which means that some mapping can be needed. Operating on the size, shape, encoding and style axes puts some demands on the font system. Add to this the (often) partial (due to lack of fonts) bold support and it gets even more complicated. In OpenType all the alphabets come from one font.

- There is additional math auto-definition and loading code for the sizes used in text scripts and titles.

All this has resulted in a pretty complex subsystem. Although going OpenType (and emulated OpenType with Type 1 fonts as we do in MkIV) removes some complications, like encodings, it also adds complexity because of the many possible font features, either dependent or not on script and language. Text as well as math got simpler in the TeX code, though that was traded for quite a bit of Lua code to deal with new features.

So, in order to let the font subsystem not impact performance too much, let alone extensive memory usage, the ConTeXt font subsystem is rather optimized. The biggest burden comes from fonts that have a dynamic (adaptive) definition because then we need to do quite a bit of testing per font switch, but even that has always been rather fast.

2 Reality

In MkIV and therefore also in LuaMetaTeX (LMTX) more font magic happens. The initial node lists that make up a box or paragraph can get manipulated in several ways and often fonts are involved. The font features (smallcaps, oldstyle, alternates, etc.) can be defined as static (part of the definition) or as dynamic (resolved on the spot at the cost of some overhead). Characters can be remapped, fonts can be replaced. The math subsystem in MkIV was different right from the start: we use a limited number of families (regular, bold, l2r and r2l), and stay abstract till the moment we need to deal with the specific alphabets. But still, in MkIV, we have the families with three fonts.

In the LuaMetaTeX manual we show some math magic for different fonts. As a side effect, we set up half a dozen bodyfont collections: Lucida, Pagella, Latin Modern, Dejavu, the math standard Cambria, etc. Even with delayed and shared font loading, we end up with 158 instances but quite a few of them are math fonts, at least six per bodyfont size: regular and bold (emboldened) text, script and scriptscript. Of course most are just copies with different scaling that reuse already loaded resources. In the final PDF we have 21 subsetted fonts.

If we look at the math fonts that we use today, there is however quite some overlap. It starts with a text font. From that, script and scriptscript variants are derived, but often these variants use many text size related shapes too. Some shapes get alternatives (from the `ssty` feature), and the whole clone gets scaled. But, much of the logic of, for instance, extensibles is the same.

A similar situation happens with large CJK fonts: there are hardly any advanced features involved there, so any size is basically a copy with scaled dimensions, and these fonts can be truly huge!

When we talk about features, in many cases in ConTeXt you don't define them as part of the font. For instance small caps can best be triggered by using a dynamic feature: applied to a specific stretch of text. In fact, often features like superiors of fractions only work well on characters that fit the bill and produce weird side effects otherwise (a matter of design completeness).

When the font handler does its work there are actually four cases: no features get applied (something that happens with, for instance, most monospaced fonts); base mode is used (which means that the TeX machinery takes care of constructing ligatures and injecting kerns); and node mode (where Lua handles the features). The fourth case is a special case of node mode where a different feature set is applied.[1] At the cost of some extra overhead (for each node mode run) dynamic features are quite powerful and save quite a lot of memory and definitions.[2] The overhead comes from much more testing regarding the font we deal with because suddenly the same font can demand different treatments, depending on what dynamic features are active.[3]

Although the font handling is responsible for much of the time spent in Lua, it is still reasonable given what has to be done. Because we have an extensible system, it's often the extensions that takes additional runtime. Flexibility comes at a price.

3 Progress

At some point I started playing with realtime glyph scaling. Here realtime means that it doesn't depend on the font definition. To get an idea, here is an example (all examples are additionally scaled for *TUGboat*):

```
test {\glyphxscale 2500 test} test
```

test **test** test

The glyphs in the current font get scaled horizontally without the need for an extra font instance. Now, this kind of trickery puts some constraints on the font handling, as is demonstrated in the next example. We use Latin Modern because that font has all these ligatures:

[1] We also have so-called plug mode where an external renderer can do the work but that one is only around due to some experiments during Idris Hamid's font development.

[2] The generic font handler that is derived from the ConTeXt one doesn't implement this, so it runs a little faster.

[3] Originally this model was introduced for a dynamic paragraph optimization subsystem for Arabic but in practice no one uses it because there are no suitable fonts.

```
\definedfont[lmroman10-regular*default]%
e{\glyphxscale 2500 ff}icient
ef{\glyphxscale 2500 f}icient
ef{\glyphxscale 2500 fi}cient
e{\glyphxscale 2500 ffi}cient
```

efficient efficient efficient efficient

In order to deal with this kind of scaling, we now operate not only on the font (id) and dynamic feature axes, but also on the scales, of which we have three variants: glyph scale, glyph xscale and glyph yscale. There is actually also a state dimension but we omit that for now (think of flagging glyphs as initial or final). This brings the number of axes to six. It is important to stress that in these examples the same font instance is used!

Just for the record: several approaches to switching fonts are possible but for now we stick to a simple font id switch plus glyph scale settings at the TeX end. A variant would be to introduce a new mechanism where id's and scales go together but for now I see no real gain in that.

4 Math

Given what has been discussed in the previous sections, a logical question would be "Can we apply scaling to math?" and the answer is "Yes, we can!". We can even go a bit further and that is partly due to some other properties of the engine.

From pdfTeX the LuaTeX engines inherited character protrusion and glyph expansions, aka hz. However, where in pdfTeX copies of the font are made that carry the expanded dimensions, in LuaTeX at some point this was replaced by an expansion field in the glyph and kern nodes. So, instead of changing the font id of expanded glyphs, the same id is used but with the applied expansion factor set in the glyph. A side effect was that in places where dimensions are needed, we call functions that calculate the expanded widths on request (as these can change during linebreak calculations) in combination with accessing font dimensions directly. This level of abstraction is even more present in LuaMetaTeX. This means that we have an uniform interface to fonts and as a side effect scaling need be dealt with in only a few places in the code.

Now, in math we have a few more complications. First of all, we have three sizes to consider and we also have lots of parameters that depend on the size. But, as I wanted to be able to apply scaling to math, the whole machinery was also abstracted in a way that, at the cost of some extra overhead, made it easier to work with scaled glyph properties. This means that we can stick to loading only one bodyfont size of math (note that each math family has three sizes, where the script and script sizes can have different, fine tuned, shapes) and just scale that on demand.

Once all that was in place it was a logical next step to see if we could stick to just a single instance. Because in LuaMetaTeX we try to load fonts efficiently we store only the minimally needed information at the TeX end. A font with no math therefore has less data per glyph. Again, this brings some abstraction that helped to implement the one instance mechanism. A math glyph has optional lists of increasing sizes and vertical or horizontal extensibles. So what got added was an optional chain of smaller sizes. If a character has three different glyphs for the three sizes, the text glyph has a pointer to the script glyph which in turn has a pointer to the scriptscript glyph. This means that when the math engine needs a specific character at a given size (text, script, scriptscript) we just follow that chain.

In an OpenType math font the script and scriptscript sizes are specified as percentages of the text size. When the dimensions of a glyph are needed, we just scale on the fly. Again this adds some overhead but I'm pretty sure that no user will notice.

So, to summarize: if we need a character at scriptscript size, we access the text size glyph, check for a pointer to a script size, go there, and again check for a smaller size. We use only what fits the bill. And, when we need dimensions we just scale. In order to scale we need the relative size, so we need to set that up when we load the font. Because in ConTeXt we also can assemble a virtual OpenType font from Type 1 fonts, it was actually that (old) compatibility feature, the one that implements Type 1 based on OpenType math, that took the most time to adapt, not so much because it is complicated but because in LMTX we have to bypass some advanced loading mechanisms. Because we can scale in two dimensions the many (font-related) math parameters also need to be dealt with accordingly.

The end result is that for math we now only need to define two fonts per bodyfont setup: regular and bold at the natural scale (normally 10pt) and we share these for all sizes. As a result of this and what we describe in the next section, the 158 instances for the LuaMetaTeX manual can be reduced to 30.

5 Text

Sharing instances in text mode is relatively simple, although we do have to keep in mind that scaling is an extra axis when dealing with font features: two neighboring glyphs with the same font id and dynamics but with different scales are effectively from different fonts.

Another complication is that when we use font fallbacks (read: take missing glyphs from another font) we no longer have a dedicated instance but use a shared one. This in itself is not a problem but we do need to handle specified relative scales. This was not that hard to patch in ConTEXt LMTX.

We can enforce aggressive font sharing with:

```
\enableexperiments[fonts.compact]
```

After that we often use fewer instances. Just to give an idea, on the LuaMetaTEX manual we get these stats:

```
290 pages, 10.8 sec, 292M lua, 99M tex, 158 instances
290 pages,  9.5 sec, 149M lua, 35M tex,  30 instances
```

So, we win on all fronts when we use this glyph scaling mechanism. The magic primitive that deals with this is named `\glyphscale`; it accepts a number, where `1200` and `1.2` both mean scaling to 20% more than normal. But it's best not to use this primitive directly.

A specific scaled font can be defined using the `\definefont` command. In LMTX a regular scaler can be followed by two scale factors. The next example demonstrates this (as can be seen, the `yoffset` affects the baseline):

```
\definefont[FooA][Serif*default @ 12pt 1800 500]
\definefont[FooB][Serif*default @ 12pt 0.85 0.4]
\definefont[FooC][Serif*default @ 12pt]

\definetweakedfont[runwider] [xscale=1.5]
\definetweakedfont[runtaller][yscale=2.5,
                              xscale=.8,yoffset=-.2ex]

\def\testtext{test test \runwider test test
                        \runtaller test test}
{\FooA \testtext}\par
{\FooB \testtext}\par
{\FooC \testtext}\par
```

test test test test test test
test test test test test test
test test test test test test

We also use the new `\definetweakedfont` command here. This example not only shows the two scales but also introduces the offset.

In compact mode this is one font. Here is another example:

```
\definetweakedfont[squeezed][xscale=0.9]
\startlines
$a = b^2 + \sqrt{c}$
{\squeezed $a = b^2 + \sqrt{c}$}
\stoplines
```

$$a = b^2 + \sqrt{c}$$
$$a = b^2 + \sqrt{c}$$

Watch this:

```
\startcombination[3*1]
 {\bTABLE
    \bTR \bTD foo \eTD \bTD[style=\squeezed] $x = 1$
    \eTD \eTR
    \bTR \bTD oof \eTD \bTD[style=\squeezed] $x = 2$
    \eTD \eTR
  \eTABLE}
 {local}
 {\bTABLE[style=\squeezed]
    \bTR \bTD $x = 1$ \eTD \bTD  $x = 3$ \eTD \eTR
    \bTR \bTD $x = 2$ \eTD \bTD  $x = 4$ \eTD \eTR
  \eTABLE}
 {global}
 {\bTABLE[style=\squeezed\squeezed]
    \bTR \bTD $x = 1$ \eTD \bTD  $x = 3$ \eTD \eTR
    \bTR \bTD $x = 2$ \eTD \bTD  $x = 4$ \eTD \eTR
  \eTABLE}
 {multiple}
\stopcombination
```

foo	$x = 1$
oof	$x = 2$

local

$x = 1$	$x = 3$
$x = 2$	$x = 4$

global

$x = 1$	$x = 3$
$x = 2$	$x = 4$

multiple

An additional style parameter is also honored:

```
\definetweakedfont[MyLargerFontA]
                  [scale=2000,style=bold]
test {\MyLargerFontA test} test
```

This gives:

test **test** test

Just for the record: the Latin Modern fonts, when set up to use design sizes, will still use the specific size-related files.

6 Hackery

You can use negative scale values, as is demonstrated in the following code:

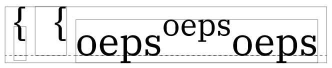
```
\bTABLE[align=middle]
 \bTR
  \bTD a{\glyphxscale 1000 \glyphyscale 1000 bc}d\eTD
  \bTD a{\glyphxscale 1000 \glyphyscale-1000 bc}d\eTD
  \bTD a{\glyphxscale-1000 \glyphyscale-1000 bc}d\eTD
  \bTD a{\glyphxscale-1000 \glyphyscale 1000 bc}d\eTD
 \eTR
 \bTR
  \bTD \tttf +1000 +1000 \eTD
  \bTD \tttf +1000 -1000 \eTD
  \bTD \tttf -1000 -1000 \eTD
  \bTD \tttf -1000 +1000 \eTD
 \eTR
\eTABLE
```

gives:

abcd	apcd	da	dd
+1000 +1000	+1000 -1000	-1000 -1000	-1000 +1000

Glyphs can have offsets and these are used for implementing OpenType features. However, they are also available on the TEX side. Take this example where we use the new `\glyph` primitive (a variant of `\char` that takes keywords):

```
\ruledhbox{
 % left curly brace:
 \ruledhbox{\glyph yoffset 1ex options 0 123}
 \ruledhbox{\glyph xoffset .5em yoffset 1ex
                  options "C0 123}
 \ruledhbox{oeps%
   {\glyphyoffset 1ex \glyphxscale 800
    \glyphyscale\glyphxscale oeps}oeps}
}
```



This example demonstrates that the `\glyph` primitive takes quite a few keywords: `xoffset`, `yoffset`, `xscale`, `yscale`, `left`, `right`, `raise`, `options`, `font` and `id` where the last two take a font identifier or font id (a positive number). For this article it's enough to know that the option indicates that glyph dimension should include the offset. In a moment we will see an alternative that doesn't need that.

```
\samplefile{jojomayer}
{\glyphyoffset .8ex
 \glyphxscale 700 \glyphyscale\glyphxscale
 \samplefile{jojomayer}}
{\glyphyscale\numexpr3*\glyphxscale/2\relax
 \samplefile{jojomayer}}
{\glyphyoffset -.2ex
 \glyphxscale 500 \glyphyscale\glyphxscale
 \samplefile{jojomayer}}
\samplefile{jojomayer}
```

To quote Jojo Mayer:

If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become. If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become. If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become. If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become. If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become.

Keep in mind that this can interfere badly with font feature processing which also used offsets. It might often work out okay vertically, but less well horizontally.

The scales, as mentioned, works with pseudo-scales but that is sometimes a bit cumbersome. This is why a special `\numericscale` primitive has been introduced.

```
1200 : \the\numericscale1200
1.20 : \the\numericscale1.200
```

Both these lines produce the same integer:

```
1200 :  1200
1.20 :  1200
```

You can do strange things with these primitives but keep in mind that you can also waste the defaults.

```
\def\UnKernedTeX
  {T%
   {\glyph xoffset -.2ex yoffset -.4ex `E}%
   {\glyph xoffset -.4ex options "60 `X}}
We use \UnKernedTeX\ and {\bf \UnKernedTeX} and
{\bs \UnKernedTeX}: the slanted version could
use some more left shifting of the E.
```

This gives the TEX logos but of course we normally use the more official definitions instead.

We use TEX and **TEX** and ***TEX***: the slanted version could use some more left shifting of the E.

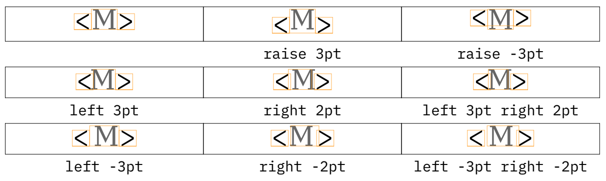
Because offsets are (also) used for handling font features like mark and cursive placement as well as special inter-character positioning, the above is suboptimal. Here is a better alternative:

```
\def\UnKernedTeX
  {T\glyph left .2ex raise -.4ex `E%
    \glyph left .4ex `X\relax}
```

The result is the same:

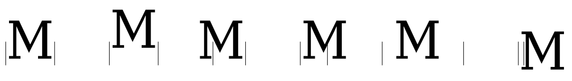
We use TEX and **TEX** and ***TEX***: the slanted version could use some more left shifting of the E.

But anyway: don't overdo it. We have dealt with such cases for decades without these fancy new features. The next example shows margins in action:



Here is another way of looking at it:

```
\glyphscale 4000
\vl\glyph                           `M\vl\quad
\vl\glyph raise  .2em               `M\vl\quad
\vl\glyph left   .3em               `M\vl\quad
\vl\glyph              right  .2em`M\vl\quad
\vl\glyph left  -.2em right -.2em`M\vl\quad
\vl\glyph raise -.2em right  .4em`M\vl
```



The raise as well as left and right margins are taken into account when calculating the dimensions of a glyph.

7 Implementation

Discussing the implementation in the engine makes no sense here, also because details might change. However, it is good to know that many properties travel with the glyph nodes, for instance the scales, margins, offsets, language, script and state properties, control over kerning, ligaturing, expansion and protrusion, etc. The dimensions (width, height and depth) are not stored in the glyph node but calculated from the font, scales and optionally the offsets and expansion factor. One problem is that the more clever (and nice) solutions we cook up, the more it might impact performance. So, I will delay some experiments till I have a more powerful machine.

One reason for *not* storing the dimensions in a glyph node is that we often copy those nodes or change character fields in the font handler and we definitely don't want the wrong dimensions there. At that moment, offsets and margin fields don't reflect features yet, so copying them is no big deal because at that moment these are still zero. However, dimensions are rather character bound so every time a character is set, we also would have to set the dimensions. Even worse, when we can set them, the question arises if they were already set explicitly. So, this is a can of worms we're not going to open: the basic width, height and depth of the glyph as specified in the font is used and combined with actual dimensions (likely already scaled according the glyph scales) in offset and margin fields.

Now, I have to admit that especially playing with using margins to glyphs instead of font kerns is more of an experiment to see what the consequences are than a necessity, but what would be the joy of TEX without such experiments? And as usual, in ConTEXt these will become options in the font handler that one can enable, or not.

◇ Hans Hagen
`http://pragma-ade.com`

Some fonts with recent TEX support

Karl Berry

(LA)TEX support for many new typeface families has been created in recent years. Here is an extremely terse visual overview of most of those appearing in the past year or so. All the fonts are shown here at their own nominal size of 10pt.

All the fonts shown here are available in Type 1 format. Almost all are also available in OpenType or TrueType, but fonts available only in OpenType/TrueType are omitted, regrettably.

Each of these families has its own set of additional variants (bold, bold italic, small caps, different encodings, etc.). For more complete showings, exact package invocations, the myriad other fonts available, etc., please see the urls in the signature.

Serif

Clara: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Domitian: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

ETbb: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

IbarraRealNova: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

MLModern: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Spectral: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

TeXGyreScholaX: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Sans serif

Archiv0: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Arvo: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Atkinson: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Gudea: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

HindMadurai: ABC FGHMQ abc fghlmq 012

Inter: ABC FGHMQ abc fghlmq 012

Josefin: ABC FGHMQ abc fghlmq 012

Magra: ABC FGHMQ abc fghlmq 012

Nunito: ABC FGHMQ abc fghlmq 012
ABC FGHMQ abc fghlmq 012

Oswald: ABC FGHMQ abc fghlmq 012

Slab serif and typewriter

AlfaSlabOne: ABC FGHMQ abc fghlmq 012

`Cascadia Code: ABC FGHMQ abc fghlmq 012`

`Courier10Pitch: ABC FGHMQ abc fghlmq 012`
`ABC FGHMQ abc fghlmq 012`

◇ Karl Berry
`https://tug.org/FontCatalogue`
`https://ctan.org/topic/font`

doi.org/10.47397/tb/42-1/tb130berry-fontlist

Generalized mediation operation in METAFONT

Hu Yajie

Abstract

The macros on page 299 of *The METAFONTbook*, which generalize METAFONT's mediation operation, have some bugs which went unnoticed for years. This article discusses how to fix the bugs, and some other improvements to the macros.

1 The problem

METAFONT's mediation operation allows us to write

- `1/3[z1,z2]` for the point one-third of the way from z_1 to z_2,
- `1/2[z1,z2]` for the point midway between z_1 and z_2,
- `.8[z1,z2]` for the point eight-tenths of the way from z_1 to z_2,

and, in general, `t[z1,z2]` stands for the point that lies a fraction t of the way from z_1 to z_2.

Our goal is to extend METAFONT's syntax so that it will accept generalized mediation formulas like `1/2[z1,z2,z3]` and `.4[z1,z2,z3,z4]`, computed as in the construction of Bézier curves (see Figure 1).

2 The original macros

Page 299 of *The METAFONTbook* gives some macros that implement the generalized mediation operation. The basic idea is to make `[` a macro that counts how many comma-separated expressions follow, up to the matching `]`. If there are fewer than three, as in any of

```
path p[][]a
x[n]
1/3[z1,z2]
```

we don't need to do anything special, so we restore the expressions in primitive brackets. Otherwise we store away the expressions and make

`[a,b,c]` expand to `Bernstein 3`,
`[a,b,c,d]` expand to `Bernstein 4`,
...

The binary-operator-like `Bernstein` macro then absorbs the fraction to the left and computes the result $t[u_1,\ldots,u_n] = \sum_{k=1}^{n} \binom{n-1}{k-1}(1-t)^{n-k}t^{k-1}u_k$.

However, the *METAFONTbook* macros have two bugs which can cause innocent commands like

```
draw flex((0,0),(100,100),(300,0));
```

to stop working. The first bug is easy to fix: rename the private variable $n\_$ to $bn\_$ to avoid a name conflict with the $n\_$ in plain METAFONT's `flex` routine. The second bug is harder to find: the definition of `flex`

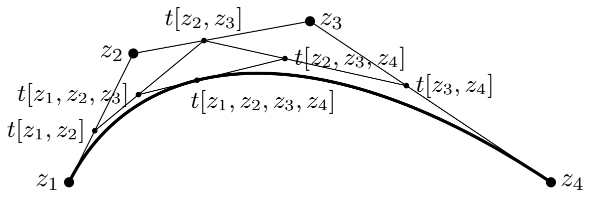


Figure 1: The generalized mediation operation

says `z_[incr n_]`, which usually increases $n\_$ once. But the `[` macro evaluates the expressions up to the matching `]` twice, once to count their number and once in primitive brackets, so $n\_$ gets increased twice and you get index mismatch errors. This bug can be fixed by changing `if bn_<3: [[[t]]]` on line 6 to

```
    if bn_=0: [[[]]]
elseif bn_=1: [[[u_[[[1]]] ]]]
elseif bn_=2: [[[u_[[[1]]], u_[[[2]]] ]]]
```

to reuse the result of the first-time evaluation.

3 The improved macros

While fixing the bugs, I also discovered some other improvements to the *METAFONTbook* macros:

```
let [[[ = [; let ]]] = ];
def [ = for u = enddef;
def ] = , hide(bn_ := 0; let v_ = \; ):
 if incr bn_ = 1: hide(def v_ = u enddef)
 else: hide(expandafter def expandafter v_
  expandafter = v_, u enddef) fi endfor
 if bn_ < 3: [[[v_]]]
 else: Bernstein bn_ fi enddef;
primarydef t Bernstein nn = begingroup
 c_[[[1]]] := 1; for n = 1 upto nn - 1:
  c_[[[n + 1]]] := t * c_[[[n]]];
  for k = n downto 2: c_[[[k]]] :=
   t[[[c_[[[k]]], c_[[[k - 1]]] ]]]; endfor
  c_[[[1]]] := (1 - t) * c_[[[1]]]; endfor
 bn_ := 0; for u = v_: + c_[[[incr bn_]]] * u
 endfor endgroup enddef;
```

The first improvement is that `[` and `]` are changed to macros which expand separately; this allows the `]` to be buried in another macro like `]]`, a single token which plain METAFONT expands to `] ]`. The second improvement is that the expressions between `[` and `]` are now stored in a "list macro" instead of an array. This makes the code simpler, readily adaptable to new types like METAPOST colors, and diagnostics with `show` and `showdependencies` more readable:

```
*show 2[a,b,c];
>> 4c-4b+a   (formerly u_1 or %CAPSULE4691)
```

I gratefully thank Donald Knuth for suggesting that I write this note.

◇ Hu Yajie
https://github.com/dine2014

Animating Fourier series decomposition of a character with LuaTEX and MPLIB

Maxime Chupin

Abstract

In this article, we will see how, thanks to METAPOST and MPLIB and LuaTEX, we can build an animation illustrating in a mechanical way the Fourier decomposition of a closed contour.

This is a translation by the author of the original article in French, published in La Lettre GUTenberg *number 41 [2] of the French TEX user group.*

1 Introduction

The video artist 3Blue1Brown,[1] mathematical popularizer on YouTube, has made a video illustrating the Fourier decomposition of a closed path by animations of gear mechanisms of circles put end to end. The result is magnificent and bewitching (see figure 1).

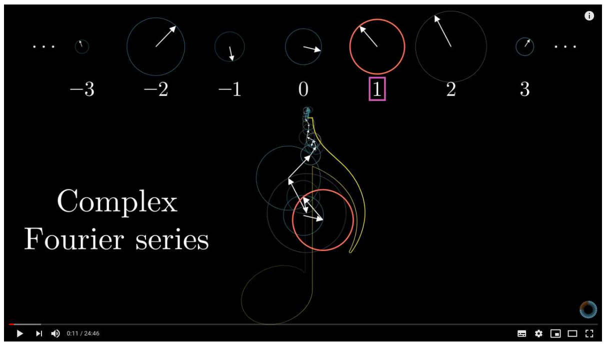


Figure 1: *But what is a Fourier series? From thermal transfer to drawings with circles* from 3Blue1Brown on YouTube.

It is not easy to describe these animations in words, but the idea is to put circles of different diameters with inscribed vectors that rotate at different speeds end to end, and the end of this broken line traces the closed curve (the music note in figure 1).

This viewing made me want to do this with our favorite tools, specifically with LuaLATEX and METAPOST, itself included in LuaTEX via the MPLIB library. I also thought it would be interesting to make these animations with the outline of a glyph of a character (of one part, i.e., connected). So I started working on this project.

2 Mathematical principle

We therefore consider a closed curve in $\mathbf{R}^2$ that can be considered as a periodic function $f : \mathbf{R} \mapsto \mathbf{C}$. The period is considered to be equal to 1. Without going into details, the Fourier series decomposition of f is:

$$\forall t \in [0,1], \quad f(t) = \sum_{n=-\infty}^{+\infty} c_n(f)\mathrm{e}^{\mathrm{i}n2\pi t},$$

where

$$c_n(f) = \int_{-1/2}^{1/2} f(t)\mathrm{e}^{-\mathrm{i}n2\pi t}\mathrm{d}t.$$

Numerically, we will work with discrete versions of this decomposition in Fourier series. Consider two integers N and M large enough and M even. In the discrete world, we will no longer have the continuous f function but samples along the path, which we will denote by $(f_1, f_2, \ldots, f_N)$ where the $f_i \in \mathbf{C}$. We then have:

$$\forall t \in [0,1], \quad f(t) \simeq \sum_{n=-M/2}^{M/2} \tilde{c}_n(f)\mathrm{e}^{\mathrm{i}n2\pi t}, \qquad (1)$$

where

$$\tilde{c}_n(f) = \sum_{k=0}^{N} \frac{1}{N} f_{k+1}\mathrm{e}^{-\mathrm{i}\frac{2nk}{N}}. \qquad (2)$$

The $M+1$ $\tilde{c}_n(f)$ will be called the Fourier coefficients.

Geometrically, we can see the relation (1) as a sum of vectors of $\mathbf{R}^2$ (thus put end to end) with norm the *modulus* of the complex number and, as orientation, its *argument*.

Thus, when t runs through the interval $[0,1]$, these vectors rotate and the end of the last vector draws the closed curve[2] that we have decomposed.

3 Get a set of points of $\mathbf{R}^2$ of the contour

We must therefore construct, from the contour of a glyph, the sequence $(f_1, f_2, \ldots, f_N)$ presented above.

3.1 Thanks to METAPOST

First, we need to obtain a discretization of the closed contour of a given glyph. METAPOST [5], with the MetaFun [3] format, allows us to do this quite easily. For the example, we will take the glyph f, 500 points for the discretization, and a certain homothetic factor set to 0.1 for the display.

The METAPOST code is the following:

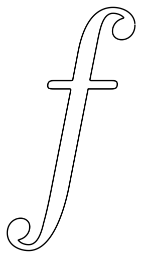
```
fontmapfile "=lm-ec.map";
picture lettre; path contourLettre; path p;
lettre := glyph "f" of "ec-lmri10";
nbrPoints := 500; scale := 0.1;
beginfig(1);
for item within lettre:
 contourLettre := pathpart item;
 for i:=1 upto nbrPoints:
  if i=1:
   p := point i/nbrPoints along contourLettre;
```

[1] youtube.com/watch?v=-qgreAUpPwM

[2] Or rather an approximation of the contour of the glyph.

```
  else:
   p := p--(point i/nbrPoints
            along contourLettre);
  fi;
 endfor;
 draw p scaled scale;
endfor; endfig; end;
```

The result is shown below, using the `luamplib` package [4] to use METAPOST directly in this article.



We will not detail this code here. It seems a bit complex but this is due to the METAPOST's glyph structure that (fortunately) allows having several parts for a glyph. Although here we will consider only letters with one (connected) part, we must adhere to the general data structure. The thing to remember is that we have METAPOST code which allows us to obtain a set of points constituting a discretization of the character glyph outline.

3.2 The list of points with Lua

The computation of the Fourier series decomposition of this closed curve is theoretically possible in TEX, but Lua [6] offers us more capabilities, more speed, and easier coding.[3] So, using LuaTEX, we want to retrieve this list of points on the Lua side. Another advantage is that, as we have already said, LuaTEX includes METAPOST via the Lua library, MPLIB (see [7]).

3.3 Search for font files

With METAPOST (or MPLIB), unfortunately we can use only Type 1 fonts. There is a little subtlety concerning the opening of the font file: METAPOST asks for a '`pfb`' file type, while `kpse` asks for a '`type1 fonts`' file type. Taco Hoekwater, on the `metapost@tug.org` mailing list, provided me with the search function that handles this little problem:

```
local mpkpse = kpse.new('luatex', 'mpost')
local function finder(name, mode, ftype)
if mode == "w" then
 return name
else
 if ftype == 'pfb' then
  ftype='type1 fonts'
 end
 return mpkpse:find_file(name,ftype)
end; end
```

[3] At least, for me ...

3.4 From METAPOST to a Lua table

Having passed this small technical difficulty, we will present here a Lua function that allows us to build a Lua table which contains the points generated by METAPOST.

```
function getpathfrommp(s,nbrPoints,scale)
-- define a Lua function which retrieves the list of
-- nbrPoints points made from the outline of the
-- character s; scale is a homothety parameter

-- launch MetaPost session using our search function
local mp = mplib.new({find_file = finder,})
-- metafun Format
mp:execute('input metafun ;')
-- we store the output of the MetaPost code execution
local rettable; rettable = mp:execute(
'fontmapfile "=lm-ec.map";
picture lettre; path contourLettre;
lettre := glyph "' .. s .. '" of "ec-lmri10";
path p; beginfig(1);
for item within lettre:
 contourLettre := pathpart item;
 for i:=1 upto'.. nbrPoints ..':
  if i=1: p := point i/'..nbrPoints..' along
    contourLettre;
  else: p:= p--(point i/'..nbrPoints..' along
    contourLettre);fi;
endfor;
draw p scaled '..scale..';
endfor; endfig; end;') -- MetaPost code as above
output = {} -- initialization
-- if the MetaPost code execution went well
if rettable.status == 0 then
 figures = rettable.fig -- figure list
 figure = figures[1] -- first and only figure
 local objects = figure:objects() -- object list
 -- compose figure from the first and only object:
 local segment = objects[1]
 for point =1, #segment.path do
  output[point] = {}
  output[point].x = segment.path[point].x_coord
  output[point].y = segment.path[point].y_coord
 end
end
else print("error") end;
return output
end
```

To roughly explain the above code, the purpose is to get the output of the execution of a code by METAPOST (MPLIB here). This output has a Lua structure (see the LuaTEX documentation [7], section `mplib`). So we browse this structure to extract the list of points we are interested in: first of all the list of figures, which here is limited to a single one, then inside the first figure, we look for the `objects` which again are limited to a single `object` (our closed curve), then we browse the `path` (METAPOST) of the object, named here `segment`, and finally we retrieve the x and y coordinates that we store in our output variable `output`. For a description of the Lua functions allowing us to browse the structure of the object produced by the execution of the METAPOST code, please refer to the LuaTEX documentation.

We pass in three parameters: the character whose outline we want to trace (`s`), the number of points (`nbrPoints`), and the homothety (`scale`). Our code is not robust, because if the glyph corresponding to the character `s` is not connected, there is a strong chance that the code will not work.

4 Fourier series decomposition with Lua

4.1 Call to an external library

The Fourier series decomposition is done with complex numbers as presented previously. Complex numbers are not natively managed by Lua, but many libraries are available on the Web that implement computation functions on complex numbers. I chose the `complex.lua` file available at `lua-users.org/wiki/ComplexNumbers`.

To use this library, we need:

1. on the LATEX side, to load the `luapackageloader` package (see the end of this article for the complete LATEX code);
2. on the Lua side, to call the file `complex.lua` via the following code:

```
complex = require "complex"
```

4.2 Convert the list of coordinates of $\mathbf{R}^2$ into a list of complex numbers

To ease the computations, a function is created to convert the list of coordinates obtained by the Lua function `getpathfrommp` into a list of complex numbers. This is done by the following code, which needs no further explanation.

```
function pathToComplex(path)
local complexPath
complexPath = {}
for i=1,#path do
 complexPath[i]
   = complex.new(path[i].x,path[i].y)
end
return complexPath
end
```

4.3 Implementation of Fourier coefficients calculation

The computation of the Fourier coefficients $\tilde{c}_n(f)$ of equality (2) is easily implemented with Lua, as shown in the following code.

```
function cn(f,n)
local CN = complex.new(0.0,0.0)
local N = #f
for i=0,N-1 do
 exposant = complex.new(0.0,-2.0*math.pi*n*i/N)
 Exp = complex.exp(exposant)
 CN = complex.add(CN,complex.mulnum(complex.mul
    (f[i+1],Exp),1.0/N))
end
return CN; end
```

From this, we need to build the list

$$(c_{-M/2}, c_{-M/2+1}, \ldots, c_{-1}, c_0, c_1, \ldots, c_{M/2}),$$

which is done by the following Lua function:

```
function cnList(f)
local CNlist = {}
local M = #f
for i=0,M do
 CNlist[i] = cn(f,math.floor(i-M/2))
end
return CNlist; end
```

5 Plot with mplibcode

Once all these code bricks are prepared, we just have to implement the drawing with the help of the `mplibcode` environment of the `luamplib` package [4] (or we could use TikZ). This function has several arguments:

- a discretized `path`, i.e., the set of coordinates (x, y) of the contour of the glyph;
- its conversion into complexes (`complexPath`);
- a list of Fourier coefficients (`cnList`);
- a desired number of Fourier coefficients ($M + 1$ in the previous equations), i.e., the number of circles and vectors drawn (`nbrFourier`);
- a time $t \in [0, 1]$.

```
function coreDecomp(path, complexPath, cnList,
    nbrFourier, t)
-- path: list of R^2 points
-- complexPath: complex list of these points
-- cnList: list of Fourier coefficients
-- nbrFourier: number of Fourier coefficients
-- initialization
local str
local cnListRotated = {}
local zero = math.floor(#cnList/2)
local NFourier = math.floor(nbrFourier/2)
```

```
cnListRotated[zero] = cnList[zero]
-- multiplication by e^{2i k pi t}
for k=1,zero do
 cnListRotated[zero+k] = complex.mul(
    cnList[zero+k],complex.exp(complex.new(0.0,
                               k*2*math.pi*t)))
 cnListRotated[zero-k] = complex.mul(
    cnList[zero-k],complex.exp(complex.new(0.0,
                              -k*2*math.pi*t)))
end
-- beginning of mplibcode
local str = "\\begin{mplibcode}\nverbatimtex
  \\leavevmode etex; beginfig(1);"
-- MetaPost code of the glyph to draw
local mpCodeLetter = mpCodePath(path)
str = str..mpCodeLetter -- concatenation
-- complex current point at which
-- we draw the next circle
local currentC = complex.new(0,0)
-- add the drawing of the circle and the vector
str = str .. mpCodeCircle(cnListRotated[zero],
                          currentC)
currentC = complex.add(currentC,
                       cnListRotated[zero])
for i=1,NFourier do -- for all Fourier coeff
 str = str..mpCodeCircle(cnListRotated[zero+i],
                          currentC)
 currentC = complex.add(currentC,
                       cnListRotated[zero+i])
 str = str..mpCodeCircle(cnListRotated[zero-i],
                          currentC)
 currentC = complex.add(currentC,
                       cnListRotated[zero-i])
end
str = str.."endfig;\n\\end{mplibcode}\n"
      .."\\newpage" -- closing
return str; end
```

To help in reading this code: `cnListRotated[i]` corresponds to the terms in the sum (1) of $\tilde{c}_n(f)e^{2ik\pi t}$, since the multiplication by $e^{2ik\pi t}$ can be seen as a rotation in the complex plane.

The main purpose of this function is to construct a string containing the `mplibcode` that will be sent to LaTeX via the Lua `tex.sprint()` function. The `coreDecomp` function above calls two other Lua functions that produce the METAPOST code of the drawing:

- the function `mpCodePath(path)`, which takes as argument the list of the contour points of the glyph and draws the glyph;[4]
- the function `mpCodeCircle(cn,shift)`, which takes as argument a coefficient of Fourier `cn` (after rotation) and an $\mathbf{R}^2$ shift which is the end of the broken line where the vector and the circle must be drawn.

[4] There is also a frame drawn around it to make sure that all images have the same size and thus be able to chain the images to produce an animation.

The code for these two functions is below. They mainly consist of the concatenation of strings to produce METAPOST code.

```
function mpCodePath(path)
-- plot the path contour
local str = ""
str = str.."path p; p:="
for i=1,#path do
 str = str.."("..string.format("%f",path[i].x)
      ..","..lstring.format("%f",path[i].y)
      ..")--"
end
str = str.."cycle; draw p;\n"
str = str.."pair ll,lr,ur,ul; ll:=llcorner p;"
      .."ur:=urcorner p; lr:=lrcorner p;"
      .."ul:=ulcorner p;\n"
str = str.."Wdth := abs(xpart lr - xpart ll);"
      .."Hght := abs(ypart ul- ypart ll);"
      .."prcW := 0.8; prcH := 0.3;\n"
str = str.."draw (ll+(-prcW*Wdth,-prcH*Hght))"
      .."--(lr+(+prcW*Wdth,-prcH*Hght))"
      .."--(ur+(+prcW*Wdth,+prcH*Hght))"
      .."--(ul+(-prcW*Wdth,+prcH*Hght))"
      .."--cycle;\n"
return str; end

function mpCodeCircle(cn,shift)
-- draw the circle and the vector corresponding to
-- the Fourier coefficient centered at points shift
local str
local abs,arg
abs,arg = complex.polar(cn)
str = "draw fullcircle scaled "
      ..string.format("%f",2*abs).."shifted ("
      ..string.format("%f",shift[1])..","
      ..string.format("% f",shift[2])
      ..") withcolor (0.7,0.7,0.7);\n"
str = str.."drawarrow ((0,0)--("
      ..string.format("%f",cn[1])..","
      ..string.format("%f",cn[2]).."))shifted("
      ..string.format("% f",shift[1])..","
      ..string.format("%f",shift[2])
      ..") withpen pencircle scaled 1pt "
      .."withcolor (0.7,0.3,0.3);"
return str; end
```

5.1 Generate images for any $t \in [0,1]$

To create the animation, we generate the images with a discretization of the time interval $[0,1]$. This can be done with the following function.

```
function plotDecompAnim(letter,nbrPoints,
    nbrFourier,scale,nbrFrame)
-- letter: character that we want to decompose
-- nbrPoints: number of points in discretization
-- nbrFourier: number of Fourier coefficients
-- scale: homothetic coefficient
-- nbrFrame: frame number
local str
local path = getpathfrommp(letter,nbrPoints,
                           scale)
local complexPath = pathToComplex(path)
local cnList = cnList(complexPath)
for frame=0,nbrFrame-1 do
 t = frame/nbrFrame
 str = coreDecomp(path,complexPath,cnList,
    nbrFourier,t)
 tex.sprint(str)
end
end
```

The Lua functions presented are all put in a single `Fourier.lua` file.

6 Animations and code

Once all these Lua functions are implemented, we just have to load them and call the Lua function `plotDecompAnim` using the command `\directlua`, as shown in the following code.

```
\documentclass{article}
\usepackage{luapackageloader}
\usepackage{luamplib}
\directlua{dofile("Fourier.lua")}
\pagestyle{empty}
\begin{document}
\directlua{
  plotDecompAnim("f",300,50,0.26,360)
}
\end{document}
```

To conclude, we have considered only three files: the LaTeX file above `fourier.tex`, the `Fourier.lua` file which contains all our Lua functions presented here, and the `complex.lua` file retrieved from the web. To compile and produce the PDF, which contains as many pages as there are images, we put these three files in the same directory and run `lualatex` on our `fourier.tex` file:

```
$ lualatex fourier.tex
```

If you read this article as a PDF file with Acrobat Reader, you will be able to see the generated animation (cf. figure 2) with the `animate` package [1]. Otherwise, all the code and the animation are visible and downloadable here:

`fougeriens.org/~mc/?page=exemples&dir=fourier`

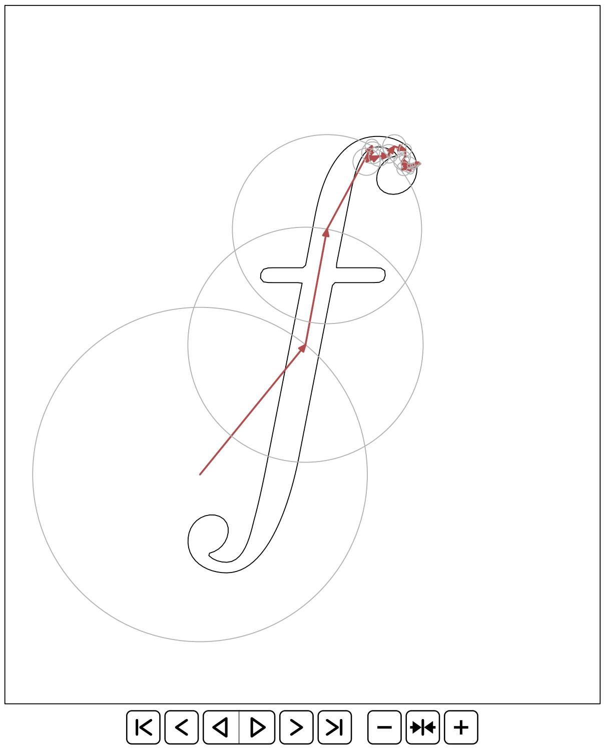


Figure 2: Animation

References

[1] A. Grahn. The `animate` package, 2020. `https://ctan.org/pkg/animate`.

[2] Association GUTenberg. *La Lettre GUTenberg* numéro 41, Décembre 2020. `https://www.gutenberg.eu.org/IMG/pdf/lettre41.pdf`.

[3] H. Hagen. *Metafun*. `http://www.pragma-ade.com/general/manuals/metafun-p.pdf`, 2020. v. 2.11.3.

[4] H. Hagen, T. Hoekwater, et al. The `luamplib` package. `https://ctan.org/pkg/luamplib`, 2020. v. 2.11.3.

[5] J.D. Hobby, MetaPost Development Team. *MetaPost, a user's manual*. `https://ctan.org/pkg/metapost`, 2019. v. 2.0.

[6] R. Ierusalimschy. *Programming in Lua*. `Lua.org`, 2016.

[7] LuaTeX development team. *LuaTeX Reference Manual*. `http://www.luatex.org/svn/trunk/manual/luatex.pdf`, March 2020. v. 1.12.

◇ Maxime Chupin
29 rue Pierre et Marie Curie
91400 Orsay
France
`mc (at) melusine dot eu dot org`
`https://fougeriens.org/~mc/`

Working remotely from an island: arara and other tools

Island of TEX (developers)

Abstract

Over the last two years, the Island of TEX has complemented the TEX ecosystem with some auxiliary tools. This article is a short review of the last year's (more or less) achievements and a preview of upcoming changes.

1 Providing a home for TEX-related projects

In 2019, the Island of TEX started as a small group of two friends trying to improve the TEX ecosystem (cf. fig. 1). What started as small steps towards a new arara release with some little side-projects became a more and more interesting place to stop by in 2020.

Figure 1: The Island of TEX logo.

The last year started off in full preparation of releasing arara version 5 in due time for the TEX Live 2020 pretest. A new arara version, new problems. Software engineering becomes "fun" as soon as users are concerned, yet we are determined to help our users. More about where this view got us and arara in a later section.

Apart from our flagship project, the island focused on new frontiers over the whole last year. Therefore, we

- stabilized TEXplate,
- created an archive for stale TEX related projects (containing only a backup of the ε_χTEX repository for now) to preserve history,
- migrated checkcites, a tool to check for missing or unused references, to the island,
- published the new albatross tool, and
- worked on publicity.

Of our efforts, three projects received a fair share of attention: our Docker images, the new TEXdoc online tool and the shiny little albatross. Let us drop some words about each of these projects before looking at arara and the future.

2 New tools for a modern TEX ecosystem

In *TUGboat* 40:3,[1] we introduced the island's Docker images for easily reproducible builds as well as a semi-official response to the need for continuous integration (CI). Our images were among the first using vanilla TEX Live, providing the required tools for running software included in TEX Live (Java Virtual Machine (JVM), Python, etc.). Additionally, we provide TEX Live releases from 2014 on and let the user decide whether they want to pull all the documentation and source files into their CI configuration.

With attention came the idea to more officially publish our images as `texlive/texlive`, which we gladly did. Now we are managing the Docker Hub releases at `https://hub.docker.com/r/texlive/texlive`. Although unnoticed at first, we even saw DANTE e.V., the German-speaking TEX user group, basing their Docker images on ours.

Apart from applications in CI, the Docker images have lured us into creating another tool based on them. TEXdoc online, which we introduced in *TUGboat*'s previous issue,[2] is now the software running `https://texdoc.org`, the successor to `https://texdoc.net` (which now redirects), thanks to Stefan Kottwitz. We have incorporated a few improvements, among others HTTPS support. You can easily host your own instance.

After finishing TEXdoc online to a production-ready degree, the island turned to the development of another handy tool, albatross. This command line tool, with a silly yet adorable name, solves a very common problem: finding (system) fonts that provide a certain glyph. Users may provide the glyphs themselves — e.g, ß — or their corresponding Unicode code points in hexadecimal notation — e.g, `0xDF`. Currently, it is a thin wrapper around `fc-list` but there are plans to make it even more useful. As it is in TEX Live, you should give it a shot.

3 arara — feeling at home on the island

In 2019, arara, the cool TEX automation tool, was one of the first new citizens of the Island of TEX. It moved just in time to work on a new version, version 5. This version was special in many ways, first and foremost as it followed its predecessor after such a short period of time.[3] Behind the scenes, we finished a major rewrite of arara, mainly working on features from user feedback, especially directory support and the processing of multiple files.

[1] `tug.org/TUGboat/tb40-3/tb126island-docker.pdf`
[2] `tug.org/TUGboat/tb41-3/tb129island-texdoc.pdf`
[3] Some members of the TEX community at StackExchange might remember that working on arara version 4 was one of Paulo's major distractions while writing a never-ending thesis.

After releasing version 5 for the TeX Live 2020 pretest, we got hooked by the idea of aligning release schedules of arara with TeX Live releases. So we had big plans and started to work on version 6 right after releasing version 5. We might have been a bit too ambitious, though, as we received some complaints about a non-working version 5 from our users. Somehow, they caught our failure to test the new release on some versions of Windows. Well, everything has been fixed and we were able to move on.

Approximately three quarters of 2020 remained and we tried to make version 6 shine through an enhanced feature set and optimized workflows. Some new features we want to highlight:[4]

- Preambles (think of that as the commands arara will execute on that file) have received new options. Among others, you may now define your workflows using preambles and set a global default preamble. That way, you may now even call arara on files without those special arara comments (directives) and make our tool (auto)magically execute your default preamble. Users wanted to be able to switch their editor's default compiler to arara, even for files without explicit statements, and now they can.
- You may pass parameters from the outside into your build flow. Call arara as

  ```
  arara -P jobname=thesis file.tex
  ```

  and you can receive that parameter within your arara rules and directives like this (line breaks added for *TUGboat* formatting; this directive should be on one line):

  ```
  % arara: pdflatex: { options: [
    "-jobname=@{getSession()
                      .get("arg:jobname")}" ] }
  ```

  Multiple users have requested being able to parametrize their directives, so we finally managed to implement it.
- We added eight new rules and improved the rule format. The most frequent requests we receive are about supporting new tools in our rule set. We gladly add more TeX tools to arara, so if you are missing something feel free to contact us.
- For quite some time, arara has been a very powerful tool and has been criticized for being so powerful. We now implemented a first draft of a safe mode that restricts arara in some of the more harmful execution steps. Do not expect real safety, though. This is going to take more work to prevent obvious malicious behaviour.
- One of the most prominent problems with arara has been the lack of good introductory material, especially as the manual has grown. Hence, we now provide a quick start guide for new users. If you do not use arara yet, maybe this guide is for you. Interested? Simply run `texdoc arara-quickstart` on your local system.

The above is only a short excerpt of the change log but probably the most important changes for users. So to come back to the initial statement: version 6 is a major milestone in many ways but most importantly because the technical improvements of version 5 allowed us to implement so many features our users have been waiting for (sometimes for years).

4 Perspectives

The Island of TeX will continue to work on improving the TeX ecosystem in 2021. We hope to be as productive as we were in 2020. So let us try to outline our near-term goals.

First of all, we want to improve albatross and checkcites to make them even more useful. For the latter that will most probably include a rewrite. Anyway, these small helpers are what makes the daily TeX workflow a bit more fluent so we will try to gather new ideas for little new helpers. We have welcomed a new member to the island, who will reveal his first IoT tool soon.

However, the major plans and projects are settled around arara. In the long run, we want the tool to provide far more than just a CLI tool for compiling TeX. We have plans for a documented API for defining TeX flows that is not as `latexmk`-centred as some of the existing APIs floating around and some kind of arara daemon that will be able to translate multiple documents in parallel.

Furthermore, most of our tools are written in Kotlin and therefore rely on the JVM. With the so-called Kotlin multiplatform projects (MPP), we will try to get closer to native performance. Among others, this could also be useful in scenarios where a computer (or cloud service for that matter) with a TeX Live installation may not run a JVM.

To wrap this up, we have many plans and hope to realize as much as possible. If you are interested in helping us develop ideas or even implementing some code: visas for the island are free and easy to get, so feel free to reach out.

◇ Island of TeX (developers)
`https://gitlab.com/islandoftex`

[4] A detailed discussion of new features is in our blog post about arara's new release on its website at `https://islandoftex.gitlab.io/arara/`.

LuaMetaTEX programming features

Hans Hagen

1 Introduction

Sometimes you can read (or hear) comments about TEX not being a real programming language or the wish for it to be more like a typical procedural language. A discussion about this is somewhat pointless because it relates to experiences and preferences. Also, when we mention TEX, we are talking about an interpreter, a language, a set of macros and in practice, about an ecosystem, simply because all kinds of resources are involved — especially the ecosystem is one reason why a successor is not showing up.

So, when we discuss the language aspect, it concerns a macro language and that is for a good reason: one can mix content and operations on that content in one document source. That source is interpreted and processed as it goes. This is contrary to a procedural language, where one explicitly has to push content into some procedure. These are a bit of a mix, e.g., webpage templates where some elements are snippets of programs and a preprocessor assembles the result.

```
\def\MyMacroA#1{This or #1!}
\def\MyMacroB{that}
\MyMacroA{\MyMacroB}
```

Here the last line will result in “This or that!” ending up in the output. But it must be noted that `\MyMacroB` is passed as a token, and only in the body of the macro does it get expanded into “that”.

```
\edef\MyMacroC{\MyMacroB}
```

The code above defines a new macro with the expanded text as body. To expand or not, that is often the question. Now compare this code with the following:

```
function MyFunctionA(one)
    return "This or " .. one .. "!"
end
function MyFunctionB()
    return "that"
end
function MyFunctionC(one)
    return "This or " .. one() .. "!"
end

MyFunctionA("that")
MyFunctionA(MyFunctionB())
MyFunctionC(MyFunctionB)
```

The first function expects a string and returns a concatenation. The second function returns a string. The first call gets a string passed and the second one too because we call that function. But the third call passes the function itself, which is why the third function has to call it explicitly in the function body. It is this property that, in my opinion, complicates matters when you want to do typesetting in such a language: the more you nest the more dangers there are for asynchronous side effects. This can be understood from the following example:

```
function MyFunctionA(one)
    print("A")
    return "This or " .. one .. "!"
end
function MyFunctionB()
    print("B")
    return "that"
end

MyFunctionA(MyFunctionB())
```

Here we print `B` before we print `A`. Now, one can certainly argue that in spite of this, functions are easier to understand than macros (which can also have surprising side effects). Indeed, when one works on an abstract document tree where content is fetched from, say, a database that might be true but most TEX users mix content and operations.

In the following sections I will introduce some of the additional features that LuaMetaTEX provides. They are the result of experiencing many years of macro writing and the wish to come up with readable code using native features of the language when possible. Of course in ConTEXt we have a high level interface for dealing with typographical constructs and properties but deep down the code looks less clear. Putting layer upon layer doesn’t help much either, so we don’t go that route. Using funny characters like `!?@_:` doesn’t make things look better either. We do have lots of so-called low-level macros but it doesn’t make much sense to come up with a pseudo-programming layer while in fact the engine could make better facilities available; so that is the route we follow. After decades it had become clear that none of the successor TEX variants have filled in the gaps in this way, so at some point I decided that LuaMetaTEX should do it (at least for ConTEXt).

While ConTEXt MkII was written for the more traditional engines pdfTEX and XƎTEX, MkIV targets LuaTEX. It resulted in a rewrite of many components and a freeze of MkII. It made no sense to cripple ourselves so in the end we went further than originally expected. Then, when LuaMetaTEX development started, again a rewrite happened, but this time the reason was to make the code base a bit more efficient (less indirectness) by using extended native functionality. Apart from other benefits of this new engine, it gives a bit nicer code and the fewer layers we have the better. This is why ConTEXt LMTX (a.k.a. MkXL) again has a split-off code base so that MkIV is not harmed. All that said, I do admit that, lacking other TEX challenges, it is also fun to explore new venues.

2 Conditions

It must be said that when one goes even a little beyond simple TEX programming, one could indeed wish for a bit more comfort. Take this:

```
\def\MyMacro#1#2%
  {\ifdim#1<#2\relax
     less%
   \else\ifdim#1=#2\relax
     equal%
   \else
     more%
   \fi\fi}
```

One needs to keep track of the nesting here in order to have the right number of `\fi`'s.

```
\def\doifelse#1#2#3#4%
  {\edef\a{#1}\edef\b{#1}%
   \ifx\a\b#3\else#4\fi}
```

The temporary macros are needed in order to be able to compare the expanded meanings. But when `#3` and `#4` are macros that look ahead you can imagine that when they see `\else` or `\fi` things can get confused. Compare this to:

```
function doifelse(a,b,c,d)
    if a == b then
        c()
    else
        d()
    end
end
```

Here the compiler creates code that calls either `c` or `d` without them having to bother about leaving the condition. In TEX-speak we would need to have something like this:

```
\def\firstoftwoarguments #1#2{#1}
\def\secondoftwoarguments#1#2{#2}
\def\doifelse#1#2#3#4%
  {\edef\a{#1}\edef\b{#1}%
   \ifx\a\b
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

And when you try that with the first example where we had a nested condition you can imagine that it quickly starts looking complex. Another aspect of the last macro is that it uses two temporary macros that better have names that don't clash, so the ones we choose here are pretty bad. I will come back to dealing with that later.

One gets accustomed to this and often this kind of code is hidden from the user so only macro writers are victims here. But, being one myself, the question is, can we make the code look nicer? Let's redo the first example with LuaMetaTEX:

```
\def\MyMacro#1#2%
  {\ifdim#1<#2\relax
     less%
   \orelse\ifdim#1=#2\relax
     equal%
   \else
     more%
   \fi}
```

Many programming languages have something like `elseif` but because TEX has quite a number of different tests, `\elseifdim` makes no sense but the more generic `\orelse` does. We can even think of:

```
\def\MyMacro#1#2%
  {\ifcmpdim#1#2\relax
     less%
   \or
     equal%
   \else
     more%
   \fi}
```

And because LuaMetaTEX provides this test, one obstacle is gone. (Aside: if the `\relax` is not desired in the expansion, `\dimexpr` can be used:

```
\ifdim\dimexpr#1\relax=\dimexpr#2\relax
```

This is supported in all engines except the original TEX. There are yet more possibilities in LuaTEX and LMTX that we won't go into here, like `\beginlocalcontrol`.)

We leave it to the reader to come up with a traditional TEX implementation of this:

```
\def\MyMacro#1#2%
  {\ifcmpdim#1#2\relax
     \expandafter\firstofthreearguments
   \or
     \expandafter\secondofthreearguments
   \else
     \expandafter\thirdofthreearguments
   \fi}
```

And how nice it would be to be able to do this:

```
\def\doifelse#1#2%
  {\iftok{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

And so, LuaMetaTEX has such a primitive test. Keep in mind that defining `\iftok` as a macro is possible here but that won't work well nested, even with `\orelse`:

```
\iftok{.}{.}
\orelse\iftok{..}{..}
\orelse\iftok{...}{...}
\fi
```

When a condition succeeds or fails TEX enters fast scanning mode to skip over the branch that is not used. For that it needs to know if a token is a test, which is why defining `\iftok` as a macro is no help. We could flag a macro as a test and I actually played with this, but it means that we need to test a macro property independent of the current condition handler and that is something for later. As an intermediate solution we have an `\ifcondition` primitive that is seen as a condition when fast scanning happens and as a no-op when a condition is expected in which case the following macro has to expand to a condition itself. Something like this:

```
\ifcondition\mytest{.}{.}
\orelse\ifcondition\mytest{..}{..}
\orelse\ifcondition\mytest{...}{...}
\fi
```

Because we have Lua there are also ways to let Lua functions behave like if tests but that is beyond this overview, since it goes beyond the macro language. In ConTEXt we use this feature to implement some bitwise operations and tests.

In the engine we provide this repertoire of tests: `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifabsnum`, `\ifabsdim`, `\ifchknum`, `\ifchkdim`, `\ifcmpnum`, `\ifcmpdim`, `\ifnumval`, `\ifdimval`, `\iftok`, `\ifcstok`, `\ifcondition`, `\ifflags`, `\ifempty`, `\ifrelax`, `\ifboolean`, `\ifmathparameter`, `\ifmathstyle`, `\ifarguments`, `\ifparameters`, `\ifparameter`, `\ifhastok`, `\ifhastoks` and `\ifhasxtoks`.

Some of these are variants of `\ifcase`, needed when there are more than two outcomes possible. In addition there are `\unless`, `\else`, `\or`, `\orelse` and `\orunless`. The new primitives are discussed in documents that come with the ConTEXt distribution.

With respect to testing arguments, you can also use the pseudo-counter `\lastarguments` (watch the 'last' in the name) and somewhat less efficient but more reliable `\parametercount` instead as these are indicators of the number of passed commands.

3 Protection

In the previous section we mentioned that using auxiliary macros is tricky because they can clash with existing macros. In fact, this is true for any macro! I therefore decided to do what has been on the agenda for a while: add a mechanism that protects against overload. This is still experimental and the impact on users can only be tested after most ConTEXt users have switched to LMTX, which may take a while. This also means that it will take a while before the related primitives are considered stable (although I'm sure not much will change). Let's take a previous example:

```
\permanent\def\firstoftwoarguments #1#2{#1}
\permanent\def\secondoftwoarguments#1#2{#2}
\permanent\protected\def\doifelse#1#2%
  {\iftok{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

Here the three macros are defined as permanent. The test itself is protected against expansion (which it has always been so we keep that). Depending on the value of the `\overloadmode` variable (discussed below) a user will get a warning or fatal error. By default there is no checking (but I might give the `\immutable` prefix, also discussed below, an "always check for it" property).

The whole repertoire of prefixes related to overload protection is given in the following table.

`frozen`	a macro that has to be redefined in a managed way
`permanent`	a macro that had better not be redefined
`primitive`	a primitive that normally will not be adapted
`immutable`	a macro or quantity that cannot be changed, it is a constant
`mutable`	a macro that can be changed no matter how well protected it is
`instance`	a macro marked (for instance) to be generated by the user interface
`overloaded`	when permitted the flags will be adapted
`enforced`	all is permitted (but only in zero mode or 'initex' mode)
`aliased`	the macro gets the same flags as the original

For the first five the primitive state has no related prefix primitive; it is set by the engine itself. Maybe someday I will decide to permit defining primitives, which would take hardly any code to implement. Permanent macros are (as shown) those that we don't want users to redefine, and frozen ones are mildly protected. They can be redefined when the

\overloaded prefix is used. A mutable macro can always be redefined, think of temporary macros, while an immutable can never be redefined. The instance property is just a signal that we're dealing with an instance, which can be handy when we trace. The \aliased prefix will copy properties, so this:

```
\aliased\let\forgetaboutit\relax
```

makes \forgetaboutit a reference to the current meaning of \relax (because that is what \let does) but also protects it like a primitive (because that is what \relax is).

The \enforced prefix is special. It only has a meaning inside a macro body or token register and it gets converted in a (hidden) \always prefix when in so-called ini mode (when the format is made). This permits system macros to overload in spite of heavy protection against it. Think of macros like \NC where the meaning can differ depending on the kind of table mechanism used, or \item which can differ by environment. We can protect these against overloading by the user but still redefine them. Of course, when the overload mode is zero, all can be redefined.

The value of \overloadmode determines to what extent a user will be annoyed when an existing macro is redefined, as shown in the table below. That can also be an instance defined by commands like \definehighlight although these normally are just \frozen \instance which means that a low level of protection only issues a warning.

		immut-able	perm-anent	prim-itive	frozen	instance
1	warning	⋆	⋆	⋆		
2	error	⋆	⋆	⋆		
3	warning	⋆	⋆	⋆	⋆	
4	error	⋆	⋆	⋆	⋆	
5	warning	⋆	⋆	⋆	⋆	⋆
6	error	⋆	⋆	⋆	⋆	⋆

The even values (except zero) will abort the run. A value of 255 will freeze this parameter. At level five and above the instance flag is also checked but no drastic action takes place. We use this to signal to the user that a specific instance is redefined (of course the definition macros can check for that too).

4 Alignments

In ConTEXt many commands are defined using the prefix \protected, which is handy when they are used in a context where expansion would not work out well, like writing to file or inside an \edef. However, this is impossible when we use the alignment mechanism. This has to do with the fact that the parser looks ahead to see if we have (for instance) a \noalign primitive. And since the parser doesn't look inside a \protected macro, this fails:

```
\protected\def\MyMacro{\noalign{\vskip 10pt}}
```

It also works out badly for macros that look for arguments. A dirty trick is:

```
\def\MyMacroA{\noalign\bgroup\MyMacroB}
\def\MyMacroB{\dosingleempty\MyMacroC}
\def\MyMacroC[#1]{....\egroup}
```

This somewhat over the top approach can now (in LuaMetaTEX) be simplified to the following. Let's also go crazy with prefixes here:

```
\noaligned\permanent\tolerant\protected
\def\MyMacroA[#1]%
  {\noalign\bgroup....\egroup}
```

For the record: in LuaMetaTEX the \noalign construct can be nested which again simplifies some (ConTEXt) code. Keep in mind that until now we could do whatever we wanted in traditional TEX speak, apart from making such macros \protected.

5 Definitions

From the perspective of the above it will become clear that in a system like ConTEXt quite a number of definitions are candidates for being flagged. You also need to think of symbolic character names or math symbols. For instance dimensions defined by \dimendef also get a permanent status. This means that one cannot redefine \scratchcounter but still its value can be changed. At this moment I see no reason to have a flag for preventing that (also because it would add overhead), but it might become an option some day.

However, there are often quantities that need overload protection, such as constant values. This is why we have:

```
\immutable \integerdef    \plusone   1
\immutable \dimensiondef  \onepoint  1pt
\immutable \gluespecdef   \zeroskip  0pt plus0pt
                                             minus0pt
\immutable \mugluespecdef \onemuskip 1mu
```

Those will never change and are a macro-like variant of registers but with an efficient storage model and behaving like a register. But one cannot use the operators like \advance on them. Their intended usage is as a constant.

Another definition-related extension involves \csname. In LuaTEX we introduced more robust handling of \ifcsname as well as an extra accessor:

```
\ifcsname f o o\endcsname
 \lastnamedcs % reference to the constructed \cs
\fi
```

as well as:

```
\begincsname f o o\endcsname
```

which doesn't define \f o o as a "relaxed" macro when it doesn't already exist. Both \begincsname and `\lastnamedcs` avoid a second name construction, as in:

```
\ifcsname f o o\endcsname
 \csname f o o\endcsname
\fi
```

Keep in mind that these additions are a side effect of control sequences being in UTF-8 format so we want to avoid unnecessary construction of temporary strings and related expansion.

Original TEX only has `\csname`; ε-TEX and LuaTEX added some companion primitives to that, and LuaMetaTEX again extends the repertoire:

```
\letcsname  f o o\endcsname\relax
\defcsname  f o o\endcsname{...}
\edefcsname f o o\endcsname{...}
\gdefcsname f o o\endcsname{...}
\xdefcsname f o o\endcsname{...}
```

This saves passing some arguments to a helper like `\setvalue` which is a bit more efficient and it also saves a token. (The ConTEXt format file became quite a bit smaller when the extensions discussed here were applied.) The `\ifcsname` primitive has been made somewhat more efficient by honoring macros that were defined as `\protected` which (we think) means: don't expand me in those cases where it makes no sense. So here we have an (in my opinion) acceptable downward incompatibility with engines that conform to ε-TEX.

There are a few more definition related new primitives, like:

```
 % shortcut for \global\let:
\glet\MyMacroA\MyMacroB
 % also works for registers:
\swapcsvalues\MyMacroA\MyMacroB
\futuredef\DoWhatever\MyMacro{...}
 % \protected like this:
\expand\MyProtectedMacro
```

6 Arguments

Let's start with a teaser. A previous definition needed a helper to gobble one of two arguments. The following does the same but it just gobbles and doesn't store the argument, which is why we use `#1` in both cases. This avoids storing token lists for the unused arguments.

```
\permanent\def\firstoftwoarguments #1#-{#1}
\permanent\def\secondoftwoarguments#-#1{#1}
```

Because anything other than a digit after a `#` triggers an error I saw no reason not to support some more: it doesn't hurt downward compatibility, unless you use TEX to generate error messages. Here is the full list of extensions, of which I will discuss a few (more can be found in the ConTEXt distribution and source code).

`+`	keep the braces
`-`	discard and don't count the argument
`/`	remove leading and trailing spaces and pars
`=`	braces are mandatory
`_`	braces are mandatory and kept
`^`	keep leading spaces
`1-9`	an argument
`0`	discard but count the argument
`*`	ignore spaces
`:`	pick up scanning here
`;`	quit scanning

We have a few useful characters left, such as `<` and `>` so who knows what future extensions might show up.

Delimited arguments are used frequently in ConTEXt; take this:

```
\def\MyMacro[#1][#2]{...}
```

Here the call is rather sensitive, for instance this will fail:

```
\MyMacro[A] [B]
```

We can cheat and define:

```
\def\MyMacro[#1]#2[#3]{...}
```

in which case `#2` gets what sits between the brackets. But still these two arguments have to be given. So, in MkII and MkIV you will find indirectness like the following:

```
\def\MyMacro{\dodoubleempty\doMyMacro}
\def\doMyMacro[#1][#2]{}
```

However, in LMTX you can find this alternative:

```
\tolerant\def\MyMacro[#1]#*[#2]{...}
```

The `\tolerant` will make the parser quit when no match can be made and the `#*` will gobble spaces. In fact, we often do this:

```
\tolerant\protected\def\MyMacro[#1]#*[#2]{...}
```

and if we want overload protection:

```
\permanent\tolerant\protected
 \def\MyMacro[#1]#*[#2]{...}
```

The combination of `\tolerant` and `\protected` with either expansion or not of a macro gives four variants of low-level macro commands: *normal*, *tolerant normal*, *protected* and *tolerant protected*. In LuaTEX that protection against expansion is implemented in a more indirect way, just like in ε-TEX. There we also have `\long` and `\outer` properties so we have *normal*, *long normal*, *outer normal* and *long outer normal*. Making protected against expansion a native command would have given another four command codes. Combining that with tolerant would again double it so we then would end up with 16 command codes. But in LuaMetaTEX we dropped the long and outer properties. In ConTEXt we never used outer and always want long anyway.

The reason for mentioning these details is to make clear that the introduced overhead can be neglected when we compare to LuaTEX, apart from the fact that we gain from the expansion protection being a first class feature now, macros without arguments being stored more efficiently, the parser being a little optimized and so on.

But of course the biggest benefit is that, when we look at the example above, we avoid indirectness. It looks nicer. It gives less clutter in tracing. It takes fewer tokens in the format (where each token takes eight bytes). It runs a little faster. It demands no trickery. Take your choice. For the record: you don't want to know what the set of `\dodoubleempty` macros looks like, as they themselves use indirectness and are highly optimized for performance.

The list of possible features has more than skipping spaces. Here's another example:

```
\tolerant\def\MyMacro[#1]#;(#2){<#1#2>}
```

Here `\MyMacro` accepts `[A]` and then quits or when not seen, checks for `(A)` and when not found is still happy. So, either `#1` or `#2` has a value. How do we know what arguments got grabbed? There are several ways to find out:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifarguments
     % zero arguments
   \or
     % one argument
   \else
     % two arguments
   \fi}
```

This test uses the count from the last expansion so if any macro expansion happens before the test you can get the wrong value! The next test provides feedback about what argument got a value:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifparameters
     % all empty
   \or
     % first has value
   \else
     % second has value
   \fi}
```

But still may not be enough so we can also explicitly test for a parameter. But again be aware of nesting:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifparameter#1\or
     % first has value
   \fi
   \ifparameter#2\or
     % second has value
   \fi}
```

This is pretty robust but expands the arguments in the test:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\unless\iftok{#1}{}%
     % first has value
   \fi
   \unless\iftok{#2}{}%
     % second has value
   \fi}
```

When we use a colon instead of a semicolon the parser knows where to pick up after a match fails:

```
\tolerant\def\MyMacro[#1]#:#2{...}
```

So, the argument between brackets is optional and the single token or braced second argument (turned into a token list) is mandatory.

The other extensions more or less speak for themselves: they grab arguments and discard or keep braces and such in cases where TEX would treat them specially when storing or passing them on. Speaking of braces, in spite of what one might expect (assuming that braces are more a TEX thing than brackets) the following two definitions perform equally well

```
\def\foo[#1]{} \foo[1]
\def\foo  #1{} \foo{1}
```

but:

```
\def\oof[#1]{}
\def\foo{\dosingleempty\oof}
```

performs more than 5 times worse than this:

```
\tolerant\def\foo[#1]{}
```

So, the added overhead (and there is some, also because we keep track of more) in the argument parser gets compensated well by the fact that we can avoid indirectness. The impact on an average document probably goes unnoticed.

As with much in TEX you need to be aware of (intentional) side effects. Take for instance:

```
\tolerant\def\foo#1[#2]#*[#3]{\edef\ofo{#1}}
\def\oof{\foo{oeps}}
```

That will probably not do what you expect. It has to do with how TEX interprets spaces in the context of argument parsing: they can become part of the argument (here `#1`) so anything before the first seen left bracket becomes the argument's value.

```
\tolerant\def\foo#1#*[#2]#*[#3]{\edef\ofo{#1}}
\def\oof{\foo{oeps}}
```

This however works because the first `#*` directive stops scanning for the first argument and then gobbles spaces when seen before continuing to look for the bracketed arguments. So TEX's charm is still there.

7 Introspection

Because macros have more properties and variation in arguments the `\meaning` command has a companion `\meaningfull` that displays what prefixes were applied. The `\meaningless` variant only shows the body.

Quite some effort went into normalizing the so-called command codes. Primitives are grouped into categories with similar treatments in order to keep the main loop efficient. These codes also determine the expansion contexts (think of usage in an `\edef`, how they get serialized (for instance in messages), etc. The char codes (called such because in most cases tokens represent characters of some kind) distinguish commands in these groups. Think of `\def` and `\edef` being call commands with a different code. This rather intrusive (internal) regrouping of primitives was needed in order to get a more consistent Lua token interface. So, for instance the codes are now in consecutive ranges, registers are split into internal and user variants, etc.

Also, memory management has been overhauled so we have a more dynamic allocation of various data structures (stacks, equivalents, tokens, nodes, etc.) and we use the whole 64 bit memory word to save some memory in places too. All this is the reason why it is unlikely that much will get backported to LuaTEX, also because in ConTEXt we now have a special version for LuaMetaTEX: LMTX.

8 There is more

Here we've discussed only the primitives that make the source look better while also being convenient. But it is worth mentioning that there are primitives like `\toksapp` and `\etokspre` that append and prepend tokens to a register (there are eight variants). There are ways to collect tokens for just before or after a group ends. There are some new expansion related primitives like `\expandtoken` that can be used to inject a token with some specific catcode, just like one can define active characters without the need for dirty uppercase tricks.

The typesetting department also has extensions. We can freeze paragraph properties, adjust math parameters locally, normalize lines so that at the Lua end we know what to expect (think of consistent presence of left and right skip, left and right shape related properties, left and right parfill skips, indentation being glue, etc.). Hyphenation can be controlled in more detail too, and left and right side ligatures and kerns can be influenced in the running text and go with glyphs. Talking of glyphs, there are advanced scaling options as well as support for influencing placement in the running text, which permits more efficient font handling. Boxes have more properties too: they can have offsets, an orientation, etc. which makes implementing vertical typesetting a bit easier. Rules also have shifts. We can register actions to be expanded at the end of a paragraph. All this evolved over time and has been tested in ConTEXt but will be applied more frequently after the complete code split between MkIV and LMTX. That process goes hand in hand with adapting to the new situation, remove old (obsolete) variants, removing still present experimental code, etc.

There is more but hopefully this gives an impression of how substantial the LuaMetaTEX engine differs (in added functionality) with its ancestors. Maybe it looks a bit over the top, but I did actually reject some ideas after experimenting with them. On the other hand there are still some on the agenda. For instance the engine can migrate and carry around so-called deeply buried inserts pretty well now but dealing with inserts could be made a bit easier (think of columns). So, we're not done yet.

It should be noted that contrary to what one might expect the code base is still quite okay and the binary stays well below 3 MB. In the meantime memory management is also improved and the format file got smaller. A lot of the internal reorganization relates to the fact that we have a Lua interface and exposing internals demands consistency, avoidance of (often clever) tricks, more abstraction, etc.

It is also worth noting that we can only do such a massive operation because users are willing to test intermediate versions (sometimes on very large projects) and because all changes in the code base are meticulously checked by Wolfgang Schuster who knows TEX and ConTEXt inside out. And of course we have Mojca Miklavec's compile farm to keep it available for all relevant platforms, where we use a mix of `gcc` (also with cross compilation), `clang` and `msvc` for various platforms, up to date. It definitely helps that compilation is fast (due to the refactored code base) and that I can use Visual Studio to work with the code.

In this summary I only covered some aspects of TEX. Another important set of extensions concerns the MetaPost library, where token scanners are exposed, more advanced Lua calls are possible and where no longer relevant bits of code have been removed. And we use the latest and greatest Lua 5.4 — but discussing the implications of these is for another article.

◇ Hans Hagen
`http://pragma-ade.com`

UTF-8 installations of CWEB

Igor Liferenko

Abstract

We show how to implement UTF-8 support in CWEB [1] by adding the arrays *xord* and *xchr*. Immediately after reading a Unicode character from an input file, we convert it to an 8-bit character using *xord*. On output the reverse operation is done using *xchr*. This allows us to leave core algorithms of CWEB unchanged.

Incidentally, the described method allows to use the extended character set [1] of CWEB: the characters '↑', '↓', '→', '≠', '≤', '≥', '≡', '∨', '∧', '⊂', and '⊃' can be typed as abbreviations for C language digraphs '++', '--', '->', '!=', '<=', '>=', '==', '||', '&&', '<<', and '>>', respectively.

1. Initialization

(For brevity, in the diffs following, the original code in the CWEB source is preceded with < characters, and the new code with >. Both are sometimes reformatted for presentation in this article, and for readability we sometimes leave a blank line between the pieces. The actual implementation uses the change files `comm-utf8.ch`, `cweav-utf8.ch` and `ctang-utf8.ch`, together with `common-utf8.ch` [2].)

First, we add global arrays *xord* and *xchr* to `common.w` [1]. We declare the size of the *xord* array to be 2^{16} bytes. This means that only values from the basic multilingual plane (BMP) of Unicode are permitted. We use the `wchar_t` data type for characters in input files to accommodate Unicode values.

Background: this predefined C type allocates four bytes per character (on most systems). Character constants of this type are written as `L'...'`.

```
unsigned char xord[65536];
wchar_t xchr[256];
```

These same arrays must be used in `cweave.w` [1].

```
extern unsigned char xord[];
extern wchar_t xchr[];
```

In `ctangle.w` [1] only the *xchr* array is needed.

```
extern wchar_t xchr[];
```

We initialize the *xord* and *xchr* arrays in the *common_init* function of `common.w`. First, in *xchr* we map all visible ASCII characters to themselves, like this:

```
xchr[32] = ' ';
```

Then we map the rest of the indexes of *xchr* to 127, which is the ASCII character code (`DEL`) that is prohibited in text files.

```
for (i=0; i<32; i++) xchr[i]=127;
for (i=127; i<=255; i++) xchr[i]=127;
```

Elements in the *xchr* array are overridden using the file `mapping.w` [2].

```
@i mapping.w
```

This file specifies the character(s) required for a particular installation of CWEB, for example:

```
xchr[0xf1] = L'ë';
```

The initialization of *xord* comes next. All its indexes are mapped by default to 127. Then we make it contain the inverse of the information in *xchr*.

```
for (i=0;i<=65535;i++) xord[i]=127;
for (i=0;i<=255;i++) xord[xchr[i]]=i;
xord[127]=127;
```

It remains to set the `LC_CTYPE` locale category. The behavior of the C library functions used below depends on this value.

```
setlocale(LC_CTYPE, "C.UTF-8");
```

Finally, we need the necessary headers.

```
#include <wchar.h>
#include <locale.h>
```

2. Input

For automatic conversion from UTF-8 to Unicode, we change the *input_ln* function to use *fgetwc* [3] instead of *getc*. Also, *ungetc* is changed to *ungetwc* [3] and `EOF` must be replaced with `WEOF` [3] (for this, `int` is changed to `wint_t` [3]).

```
< int c;
> wint_t c;

< while (k<=buffer_end && (c=getc(fp))
<   != EOF && c!='\n')
> while (k<=buffer_end && (c=fgetwc(fp))
>   != WEOF && c!=L'\n')

< if ((c=getc(fp))!=EOF && c!='\n') {
> if ((c=fgetwc(fp))!=WEOF && c!=L'\n') {

< ungetc(c,fp);
> ungetwc(c,fp);

< if (c==EOF && limit==buffer) return(0);
> if (c==WEOF && limit==buffer) return(0);
```

The conversion with *xord* is done immediately after a character is read.

```
< if ((*(k++) = c) != ' ') limit = k;
> if ((*(k++) = xord[c]) != ' ') limit = k;
```

doi.org/10.47397/tb/42-1/tb130liferenko-cweb

3. Output

We use *xchr* and *printf* with `%lc` conversion specifier for characters, printed on terminal during error reporting.

```
< putchar(*k);
> printf("%lc",xchr[(unsigned char)*k]);
```

The *term_write* macro uses the C library function *fwrite* to output a range of characters. We must use *xchr* for each character (except the newline character), then convert it to UTF-8 via *printf*, using `%lc` conversion specifier.

```
< @d term_write(a,b) fflush(stdout),
<      fwrite(a,sizeof(char),b,stdout)

> @d term_write(a,b) do { fflush(stdout);
>   for (int i = 0; i < b; i++)
>     if (*(a+i)=='\n') new_line;
>     else printf("%lc",xchr[(unsigned char)
>       *(a+i)]); } while (0)
```

In `cweave.w` all output to files is done via the *c_line_write* macro. This uses the C library function *fwrite* to output a range of characters. Since *xchr* must be used for each character, we loop over this range and convert each character to the external encoding and then to UTF-8 via *fprintf*, using the `%lc` conversion specifier.

```
< fwrite(out_buf+1,sizeof(char),c,
<   active_file)

> for (int i = 0; i < c; i++)
>   fprintf(active_file, "%lc",
>     xchr[(eight_bits) *(out_buf+1+i)])
```

Similarly, in `ctangle.w`, before outputting characters in C string constants, convert each of them to the external encoding and then to UTF-8 using the `%lc` conversion specifier of *fprintf*.

```
< C_putc(a);
> fprintf(C_file,"%lc",xchr[(eight_bits)a]);
```

We do not use the *translit* array when outputting non-ASCII characters in C identifiers. So, in `ctangle.w` we again convert each such character to the external encoding and then to UTF-8 via *fprintf* using the `%lc` conversion specifier.

```
< C_printf("%s",
<   translit[(unsigned char)(*j)-0200]);

> fprintf(C_file, "%lc",
>   xchr[(eight_bits) *j]);
```

For other output code no special treatment is needed, since all other output data is in ASCII, which is part of UTF-8 (except file names, which are already in UTF-8).

4. The file name buffer

File names must be in UTF-8. So, before appending characters to *cur_file_name*, we convert them to the external encoding and then to UTF-8 via C library function *wctomb* [3].

```
< *k++=*loc++;

> { char mb[MB_CUR_MAX]; int len =
>   wctomb(mb,xchr[(unsigned char)*loc++]);
> if (k<=cur_file_name_end)
>   for (int i = 0; i<len; i++) *k++=mb[i];
> else k=cur_file_name_end+1; }
```

5. Locale considerations

`cweave.w` uses the locale-dependent C library functions *islower*, *isupper* and *tolower* (the former two via *xislower* and *xisupper* macros respectively). But since we are assuming the UTF-8 locale, instead of these we must use *iswlower*, *iswupper* and *towlower* from `wctype.h` [3]. The trick is to convert from the internal encoding to the external encoding before using these functions.

```
< xislower(*x)
> iswlower(xchr[(eight_bits)*p])

< xisupper(x)
> iswupper(xchr[(eight_bits) x])
```

For *towlower* the result must be converted back from the external encoding to the internal encoding.

```
< c=tolower(c)
> c=xord[towlower(xchr[(eight_bits)c])]
```

References

[1] Knuth, D. and Levy, S. The CWEB System of Structured Documentation, 1993. ISBN 0-201-57569-8

[2] Source of the present implementation. `https://github.com/igor-liferenko/cweb`

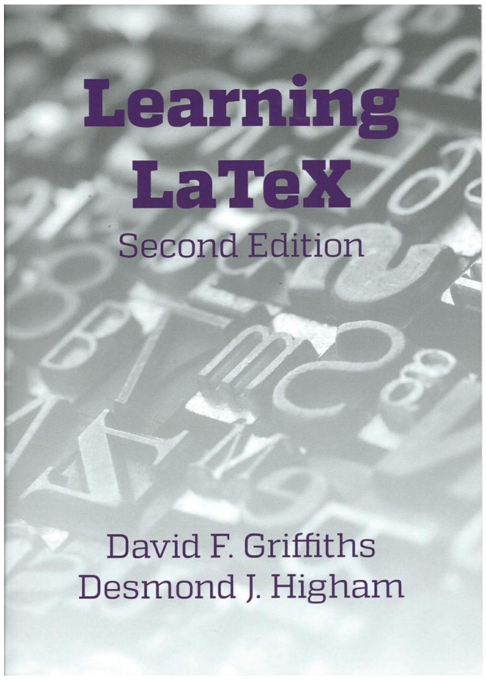
[3] Single Unix Specification. Introduction to ISO C Amendment 1 (Multibyte Support Environment). `https://unix.org/version2/whatsnew/login_mse.html`

◇ Igor Liferenko
`igor.liferenko (at) gmail dot com`

Book review: *Learning LaTeX*, Second Edition, by David F. Griffiths and Desmond J. Higham

Boris Veytsman

David F. Griffiths and Desmond J. Higham, *Learning LaTeX*, Second Edition. SIAM, Philadelphia, PA, USA, 2016, paperback, x+103pp., US$31.00, ISBN 978-1-611974-41-6.



There are two kinds of textbooks. First, there are formidable comprehensive books, covering all aspects of the subject. Even taking such a multi-pound book from its shelf is a hard task. An even harder task is reading such a book from cover to cover; only some courageous students can accomplish it. On the other hand, there are relatively slim books discussing the most important aspects of the topic and offering a student the important first step that starts the journey of a thousand li.

It is very difficult to write a book of the second kind: it requires the cruel skill of paring many pages of potential material, leaving only the essential parts. On the other hand, these books, if done right, have the important property of (near) immortality. The fundamentals of a field are long lasting. While the less essential material filling the thick volumes quickly becomes obsolete, the dog-eared slim introductory books are transferred from generation to generation, heavily read and consulted — until the progress makes even them too old. Every practitioner can name titles like these in their fields.

For several generations of LaTeX users, the book *Learning LaTeX* by Griffiths & Higham was such a title. Published in 1997, the green years of LaTeX 2_ε, this book has been an indispensable introduction to the subject, treasured by its readers. The book was reviewed on these pages in 2013 (*TUGboat* **34**:2, `tug.org/books/reviews/tb107reviews-learnltx.html`) and I have had the occasion to note how little in this book was obsolete.

The new edition is still slim, standing at 113 pages (compared to 94 pages in the first one). It adds important material about the *amsmath*, *beamer*, and *a0poster* packages, and the PDF typesetting workflow. On the other hand, the comparison between LaTeX 2_ε and LaTeX 2.09 is, of course, dropped, as well as the discussion of the *slides* document class. The discussion of Internet resources is updated.

What has persisted between editions is the authors' keen understanding of the essential features of LaTeX. Of course, some aficionados might object that their favorite parts did not survive the selection (No *tikz*? Fonts are not discussed? No information about Unicode engines?), but these objections, while understandable, would be ill-advised. This is not a book to make you a TeXnician (or LaTeXnician); this is the great introductory book to make you someone who understands LaTeX and can begin to find a way in this wonderful world of computer typesetting.

As in the previous edition, the authors' sense of humor shines through the pages. The self-referential examples ("*Don't* ***overuse*** *type-changing.* It annoys the READER. `And loses` ***`impact.`***"), its funny *Great Moments in LaTeX History* (expanded and renamed to *LaTeX through the years* in the new edition) are going to amuse new users as they did the previous ones. By the way, an updated collection of great moments can be found at `sinews.siam.org/Details-Page/writing-learning-latex`; my favorite is

> **2022:** Under current social distancing rules, the second component of every susceptible-infectious-recovered (SIR) model must be typeset as `dI/dt = \beta I \qquad S - \gamma I`.

I have no doubt this edition is going to be as beloved by new generations of LaTeX users as the previous one was by the old ones.

◇ Boris Veytsman
Systems Biology School, George Mason University, Fairfax, VA
`borisv (at) lk dot net`
`http://borisv.lk.net`
ORCID 0000-0003-4674-8113

doi.org/10.47397/tb/42-1/tb130reviews-learnltx2

Die TEXnische Komödie 4/2020–1/2021

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). Non-technical items are omitted.

Die TEXnische Komödie 4/2020

Harald König, ConTEXt meeting 2020: 6.–12. September in Sibřina nahe Prag [ConTEXt meeting 2020: 6–12 September in Sibřina near Prague]; pp. 12–19

The conference took place, as in 2018, at the former and beautifully renovated farmhouse of the Škoda family. It is a wonderful place of peace, where we could be alone and pursue TEXniques and other thoughts completely undisturbed.

Christoph Grüninger, Grafische Darstellung dünnbesetzter Matrizen [Graphical visualisation of sparse matrices]; pp. 20–26

Large, sparsely populated matrices occur in different fields of mathematics and their applications, for example in describing linear systems of equations. The matrices are too large to be printed or to be grasped by people in this form. Individual values are not relevant; instead, what matters are the structure and size of nonzero entries. With a few lines of Ti*k*Z code, graphical representations of large, sparse matrices can be derived from external files.

Wolfgang Beinert, Kolumnentitel [Column headings]; pp. 27–34

“Column heading” is a technical term in German typography for a page (page number) with or without associated text above, below or to the side of a column (type column), or within a design grid, or in classical handbooks and books of tables, outside a type area.

Alexander Krumeich, Biber für Alpine Linux – ein Docker-basiertes Buildsystem [Biber for Alpine Linux — a Docker-based build system]; pp. 34–38

In March 2019 I started the attempt to create a Docker image with TEX Live under Alpine Linux. For the documentation platform n-doc we create an environment in which the required programs and installed TEX packages are linked. A particular challenge was to determine the size of the image, to keep it small. Alpine Linux has proven to be a good choice for this in the past, since the base image is comparatively small: it is just under 5 MB in size, whereas the common Ubuntu image at this time has a size of over 80 MB.

Christine Römer, Neues in LATEX [News in LATEX]; pp. 39–42

The article gives a summary of current LATEX developments which are relevant for users. Updates for developers are omitted.

Die TEXnische Komödie 1/2021

Axel Kielhorn, Änderungsdokumentation mit ConTEXt [Change documentation with ConTEXt]; pp. 6–11

There are documents that have been finalized and — after being printed — do not change anymore. But there are also documents that are constantly changing, e.g., a specification document run through several iteration steps. When a new version appears, two questions arise: 1) What has changed?; 2) What is the current status?

Ideally, a change index provides information here and refers to the changed passages. Unfortunately, the change index often says “Various changes” and you have to hold the printouts up to the light to find the changes. The commands presented here are intended to make it easier for the author to find a good change index and also give the reader a quick overview of the given old and new versions.

Manfred Kraft, Interlinearüersetzung mit LATEX [Interlinear translations with LATEX]; pp. 12–16

For a number of years I have been translating Latin chemical and alchemical texts. For the translation and its documentation I use LATEX with Creutzig’s method of interlinear translation, for which a table-like structure for the bilingual presentation of a text is used. The foreign language and the German text are written on a pair of lines. The top line contains the foreign language text in normal type, the bottom line contains the German word-for-word translation in sans serif type. The words are set left-justified on top of each other in both lines.

Uwe Ziegenhagen, Ordnerrücken gestalten mit `ticket.sty` [Creating folder backs with `ticket.sty`]; pp. 17–19

In this article I would like to briefly introduce you to an easy way to design folder spines and other labels with `ticket.sty`.

Uwe Ziegenhagen, Ornamente in LATEX-Dokumenten mit `pgfornament` [Ornaments in LATEX documents with `pgfornament`]; pp. 20–22

I wanted nice looking certificates of attendance for a LATEX course I held, and used ornaments from the `pgfornament` package to achieve this.

Wolfgang Beinert, Giessbach; pp. 23–25

"Giessbach' is a German typographical term for a sketchy, poorly executed justification that results in whitespaces placed one below another, thus generating a small "river" of blank space. In a figurative sense, it looks like a pouring stream in the form of a "mountain stream with a waterfall".

[Received from Luzia Dietsche and Uwe Ziegenhagen.]

Zpravodaj 2020/3–4

Zpravodaj is the journal of CSTUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages. The full issue can be downloaded at `cstug.cz/bulletin`.

Petr Sojka, Úvodník [Introductory word]; pp. 105–107

In addition to introducing the content of this Zpravodaj issue, the author talks about changes to the CSTUG bylaws, about the Zpravodaj digitization project, and about the former CSTUG president Karel Horák.

Pavel Stříž, Sbohem, drahý Karle, sbohem! [Adieu, Monsieur Karel, adieu!]; pp. 108–117

Karel Horák departed from us on August 22, 2020 at the early age of 66.

Petr Sojka, Ondřej Sojka, Towards new Czechoslovak hyphenation patterns; pp. 118–126

Space- and time-effective segmentation and hyphenation of natural languages stay at the core of every document preparation system, web browser, or mobile rendering system. Recently, the unreasonable effectiveness of pattern generation has been shown — it is possible to use hyphenation patterns to solve the dictionary problem for a single language without compromise.

In this article, we will show how we applied the marvelous effectiveness of `patgen` for the generation of the new Czechoslovak hyphenation patterns that cover two languages. We show that the development of more universal hyphenation patterns is feasible, allows for significant quality improvements and space savings. We evaluate the new approach and the new Czechoslovak hyphenation patterns.

Keywords: hyphenation, hyphenation patterns, `patgen`, syllabification, syllabic hyphenation, Czech, Slovak, Czechoslovak patterns

Petr Olšák, OpTEX – pokračování maker OPmac pro LuaTEX [OpTEX — successor of OPmac macros for LuaTEX]; pp. 127–138

The article describes OpTEX — a LuaTEX format based on plain TEX and OPmac. This macro package was introduced in *TUGboat* and now it is presented for Czech and Slovak users. The presentation is slightly different than in the cited article. For example, the comparison with LATEX and ConTEXt is mentioned here including benchmarks in tables 1 and 2.

Keywords: OpTEX, LuaTEX, TEX format

Jano Kula, ConTEXt Meeting 2020; pp. 139–146

This is a short report about the ConTEXt Meeting 2020 in Sibřina near Prague.

Keywords: ConTEXt meeting

Taco Hoekwater, MetaPost definitions; pp. 147–167

The paper presents all syntactic rules for definitions in MetaPost. It contains many good and bad examples of definitions.

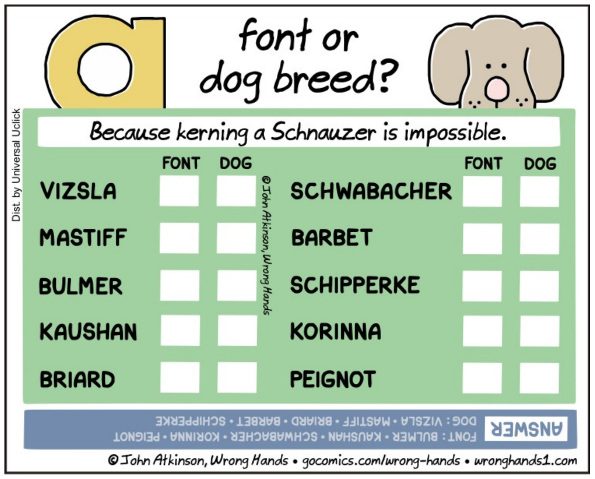
Keywords: MetaPost, definitions

Peter Wilson, It Might Work X; pp. 168–176

This paper shows an example how to use a Lua program to find the number of particular characters in a given text file. The second part of the paper shows ways to change the page layout in LATEX, with applications for typesetting a multi-page list of figures.

Keywords: Word counting, character frequency, Lua, text file, page layout, `quote` environment, `quotation` environment, `changepage` package, `memoir` class, list of figures

[Received from Vít Novotný.]


a
font or dog breed?
Because kerning a Schnauzer is impossible.
FONT DOG
VIZSLA
MASTIFF
BULMER
KAUSHAN
BRIARD
SCHWABACHER
BARBET
SCHIPPERKE
KORINNA
PEIGNOT
ANSWER
FONT: BULMER • KAUSHAN • SCHWABACHER • KORINNA • PEIGNOT
DOG: VIZSLA • MASTIFF • BRIARD • BARBET • SCHIPPERKE
Dist. by Universal Uclick
©John Atkinson, Wrong Hands
©John Atkinson, Wrong Hands • gocomics.com/wrong-hands • wronghands1.com

Comic by John Atkinson (`https://wronghands1.com`).

doi.org/10.47397/tb/42-1/tb130zprav

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from October 2020–April 2021. Descriptions are based on the announcements and edited for extreme brevity. One general notice: the CTAN multiplexor among mirrors is now available through `https://mirror.ctan.org`; previously it was only `http`.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*. A few entries which the editors subjectively believe to be of especially notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column helps people access the vast amount of material available through CTAN and the distributions. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`

fonts

Terse showings of some of these fonts, among others, are given on p. 65 of this issue.

`aesupp` in `fonts`
Variant form of italic 'æ' for some fonts.

`alfaslabone` in `fonts`
The Alfa Slab One slab serif font.

`archivo` in `fonts`
The Archivo sans serif font family.

`arvo` in `fonts`
The Arvo slab serif font family.

`atkinson` in `fonts`
The Atkinson Hyperlegible family of fonts.

`cascadia-code` in `fonts`
The Cascadia Code monospaced font family.

`eczar` in `fonts`
The Eczar font family for Devanagari and Latin.

`gudea` in `fonts`
The Gudea sans serif font family.

`hindmadurai` in `fonts`
The HindMadurai font family for Tamil.

`inter` in `fonts`
The Inter sans serif font family.

`magra` in `fonts`
The Magra sans serif font family.

`mlmodern` in `fonts`
Blacker Type 1 Computer Modern, with multilingual support.

`nunito` in `fonts`
The Nunito sans serif font family.

`oswald` in `fonts`
The Oswald sans serif font family.

`play-font` in `fonts`
The Play sans serif font family.

`rojud` in `fonts`
Type 1 font for the 42 counties of Romania.

`stepgreek` in `fonts`
A Greek font in the Times/Elsevier style, intended as a complement to STEP.

fonts/utilities

`frimurer` in `fonts/utilities`
Access to the frimurer cipher.

graphics

`causets` in `graphics/pgf/contrib`
Draw causal set (Hasse) diagrams.

`figchild` in `graphics/pgf/contrib`
Pictures for creating children's activities.

`mahjong` in `fonts/utilities`
Typesetting mahjong tiles.

`nl-interval` in `graphics/pgf/contrib`
Represent intervals on the number line.

`puyotikz` in `graphics/pgf/contrib`
Board states of Puyo Puyo games.

`pxpic` in `fonts/utilities`
Draw pictures pixel by pixel.

`sankey` in `graphics/pgf/contrib`
Draw Sankey flow diagrams.

`syntaxdi` in `graphics/pgf/contrib`
Railroad syntax diagrams.

`tikz-among-us` in `graphics/pgf/contrib`
Create some AmongUs characters.

`tikz-bbox` in `graphics/pgf/contrib`
Precise determination of bounding boxes, including control points.

`tzplot` in `graphics/pgf/contrib`
Convenient abbreviations for Ti*k*Z graphs.

info

`amiweb2c-guide` in `info`
Installation guide for AmiWeb2c 3.1.

* `knuth-pdf` in `info`
PDF collection of typeset C/WEB sources and errata in the TEX system, both original (with section numbering unchanged) and change files (for TEX Live).

`startlatex2e` in `info`
Getting started with LATEX 2_ε.

language/japanese

`gckanbun` in `language/japanese`
Kanbun typesetting for upLATEX and LuaLATEX.

macros/latex/contrib

`association-matrix` in `macros/latex/contrib`
Create association matrices.
`bithesis` in `macros/latex/contrib`
Templates for the Beijing Inst. of Technology.
`buctthesis` in `macros/latex/contrib`
Beijing University of Chemical Technology thesis template.
`chifoot` in `macros/latex/contrib`
Chicago-style footnote formatting.
`color-edits` in `macros/latex/contrib`
Colored edits for multiple-author documents.
`datax` in `macros/latex/contrib`
Import individual data points via `pgfkeys`.
`dimnum` in `macros/latex/contrib`
Commands for dimensionless numbers.
`dynbrackets` in `macros/latex/contrib`
Simplify syntax of dynamic math brackets.
`easybook` in `macros/latex/contrib`
Typeset Chinese books or notes.
`easyfloats` in `macros/latex/contrib`
Easier interface to insert figures, tables, etc.
`econlipsum` in `macros/latex/contrib`
Blind text generator for economics articles.
`eq-fetchbbl` in `macros/latex/contrib`
Match Biblical passages to verses for `exerquiz`.
`eq-pin2corr` in `macros/latex/contrib`
Add PIN security to `exerquiz` 'Correct' buttons.
`foliono` in `macros/latex/contrib`
Use folio numbers instead of page numbers.
`froufrou` in `macros/latex/contrib`
Fancy section separators.
`graphpaper` in `macros/latex/contrib`
Generate various types of graph paper.
`gridpapers` in `macros/latex/contrib`
Graph paper backgrounds and color schemes.
`highlightlatex` in `macros/latex/contrib`
Syntax highlighting extended from `listings`.
`hindawi-latex-template` in `m/l/c`
Template for Hindawi journals.
`hitreport` in `macros/latex/contrib`
Harbin Institute of Technology Report template.
`jupynotex` in `macros/latex/contrib`
Include whole or partial Jupyter notebooks.
`langsci-affiliations` in `macros/latex/contrib`
Collect and order authors and affiliations.
`lectureslides` in `macros/latex/contrib`
Combine individual PDF files into one.
`matapli` in `macros/latex/contrib`
Class for the French journal MATAPLI.
`mindflow` in `macros/latex/contrib`
Add memo section separated from main text for ideas or annotations.
`mluexercise` in `macros/latex/contrib`
Exercises/homework at the Martin Luther University Halle-Wittenberg.
`muling` in `macros/latex/contrib`
M.A. thesis class for the Dept. of Linguistics, University of Mumbai.
`ninecolors` in `macros/latex/contrib`
Select colors with proper WCAG color contrast.
* `numerica` in `macros/latex/contrib`
Numerically evaluate mathematical expressions, implemented in LaTeX.
`orcidlink` in `macros/latex/contrib`
Insert hyperlinked ORCID logo.
`orientation` in `macros/latex/contrib`
Set page orientation with Dvips/Ghostscript (`ps2pdf`).
`pbalance` in `macros/latex/contrib`
Balance last page in two-column mode.
* `pdfmanagement-testphase` in `m/l/c`
Load new backend-independent LaTeX interface to handle many PDF feature; please test now, to ease incorporation into core LaTeX.
`principia` in `macros/latex/contrib`
Typeset the Peanese notation of *Principia Mathematica*.
`profcollege` in `macros/latex/contrib`
Support for French math teaching for students 11–16 years old.
`scrlayer-fancyhdr` in `macros/latex/contrib`
Combining `fancyhdr` with `scrlayer`.
`skeldoc` in `macros/latex/contrib`
Placeholders for unfinished documents.
`skills` in `macros/latex/contrib`
Create proficiency tests.
`suppose` in `macros/latex/contrib`
Abbreviate the word 'Suppose' for math usage.
`twemojis` in `macros/latex/contrib`
Use Twitter emojis as graphics.
`utfsym` in `macros/latex/contrib`
Provide Unicode symbols via hex code.

m/l/c/beamer-contrib/themes

`beamertheme-trigon` in `m/l/c/b-c/themes`
Modern, elegant, customizable Beamer theme.
`beamerthemelalic` in `m/l/c/b-c/themes`
Beamer theme for LALIC.

macros/latex/contrib/biblatex-contrib

`biblatex-license` in `m/l/c/biblatex-contrib`
Add license data to a bibliography.

macros/luatex/generic

`chinese-jfm` in `macros/luatex/generic`
JFM files for Chinese typesetting with `luatexja`.

`luakeys` in `macros/luatex/generic`
Lua module for parsing key–value options.

macros/luatex/latex

`innerscript` in `macros/luatex/latex`
Handle script spacing more like text math, and treat `\mathinner` as `\mathord`.
`lua-typo` in `macros/luatex/latex`
Highlighting typographical flaws.
`newpax` in `macros/luatex/latex`
Handle PDF annotations, preserving links.
`semesterplanner` in `macros/luatex/latex`
Create semester plans/timetables.
`uninormalize` in `macros/luatex/latex`
Unicode normalization support for LuaLaTeX.

macros/plain

`xintsession` in `macros/plain/contrib`
Interactive calculation sessions, supporting arbitrary precision, polynomials, etc.

macros/unicodetex/latex

`aalok` in `macros/unicodetex/latex`
Class for the Marathi journal *Aalok*.
`beaulivre` in `macros/unicodetex/latex`
Class to typeset books, using `colorist`.
`colorist` in `macros/unicodetex/latex`
Base for articles or books with a colorful design.
`einfart` in `macros/unicodetex/latex`
Class to typeset articles, using `minimalist`.
`lebhart` in `macros/unicodetex/latex`
Class to typeset articles, using `colorist`.
`minimalist` in `macros/unicodetex/latex`
Base for articles or books with a simple design.
`quran-bn` in `macros/unicodetex/latex`
Bengali translations for the `quran` package.
`simplivre` in `macros/unicodetex/latex`
Class to typeset books, using `minimalist`.

macros/xetex/latex

`xesoul` in `macros/xetex/latex`
Use the `soul` package with XƎLaTeX.
`zbmath-review-template` in `macros/xetex/latex`
Template for a zbMATH Open review.

support

`albatross` in `support`
Find fonts that contain a given glyph.
`ltx2mathml` in `support`
Convert subset of LaTeX math to MathML.

TUG 2021 election report

Nominations for TUG President and the Board of Directors in 2021 have been received and validated. Because there is a single nomination for the office of President and because there are not more nominations for the Board of Directors than there are open seats, there is no requirement for a ballot this election.

For President, Boris Veytsman was nominated. As there were no other nominees, he is duly elected and will serve for a two-year term.

For the Board of Directors, the following individuals were nominated:

Karl Berry, Johannes Braams, Kaja Christiansen, Klaus Höppner, Frank Mittelbach, Ross Moore, Arthur Rosendahl.

As there were not more nominations than open positions, all the nominees are duly elected to a four-year term. Thanks to all for their willingness to serve.

Terms for both President and members of the Board of Directors will begin at the Annual Meeting.

Board members Taco Hoekwater, Will Robertson, and Herbert Voß have decided to step down at the end of this term. All have been TUG board members for many years, and their dedication and service to the community are gratefully acknowledged.

Election statements by all candidates are given below. They are also available online, along with announcements and results of previous elections.

◇ Barbara Beeton
for the Elections Committee
`tug.org/election`

Boris Veytsman

(Candidate for TUG President.)

I was born in 1964 in Odessa, Ukraine and have a degree in Theoretical Physics. I am a Principal Research Scientist with Chan Zuckerberg Initiative and an adjunct professor at George Mason University. I also do TeX consulting for a number of customers from publishers to universities to government agencies to non-profits. My current CV is available at `http://borisv.lk.net/cv/cv.html`.

I have been using TeX since 1994 and have been a TeX consultant since 2005. I have published a number of packages on CTAN and papers in *TUGboat*. I have been a Board member since 2010, Vice-President since 2016, and President since 2017. I am an Associate Editor of *TUGboat* and support `tug.org/books/`.

I consider my main goal as TUG President to keep TUG and TEX relevant in the changing world of typesetting. We are transitioning from the "user group" model: a mutual help association of users of an arcane software — to the model of a tech society entrusted with the support, advocacy and preservation of an important piece of publishing and communication infrastructure. The Board is working on this, and I try to help in the effort. We coordinate the work of developers and support important forums for developers and users: *TUGboat* and conferences. Our flagship publication, *TUGboat*, is now assigning DOIs to its papers, and TUG is a member of Crossref. We joined Open Software Initiative as an affiliated member. I have spent some effort in increasing TUG presence in social media. We started to apply for grants and received the first one for the accessibility initiative.

I am honored by the trust of TEX community and hope it allows me to continue this work.

Karl Berry

(Candidate for TUG Board of Directors.)

TEX biography: I served as TUG president from 2003–2011 and was a board member both before and after being president. I am running again for a position on the board.

I'm one of the primary system administrators and webmasters for the TUG servers, and the production manager for our journal *TUGboat*. I co-sponsored the creation of the TEX Development Fund in 2002.

On the development side, I'm currently the editor of TEX Live, the largest free software TEX distribution, and thus coordinate with many other TEX projects around the world, such as CTAN, LATEX, and pdfTEX. I developed and still (co-)maintain Web2c (Unix TEX) and its basic library Kpathsea, Eplain (a macro package extending plain TEX), and other projects. I am also a co-author of *TEX for the Impatient*, an early comprehensive book on plain TEX, now freely available. I first encountered and installed TEX in 1982, as a college undergraduate.

Statement of intent: I believe TUG can best serve its members and the general TEX community by working in partnership with the other TEX user groups worldwide, and sponsoring projects and conferences that will increase interest in and use of TEX. I've been fortunate to be able to work pro bono on TUG and TEX activities the past several years, and plan to continue doing so if re-elected.

Johannes Braams

(Candidate for TUG Board of Directors.)

Biography: I encountered TEX and friends sometime around 1985 when it was installed on our research VAX. It didn't take long for me to get hooked on LATEX and I started to think about and work on multilingual support, later to be known as *babel*. Besides that I have been active on quite a number of activities:

- For TEX and the TEX User Group
  - Co-founder and board member of the NTG, the Dutch-speaking TEX User group
  - As chairman of NTG I have served on the board of directors in 1994/1995 as special director for the NTG
  - designer of a couple of PhD-theses
  - Original author of the *babel* language support system
  - Member of the LATEX3 team, author of one of the chapters in the *LATEX Companion*
  - author, co-author or maintainer of a couple of publicly available LATEX packages and classes
- Professionally
  - System administrator for VAX/VMS and Unix systems
  - Manager of a team of System administrators
  - Functional administrator of one of the large administrative systems of KPN (PTT Telecom back then)
  - Project manager for various IT-projects within KPN for about 18 years
  - Consultant in the area of cyber security in Industrial Control Systems since 2015

Statement: In the last couple of years I have been coming back into the TEX-community. I would very much like to keep serving this wonderful community by continuing my role in the board of directors of TUG. I think TEX should and will be alive and well for many years to come, as the quality of typesetting that can be achieved by using TEX is still unsurpassed.

Kaja Christiansen

(Candidate for TUG Board of Directors.)

I was born in Warszawa, Poland and live in the city of Aarhus, Denmark. I heard about TEX for the first time in the fall of 1979. In Palo Alto at the time, I wanted to audit courses at Stanford and my top priority was lectures by Prof. Donald Knuth. That, I was told, was not possible as Prof. Knuth was on leave due to work on a text processing project... This project was TEX! Back home, it didn't take long till we had a runnable TEX system in Denmark.

I have served as a Board member since 1997, co-sponsored the creation of the TEX Development Fund and have been the chair of TUG's Technical Council since 1999. I am also a member of the Bursary and Election committees and served as TUG vice-president from 2003–2011. I share system administrator's responsibilities for the TUG server and TUG's web site, and actively contributed to several earlier versions of TEXlive. Finally, I am a board member of the Danish TEX Users Group (DK-TUG) and served as the president of DK-TUG in 2002–2011.

Statement: TEX and friends are the only software I know of that, after 30+ years, is not only alive and well, but also the best typesetting system to produce beautiful books and papers. In my rôle as a member of the board, my special interests have been projects of immediate value to the TEX community, among them system administration, TEX Live and *TUGboat*.

Klaus Höppner

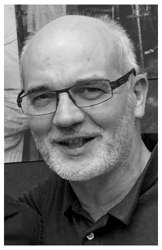
(Candidate for TUG Board of Directors.)

Biography: I got a PhD in Physics in 1997. After several years in the control systems group of an accelerator center in Darmstadt, I've been working at an accelerator for cancer therapy in Heidelberg. My first contact to LATEX was in 1991, using it frequently since then.

I have been preparing the CTAN snapshot on CD, distributed to the members of many user groups, from 1999 until 2002. I was the local organizer of TUG2015 and was heavily involved in the organization of several DANTE conferences and EuroTEX 2005. I've been a member of the TUG board since 2005 and was a member of the DANTE board until 2016, including terms acting as president, vice president, and treasurer.

Statement: As in the past, I want to be the voice of European users, in particular those who need characters with funny accents. Last year was troublesome to the whole world, and the TEX community was heavily affected, too. I'm glad that TUG managed to hold the first online international TEX conference, and these days show that organization of users in groups and the cooperation between the groups is essential.

Frank Mittelbach



(Candidate for TUG Board of Directors.)

I came in contact with TEX in the mid-eighties and over the years TEX, LATEX and typography in general became a very important part of my life. In 1990 I took over the maintenance and further development of LATEX from Leslie Lamport and together with a small number of people (most notably David Carlisle, Chris Rowley and Rainer Schöpf at that time) we designed and implemented what became LATEX2e in 1994 — the LATEX you still essentially use today (even though it has undergone smaller modifications and improvements through by now 25 further releases).

Despite all predictions made during the last decades, TEX and LATEX are alive and kicking as proven by their (still?) strong use in various ways around the world.

Nevertheless the world has changed and is changing further and in that changing world user groups like TUG need to find their place and possibly reinvent themselves by redefining and reshaping their role. With my work on the TUG board I would like to help in that process and ensure a future for high quality typography as provided by TEX.

Ross Moore

(Candidate for TUG Board of Directors.)

I would like to continue to serve on the TUG Board. Having been active in the TEX community for roughly 25 years, writing code for packages, I've been attending TUG meetings on-and-off since 1997. As well as the past 4 years, previously I had been a board member, but with a short break 2015–2016.

Since 2017 I've been working, as promised then, on developing a LATEX implementation of 'Tagged PDF', primarily for technical documents implementation in (both hard and soft) scientific fields, as well as more generally.

A motivation for this is the document "Information and Communication Technology (ICT) Final Standards and Guidelines" which was published into the US Federal Register, effectively becoming law. As well as much else, it outlines "standards for electronic and information technology developed, procured, maintained, or used by Federal agencies covered by section 508 of the Rehabilitation Act of 1973, as well as guidelines for telecommunications equipment [...] intended to ensure that information and communication technology covered by the respective statutes is accessible to and usable by individuals with disabilities." It specifies that "authoring tools capable of exporting PDF files must conform to PDF 1.7 [...] and be capable of exporting PDF files that conform to PDF/UA-1."

While documents produced by TEX-based software are compatible with PDF 1.7, it is certainly not the case that they conform to PDF/UA, which requires producing 'Tagged PDF'. Thus if LATEX or other TEX-based software can be used within US government agencies only as part of a processing chain requiring other software to complete the final document. This is not how we normally work with TEX. Furthermore, the PDF 2.0 standard, also based upon 'Tagged PDF', was finally published in December 2020, after several years in development.

Over the past 12 years (or more), I have given talks at annual TUG meetings, both in-person and online, demonstrating PDF features that tagging allows, produced using an extended version of pdfTEX. Currently I'm working on a set of LATEX macros that produce valid 'Tagged PDF' documents simultaneously satisfying PDF/UA and PDF/A-3a, hence meeting the accepted WCAG 2.0 Accessibility standards. Examples, using different LATEX document classes and built with the current pdfTEX, can be found on my web page `http://maths.mq.edu.au/~ross/TaggedPDF/`. These developments need to be further enhanced to become *de rigeur* for the way we use TEX and LATEX.

As a Director of TUG, this outlines an agenda that I'll be supporting.

Arthur Rosendahl

(Candidate for TUG Board of Directors.)

Biography: I first joined TUG in 2005 and have been a member of the board for the past four years. My interest in TEX started during my university years when, being a student of mathematics and physics, I had to use it for typesetting reports. I was prompted to dive deeper because of my interest in languages and writing systems, and soon got passionate about it (see my entry in the TUG interview corner for excruciating details).

Statement: Today, I am active in TEX development as the maintainer of various packages and programs related to multilingual typesetting (polyglossia, hyph-utf8, XƎTEX), and I regularly give talks at conferences and write articles in journals. I am on the board of several TEX users groups, founded the ConTEXt group, and have contributed to promoting TEX which in my opinion is, still today, one of the best typesetting systems available. I am truly amazed at how vibrant our user community is.

I've been vice president of TUG since 2017, and in 2021 I'm in charge of organising our yearly conference, which again will be online. If I am reelected to the TUG board, I will do my best to represent that community in all its diversity and foster the use of TEX and Metafont in the 21st century.

Spending MacTEX funds

Richard Koch

1 The creation of MacTEX

At the 2005 TUG conference in Chapel Hill, North Carolina, Wendy McKay organized a lunch for Mac users and essentially willed MacTEX into existence then and there. She declared a "MacTEX group" with no official membership or rules; apparently we became charter members, and development began. On the form to join or renew membership with TUG, Wendy requested a special line for contributions to MacTEX, as already existed for LATEX and a few other projects.

2 Spending the MacTEX Fund

Contributions to the fund were slow at first, but it has grown to several thousand dollars, and today is one of the most popular funds administered by TUG.

As the fund grew, TUG directors suggested that we find ways to spend the money. My immediate reaction was that we should use it to hire programmers to fix a few thorny issues that had come up in the Mac world. This suggestion was immediately shot down when TUG folks pointed out that a thousand dollars is an impressive contribution, but will pay a programmer's salary for only a week (or less). Gradually I learned that the TEX world is an open source miracle, created by thousands of men and women whose only reward is the gratitude of the people using their software. We cannot afford to hire programmers, but we can use the fund to smooth a few bumps along the way.

3 Specifics

Jonathan Kew originally wrote XƎTEX for the Macintosh. XƎTEX could use ordinary Mac fonts directly because Apple had a special framework to handle fonts, and the tables in TEX for kerning and the like could be replaced by calls to that framework. Later Kew extended XƎTEX to work on most platforms; then his day job changed and he had less time for TEX.

All was well until Apple decided to rewrite the font framework for a new version of Mac OS X. The framework used by Kew became "deprecated". Both old and new frameworks were shipped, but eventually the deprecated framework would be dropped and XƎTEX would stop working on the Macintosh. What to do?

We struggled with this problem for years. Then Khaled Hosny miraculously appeared. He typeset Arabic using XƎTEX, and understood XƎTEX internals, but he didn't own a Macintosh. So we bought one for him using money from the fund. Soon XƎTEX used the new font library. Problem solved!

4 LaTeXiT and TEX Live Utility

The MacTEX installer includes the widely used GUI programs 'LaTeXiT' by Pierre Chatelier and 'TEX Live Utility' by Adam Maxwell. Three years ago, Apple started requiring that install packages be notarized. Any application in a notarized package must be signed by an Apple developer and adopt a hardened runtime. It costs $100 a year to be an Apple developer, and adopting a hardened runtime requires XCode running on recent hardware. Chatelier and Maxwell objected to that $100 per year on principle and were happy with their current machines. So the programs had to be omitted from recent releases of MacTEX.

We offered to buy new hardware and pay the developer fee out of the fund, but both programmers were reluctant. We kept trying, and eventually, as the need for their programs never lessened, the authors relented. As of 2021, their programs are back in MacTEX, and have code with both ARM and Intel hardware support, thanks to the fund.

5 Bringing people to TUG conferences

In a small number of cases we paid TUG conference fees for users, and in one case we paid transportation fees from France to the US. That participant was Bruno Voisin, who was and is extremely active on Mac mailing lists. We wanted to meet him.

This was a wise use of money: Bruno is responsible for the Ghostscript we ship each year with MacTEX, including compilation instructions for CJK fonts in the Far East (working with Norbert Preining), and keeping track of Ghostscript's new transparency operators (with Herbert Schulz). Now the MacTEX build instructions for Ghostscript have folders labeled "Bruno-Preining Email" and "Bruno Contributions" and a document named "How to Compile Ghostscript" with instructions from Bruno which I mechanically follow to create the package.

6 Lesson

In the open source world, my colleagues and I in the MacTEX group do not know how to invent tasks and hire people. But if others are already doing the work, we may be able to smooth a rough spot or two using generous contributions so many people have made to the MacTEX Fund. Thanks to all!

◇ Richard Koch
`koch (at) uoregon dot edu`
`https://tug.org/mactex`

TUG financial statements for 2020

Karl Berry, TUG treasurer

The financial statements for 2020 have been reviewed by the TUG board but have not been audited. Totals may vary slightly due to rounding. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site: `https://tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue was essentially the same in 2020 compared to 2019. We ended the year with 1,189 members (23 fewer than 2019). The 2020 online conference had a net gain of about $3,700, due to generous donations and few expenses. Contributions were down about $2,000, and other categories were slightly down. Overall, 2020 income was up about 2%.

Other highlights; the bottom line

TUGboat production cost was up a little, due to page count. Postage-related expenses increased; other categories remained about the same.

The bottom line for 2020 was strongly negative, about $7,300, though still an improvement over 2019.

Balance sheet highlights

TUG's end-of-year asset total is up by around $3,000 (2%) in 2020 compared to 2019, due primarily to conference and committed fund donations.

Committed Funds are reserved for designated projects: LaTeX, CTAN, MacTeX, the TeX development fund, and others (`https://tug.org/donate`). Incoming donations are allocated accordingly and disbursed as the projects progress. TUG charges no overhead for administering these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2020 portion of this liability was converted into regular Membership Dues in January of 2020. The payroll liabilities are for 2020 state and federal taxes due January 15, 2021.

Upcoming

For 2020, we have reduced the electronic membership discount to $30, closer to the actual cost reduction of not shipping physical benefits. All other rates and fees remain the same. We hope to gain members this year; ideas are always welcome!

◇ Karl Berry, TUG treasurer
`https://tug.org/tax-exempt`

TUG 12/31/2020 (vs. 2019) Revenue, Expense

	Dec 31, 20	Dec 31, 19
ORDINARY INCOME/EXPENSE		
Income		
Membership Dues	76,030	76,125
Product Sales	3,761	5,238
Contributions Income	11,830	13,995
Annual Conference	3,721	(2,685)
Interest Income	1,430	1,934
Advertising Income	305	345
Total Income	97,078	94,952
Cost of Goods Sold		
TUGboat Prod/Mailing	(20,312)	(18,836)
Software Prod/Mailing	(2,256)	(2,194)
Members Postage/Delivery	(2,759)	(2,236)
Lucida Sales to B&H	(1,525)	(1,965)
Member Renewal	(356)	(420)
Total COGS	(27,208)	(25,651)
Gross Profit	69,870	69,301
Expense		
Contributions made by TUG	(2,000)	(1,000)
Office Overhead	(12,830)	(13,642)
Payroll Expense	(64,135)	(63,091)
Interest Expense	0	(24)
Total Expense	(78,965)	(77,757)
Net Ordinary Income	(9,095)	(8,850)
OTHER INCOME/EXPENSE		
Prior year adjustment	1,475	(78)
NET INCOME	(7,620)	(8,535)

TUG 12/31/2020 (vs. 2019) Balance Sheet

	Dec 31, 20	Dec 31, 19
ASSETS		
Current Assets		
Total Checking/Savings	174,197	171,560
Accounts Receivable	275	280
Total Current Assets	174,472	171,840
LIABILITIES & EQUITY		
Current Liabilities		
Committed Funds	57,652	47,270
Administrative Services	1,447	1,498
Prepaid Member Income	9,185	9,175
Payroll Liabilities	1,211	1,301
Total Current Liabilities	69,495	59,244
Equity		
Unrestricted	112,596	121,131
Net Income	(7,620)	(8,535)
Total Equity	104,977	112,596
TOTAL LIABILITIES & EQUITY	174,472	171,840

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants.html`. If you'd like to be listed, please see there.

Dangerous Curve
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TEX book.

Dominici, Massimiliano
Email: `info (at) typotexnica.it`
Web: `http://www.typotexnica.it`
Our skills: layout of books, journals, articles; creation of LATEX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

Monsurate, Rajiv
India
Email: `tex (at) rajivmonsurate.com`
Web: `https://www.rajivmonsurate.com`
I have over two decades of experience with LATEX in STM publishing working with full-service suppliers to the major academic publishers. I've built automated typesetting and conversion systems with LATEX and rendered TEX support for a major publisher.

I offer design, typesetting and conversion services for self-publishing authors. I can help with LATEX class/package development, conversion tools and training for publishers and typesetters for book and journal production. I can also help with full-stack web development.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: `michael.sofka (at) gmail.com`
Personalized, professional TEX and LATEX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TEX and LATEX: Automated document conversion; Programming in Perl, Python, C, R and other languages; Writing and customizing macro packages in TEX or LATEX, `knitr`.

If you have a specialized TEX or LATEX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
132 Warbler Ln.
Brisbane, CA 94005
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`

TeX and LaTeX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LaTeX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TeX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TeX-related journals, and conducted several workshops on TeX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TeX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake
90 Resaca Ave.
Box 452
Forest Knolls, CA 94933
+1 650-468-1393
Email: `jwarde (at) wardepub.com`
Web: `http://myprojectnotebook.com`

I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TeX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using LaTeX for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects. If you need assistance getting your manuscript set in TeX I can help. And if I cannot help I'll let you know right away.

TUG 2021	**15th ConTeXt Meeting**
online	**Bassenge, Belgium**
August 5–8, 2021	**Sept. 20–25, 2021**
tug.org/tug2021	**meeting.contextgarden.net**

Calendar

2021

Apr 30	Frederick W. Goudy Award Presentation and Lecture (online), Cary Graphic Arts Library, Rochester Institute of Technology, Rochester, New York. `www.rit.edu/events/goudy-award-presentation-and-lecture`
May 3 – 5	Association Typographique Internationale (ATypI) type technology forum (online), `www.atypi.org`
Jun 30 – Jul 2	Nineteenth International Conference on New Directions in the Humanities, "Critical Thinking, Soft Skills, and Technology", Universidad Complutense Madrid, Spain. (May be held online.) `thehumanities.com/2021-conference`
Jul 10	**Preprints due for TUG 2021 program**
Jul 12 – 16	International Society for the History and Theory of Intellectual Property (ISHTIP), 12th Annual Workshop, "Landmarks of Intellectual Property" (online), Bournemouth University, UK. `www.ishtip.org/?p=1027`
Jul 20 – 21	Centre for Printing History & Culture, CPHC/Print Networks Conference, "A visitor attraction: printing for tourists" (Webinar), Appleby-in-Westmorland, Cumbria, UK. `www.cphc.org.uk/events`
Jul 26 – 30	SHARP 2021, "Moving texts: From discovery to delivery". Society for the History of Authorship, Reading & Publishing. Hosted virtually by the University of Muenster. `www.sharpweb.org/main/conferences`
Aug 2 – 6	Balisage: The Markup Conference (online). `www.balisage.net`

TUG 2021 online
Presentations covering the TeX world

Aug 5	Workshops and tours.
Aug 6 – 8	The 42nd annual meeting of the TeX Users Group. `tug.org/tug2021`

Aug 9 – 13	SIGGRAPH 2021 (online). `s2021.siggraph.org`
Aug 15	**Final papers due for TUG 2021 proceedings**
Aug 18 – 22	TypeCon 2021, Philadelphia, Pennsylvania. `typecon.com`
Sep 10	The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. `prov.pub/updikeprize`
Sep 18	DANTE 2021 Herbsttagung, Saarbrücken, Germany. `www.dante.de/veranstaltungen/herbst2021`
Sep 20 – 25	15th International ConTeXt Meeting, "Expanding orbits", Bassenge, Belgium. `meeting.contextgarden.net/2021`
Oct 1 – 3	Oak Knoll Fest XXI, "Women in the Book Arts", New Castle, Delaware. `www.oakknoll.com/fest`
Oct	Association Typographique Internationale (ATypI) annual conference (online), Paris, France. `www.atypi.org`

2022

Summer	Digital Humanities 2022, Alliance of Digital Humanities Organizations, Tokyo, Japan. `adho.org/conference`

Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.

Status as of 15 April 2021

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, email: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at `tug.org/meetings.html`. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from `tug.org/calendar.html`.

TUGBOAT Volume 42 (2021), No. 1

Introductory

6 *Jacques André* / Hendrik Vervliet: 1923–2020
 • memorial and summary of work of this noted scholar of typography

4 *Barbara Beeton* / Editorial comments
 • typography and *TUGboat* news

10 *Karl Berry* / TEX entomology in 2021
 • filtering the incoming bugs for DEK's 2021 tune-up

65 *Karl Berry* / Some fonts with recent TEX support
 • terse showings of some fonts appearing on CTAN in the last year

18 *Peter Flynn* / Typographers' Inn
 • Page numbering revisited; Type 1 (PostScript) fonts; Afterthought

12 *Jérémy Just* / Year 2020 at GUTenberg
 • association is preserved by vote, but the need for more cooperation remains

3 *Boris Veytsman* / From the president
 • license usage on CTAN; dangers of non-commercial licensing

20 *David Walden* / Interview with Amelia Hugill-Fontanel
 • discussion with the associate curator at RIT's Cary Graphic Arts Collection, `cary.rit.edu`

Intermediate

86 *Karl Berry* / The treasure chest
 • new CTAN packages, October 2020–April 2021

50 *Jerzy Ludwichowski* / GUST e-foundry font projects, closing report 2019–2020
 • status of OpenType TEX Gyre fonts, including math, and new Algotype font production software

34 *Carla Maggi* / The DuckBoat — Beginners' Pond: Crazy Little Thing Called Glue
 • specifying and fixing spacing, horizontal and vertical; and floats

52 *Antonis Tsolomitis* / The NewComputerModern font family
 • OpenType fonts greatly extending Latin Modern, with complete coverage of Unicode math

40 *Joseph Wright* / Creating document commands: The good, the bad and the ugly
 • best practices for using `\NewDocumentCommand` from `xparse`

Intermediate Plus

13 *Barbara Beeton* / Lapses in TEX — a look backward
 • alternatives to past decisions that could have alleviated current problems

67 *Maxime Chupin* / Animating Fourier series decomposition of a character with LuaTEX and MPLIB
 • building an animation of a character's outline with Lua(TEX) and MetaPost

72 *Island of TEX* / Working remotely from an island: arara and other tools
 • build tool arara, font finder albatross, texdoc online

44 *Petr Olšák* / Comparison of OpTEX with other formats: LATEX and ConTEXt
 • differing concepts, implementations, usage of three formats

41 *Joseph Wright* / `\NewDocumentCommand` *versus* `\newcommand` *versus* ...
 • extended comparison of the `xparse` `\NewDocumentCommand` to traditional alternatives

Advanced

74 *Hans Hagen* / LuaMetaTEX programming features
 • advances and extensions in TEX programming in the `lmtx` engine

60 *Hans Hagen* / Scaled fonts and glyphs
 • engine extensions in `lmtx` for dynamic glyph scaling

66 *Hu Yajie* / Improvements to the generalized mediation macros in *The* METAFONT*book*
 • meeting the generalized mediation macro challenge in Appendix D of *The METAFONTbook*

7 *Donald Knuth* / The TEX tuneup of 2021
 • the 2021 updates to TEX, METAFONT, vols. A–E, et al.

81 *Igor Liferenko* / UTF-8 installations of CWEB
 • changing `cweave` and `ctangle` to read/write the Unicode BMP

56 *S.K. Venkatesan* / An attempt at creating font transitions
 • an example of mathematically interpolating between sans serif and serif fonts

Reports and notices

2 Institutional members

11 *Barbara Beeton* / Hyphenation exception log
 • update for missed and incorrect U.S. English hyphenations; see `ctan.org/pkg/hyphenex`

83 *Boris Veytsman* / Book review: *Learning LATEX, Second Edition*, by David F. Griffiths and Desmond J. Higham
 • learning LATEX by example, updated

84 From other TEX journals: *Die TEXnische Komödie* 4/2020–1/2021; *Zpravodaj* 2020/3–4

85 *John Atkinson* / Comic: font or dog breed?

88 *TUG Elections committee* / TUG 2021 election report

92 *Richard Koch* / Spending MacTEX funds

93 *Karl Berry* / TUG financial statements for 2020

94 TEX consulting and production services

96 Calendar