TUGboat

Volume 39, Number 3 / 2018

TEX Users Group

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

Individual memberships

2019 dues for individual members are as follows:

- Trial rate for new members: $20.
- 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 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 organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2019 is $110.

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: November 2018]

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

Technical support for TEX users:
support@tug.org

Contact the Board of Directors:
board@tug.org

The printer, he discovered, had the final say on how a piece of writing would be perceived. Those cold letters, forged in heat, sway the reading public in ways that even the most astute among them will never understand. Grayson understood, and he knew something else: that a printer need not be bound to the types offered by a foundry. A letter Q could be drawn a million ways, and he could create his own.

John Dunning
The Bookman's Wake (1995)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 39, NUMBER 3, 2018
PORTLAND, OREGON, U.S.A.

TUGboat editorial information

This regular issue (Vol. 39, No. 3) is the last issue of the 2018 volume year.

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 submission deadline for the first 2019 issue is March 31.

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, is 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!

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*

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

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*

MOSEK ApS, *Copenhagen*, *Denmark*

Nagwa Limited, *Windsor*, *UK*

New York University, Academic Computing Facility, *New York*, *New York*

Overleaf, *London*, *UK*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

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

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

TEXFolio, *Trivandrum*, *India*

TNQ, *Chennai*, *India*

University College Cork, Computer Centre, *Cork*, *Ireland*

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

I have been working in Silicon Valley for more than a year now. This place is famous for its startups, both for- and non-profit ones. There are many interesting tips to be learned here. I would like to share one of them.

Venture capital funders often propose to aspiring founders the following question. "Imagine money is not a problem: suppose you have a bottomless box of dollars in your office. What would you do?" I find this question very useful. Certainly money is a constraint for our (and everyone's) plans. However, if we imagine this constraint lifted, we can get a better grasp of what we ultimately want to achieve.

Maybe the following is a somewhat geeky comparison, but this exercise reminds me of a way to approach linear programming problems: we deal with constraints by first pretending there are none, and looking for the best direction to improve our target function. Only then do we reintroduce the constraints.

Now TUG is not a startup: we are an organization more than three decades old, a dinosaur by the Silicon Valley reckoning. Still I find it useful to think about the question. While the financial situation of TUG is stable, clearly there are limits to our available funds. Suppose this constraint is eliminated. What would we do?

I will list my preferences in no particular order. I would like to hear from other members about theirs. Please think for a minute what *you* would want TUG to do if money were not an issue.

There is the development fund. Right now we give small grants, from several hundreds to low thousands of dollars, to developers having interesting projects. This is definitely not enough. If we had money, we could completely change the policy. Instead of waiting for developers to suggest a project to us (and hoping they will finish it for the very small grants we can afford), we could actively lead the development process. We could identify the most important needs in the community (maybe scraping forums like TeX.sx and the texhax list) and offer substantial competitive grants to the developers. Of course, we should also give much more money to major existing projects such as CTAN, LaTeX3, TeX Live, accessibility, font development, ...

For conferences: we could increase our bursary fund and thus be able to be much more generous with helping (especially) students and members from developing countries to attend our meetings. We need many more new faces in our conferences if we want to stay viable.

Speaking of students, we could organize subsidized summer TeX camps for school and university students, develop teaching materials and reward the best teachers introducing TeX in the classroom.

For example, distance learning courses such as Coursera are quite popular, especially for programming related topics. A similar set of courses about TeX might help many novices and widen our user base. If we can make these courses free for the users, this would be a worthy project. However, we cannot expect teachers to do all the work for free. With unlimited funds, we could pay them a fair amount to create and possibly lead free distance learning courses for TeX users.

In general, the future of TeX, as for anything, is with the younger generation. We need to invest more in students and their teachers.

On another front, we could restart our book publications. TUG has produced a couple interesting books, but it has been eight years since the last of them. We could create a TeX showcase, publishing beautiful examples of TeX in action, demonstrating free fonts, interesting book design and, of course, good content.

It would be interesting to make a foray into electronic publishing. There have been many e-books made with TeX. A set of example books made by the best TeXnicians might further improve TeX's presence in this area. More could be done with *TUGboat* in this or other areas.

⋆⋆⋆

Well, this was an interesting exercise. Now we are back on Earth where our funds are limited. Still, the goals discussed here (and those I haven't thought of) are worth working for.

The task of fundraising, recruiting new members, convincing organizations to join is hard. It takes a considerable amount of time and effort. A dream like the one above helps to remind us why we are doing it.

I hope we can recruit new members and attract donations to realize at least some of these goals.

Happy TeXing!

◇ Boris Veytsman
TeX Users Group
borisv (at) lk dot net
http://borisv.lk.net

Editorial comments

Barbara Beeton

Passings:
Patricia Monohon
(30 May 1941–6 April 2018)
Vytas Statulevicius (†July 2018)

Patricia Monohon was a long-time member of TUG, and served on the board from 1997 through 2002. More importantly, she was responsible for moving the TUG office from Providence to San Francisco in 1993, and remained in charge of the office until 1997. During that period she was instrumental in selecting the sites for the annual meetings in Santa Barbara (1994) and San Francisco (1997). Patricia also proposed the site for the 2003 meeting on the Big Island, Hawaii, and served on the organizing committee.

Dr. Statulevicius was president of the Lithuanian TEX users group. He attended a number of TUG meetings along with colleagues from VTEX, a service organization that readies LATEX manuscripts for publication for numerous book and journal publishers.

TEX and the history of desktop publishing

I've never quite thought of TEX as a prime example of desktop publishing. However, it's quite true that TEX makes first-class typography available to anyone willing to expend the effort to learn how to use it properly. And it's also true that TEX is fully functional on one's desktop — thus a suitable candidate for an overview of the topic.

This was a pertinent question when the IEEE joined with the Computer History Museum in Menlo Park, California, in staging a two-day meeting on the History of Desktop Publishing in May 2017. Participants in the meeting representing TEX included Don Knuth, Chuck Bigelow, and Dave Walden; Dave's prior status as historian and editor of anecdotes for the *IEEE Annals of the History of Computing* was instrumental in having TEX considered for inclusion. The success of the meeting was sufficient to warrant a two-issue record in the *Annals*; the first of these two issues has just been published. (The TOC can be found at `https://www.computer.org/csdl/magazine/an/2018/03`.)

The relevant record of TEX's history in this context is the subject of a two-part article by Dave, Karl Berry, and myself. The first part, entitled "TEX: A branch in desktop publishing evolution, Part 1", covers the development and adoption of TEX until it was cut loose from Stanford, and appears in the first issue; the second part, covering the growth of the TEX user and developer community, will appear early next year in the second topical issue.

In addition to the published proceedings, the discussions and several ancillary interviews have been recorded and transcribed. This material is (or will be, when ready) posted online at `https://history.computer.org/annals/dtp/`. What I've read so far is fascinating. The history "extras" by and about the Seybolds, father (John) and son (Jonathan), reveal how much easier we have it now that computers are so much larger and faster.

I've been privileged to be involved at the AMS in efforts to bring the Society's publications from traditional typesetting to full composition by computer, and many of the names that appear in these recollections are familiar to me — I've even worked in various contexts with some of them. (For example, I remember the day I first heard the term "WYSIWYG", written on a blackboard in big, bold letters, at a meeting of the Graphic Computer Communications Association in Philadelphia. Even then, the approach desired by the industry was "structural" and content-driven, not based solely on appearance.) So, in this way, TEX is *not* a typical desktop word processor, but it is still personally accessible publishing, and thus a worthy member of this assemblage.

Open season for lectures on typography

This fall has been filled with events celebrating font design, typography, and typographers. I was pleased to attend three such events in Providence and the Boston area that turned out to be more closely related than one might have expected from their announcements.

Daniel Berkeley Updike and the Janson font

August 25 was the occasion for a lecture at the Museum of Printing in Haverhill, Massachusetts (`https://museumofprinting.org/`).

Daniel Berkeley Updike is best known as the founder and proprietor of the Merrymount Press in Boston. This press was one of the (if not *the*) most distinguished U.S. scholarly printing offices in the late nineteenth – early twentieth century. John Kristensen of Firefly Press (a small "fine press") told the story of Updike's printing of the authorized 1928 revision of *The Book of Common Prayer* using the Janson font.

Updike, a scholar and historian as well as a printer, chose a typeface from the seventeenth century, contemporary with earlier editions of the *Prayerbook*. The typeface was one attributed (mistakenly) to Anton Janson, and given his name. It had recently

been revived and was available from the Stempel type foundry (in Germany), cast from the original matrices. However, not all the sizes needed for the *Prayerbook* were available, so Updike — in contrast to all his earlier projects, which used type from the original foundries — chose to duplicate the type, creating new matrices from original types where they existed, and having additional matrices made for the "missing" 18-point, interpolated from the existing sizes.

The Janson matrices made for the Merrymount Press are now in the possession of Firefly Press, and Kristensen characterizes himself as "the world's last D. B. Updike 'wannabe' ".

Kristensen's talk also covered other fonts used, and works issued, by the Merrymount Press, as well as Updike's relations with other printers and institutions of the period. A comparison with the works designed by Bruce Rogers recognized the sheer beauty of Rogers' title pages and text, but pointed out that they were not easy to read, whereas Updike designed books that were not only beautiful, but meant to be read and used, an absolute requirement for works such as the *Prayerbook*. (An attendee at the lecture had experience conducting services from the *Prayerbook*, and confirmed that it is indeed eminently suited for that use.)

When the Merrymount Press ceased operation, its holdings were distributed to several sites, mostly outside Massachusetts — to keep them away from Harvard. (It wasn't explained why Updike was adamant about this.) The bulk of the fonts, most matrices, and other materials were relocated to the Huntington Library in San Marino, California. The matrices for the Mountjoye (Bell) font went to the Bancroft Library at Berkeley. The matrices for the Janson font passed through several hands, ending up with Firefly Press. The matrices for two fonts (Merrymount and Montallegro) created for the Press, along with much historical material, specimen sheets, and related papers were donated to Special Collections at the Providence Public Library, where they became the foundation of the D. B. Updike Collection. Updike was a native Rhode Islander, and his legacy is now a resource for aspiring type designers, as reported in other *TUGboat* issues, as well as later in this column.

W. A. Dwiggins — Making orders

A review of the book by Bruce Kennett — *W. A. Dwiggins: A life in design* — appeared in the previous issue of *TUGboat* (`https://tug.org/TUGboat/tb39-2/tb122reviews-kennett.pdf`). On October 13, Kennett appeared at the Museum of Printing to talk about the period of Dwiggins' life during which he worked with many of the paper mills in Massachusetts, producing advertising and promotional materials aimed at printers, the principal users of the paper products.

Dwiggins' approach was not a "hard sell", but provided information that would assist printers in making best use of each type of paper. Among the techniques he recommended were the use of line cuts rather than halftones to present illustrations; the contemporary technology did not render halftones cleanly, whereas line cuts were capable of producing sharp, attractive images. Other recommendations included matching fonts and paper to the intended final product and audience. What he provided was a toolbox, not a recipe. An attractive broadside in fact unfolded to display the image of a functional carpenter's toolkit, inviting the viewer to choose the best tools for the job.

In a related comment, Dwiggins voiced the opinion that the lowercase of available sans serif fonts was simply dreadful. On hearing that opinion, Linotype asked "can you do better?" Rising to the challenge, Dwiggins produced designs that were indeed superior to anything already available; as a result, Linotype put him on retainer, accepting unseen anything that Dwiggins produced that was applicable to their line.

A quite broad selection of examples of Dwiggins' work was on display, to be handled and inspected directly. This gave a wider appreciation of the work than can be obtained from images in a book, no matter how carefully produced.

The Updike prize for student font designers

Special Collections at the Providence Public Library is the home of an extensive typography collection that grew from a legacy of Daniel Berkeley Updike's correspondence and books on the subject, and is named in his honor. Since 2014, when a prize for student type design was launched with a lecture by Matthew Carter [1], a ceremony has been held every year [2, 3, 4] to recognize the finalists and winner of the prize, accompanied by a talk by a current practitioner of font design [5]. This year's celebration was held on October 24, and the speaker was Victoria Rushton.

Rushton is an illustration graduate of the Rhode Island School of Design (RISD).

During her undergraduate studies, she discovered that, more than anything else, she liked to include words in her drawings. After graduation, she determined that font design was much more suited to her interests and undertook training at Font Bureau to design typefaces. Three of her typefaces have been

released commercially [6], and she accepts custom lettering commissions. The style of her work tends toward script styles rather than typefaces intended solely for the setting of text.

Rushton's talk was the story of her personal journey into typeface design. It concentrated on three "unreleased" typefaces, and the inspirations for their creation.

The first typeface was a swirly, romantic script, created in the aftermath of a breakup.

The second face was based on her sister Cecilie's handwriting. Cecilie is a fabric artist, and the typeface was originally used for all the text on her web pages, as shown in slides accompanying the talk. However, even though the pages still mention the font [7], the only remaining evidence online is the logotype in the top right corner.

The third face was based on a handwritten letter by Oswald ("Oz") Cooper, a type designer, lettering artist, and graphic designer active during the early 20th century. Cooper is known largely for bold display typefaces, many of which were based on his handwriting. The typeface created by Rushton accentuated various features of Cooper's script, in particular the rounded terminals, resulting in a quirky but pleasing informal appearance that worked surprisingly well for text.

After Rushton's talk the results of the student competition were presented. Four designs were chosen for recognition, although only three of the participants were in attendance; the typeface created by the fourth was not shown.

Top honors were awarded to the "Frisk" family of fonts by Gene Hua. The family consists of numerous weights in both upright and italic forms. His inspiration was the content of old playbills. While suitable for text composition, the impression left by the design is lively rather than sober. (One of the judges did characterize the italic form as "frisky".) In addition to other prizes, the award included a trophy — a composing stick (with the winner's name on a plate on its side), which was a source of consternation to all the student competitors: what *is* this thing, and what is it used for?

In second place was the "Altar" font created by Stephanie Winarto. This text face was inspired by an Episcopal altar book set in the Merrymount font designed for Updike's Merrymount press. (The book is reminiscent of productions by William Morris, with highly decorative marginal graphics.) The new face is characterized by diamond-shaped elements, with the dots on "i"s set in red in the example of its use.

A runner-up was "Updike Nouvel" by Annibel Braden. This was based on a sign-painter's guide, which shows only uppercase letters that appear to be incised rather than printed. The example of the new typeface in contrast appears to be raised, three-dimensional, and is multi-layered, shaded, and gold-colored. The "Glyphs" tool was used in its creation.

A question and answer session followed the presentations, moderated by Matthew Bird, a member of the RISD industrial design faculty. In addition to discussing sources of inspiration, topics included the importance of choosing an appropriate name for a font (it should not already be in use), and whether the participants intend to make font design a full-time career (probably not).

A "brand" new font

A font constructed entirely from logos representing well-known brands is highlighted at `https://www.engadget.com/2018/09/01/corporate-logo-font-typeface-digital-studio/`. Most of the letters look familiar, but I failed at the attempt to identify them all. Can you do better?

References

[1] Updike prize for student type design and talk by Matthew Carter, *TUGboat* 35:1 (2014), 3–4, `tug.org/TUGboat/tb35-1/tb109beet.pdf`

[2] First annual Updike Prize and talk by Tobias Frere-Jones, *TUGboat* 36:1 (2015), 4–5, `tug.org/TUGboat/tb36-1/tb112beet.pdf`

[3] Second annual Updike Prize for student type design and talk by Fiona Ross, *TUGboat* 37:3 (2016), 257–258, `tug.org/TUGboat/tb37-3/tb117beet.pdf`

[4] Type designer Nina Stössinger speaks at 3rd annual Updike Prize event. *TUGboat* 39:1 (2018), 19, `tug.org/TUGboat/tb39-1/tb121walden-updike.pdf`

[5] Announcement of the Updike Prize. `https://www.provlib.org/research-collections/historical-collections/updike-prize-student-type-design/`

[6] Three fonts by Victoria Rushton. `https://victoriarushton.typenetwork.com/`

[7] Cecilie Rushton's web page. `https://cecilierushton.com/`

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

TUGboat open-access survey results

TUG Board

Earlier this fall (approx. August 6–September 16), the TUG Board conducted a one-question survey on increasing open access to the *TUGboat* journal. To date, *TUGboat* has been delayed open-access, with most technical articles available only to members for about one year after publication.

Clearly, we would only want to change the policy if doing so would help TUG, not hurt it. Memberships are what pay the bills, and we could not predict whether this change would cause a crash in memberships or not. We knew the survey could provide only general indications, since the responders would be self-selected and unverified, but it was the only way we could think of to garner any information at all. The survey had one question for members:

> If all *TUGboat* material were publicly available online immediately upon publication, would that make renewing your TUG membership more likely, less likely, or have no particular effect?

and the analogous question for non-members, asking about joining instead of renewing.

There was an excellent response, better than we had hoped for. 588 people responded in all, 519 (88%) as current members and 69 (12%) as non-members. At the time of the survey, TUG had approximately 1180 members, so about 44% of members responded.

Overall, approximately 79% of respondents said that the proposed full open-access would have no effect on their decision, 11% said they would be less likely to join or renew, and 10% said they would be more likely. (The exact numbers are posted at `tug.org/TUGboat`.)

Given this split response, the Board has decided on a compromise change: all *TUGboat* issues *except the current one* will now be publicly available. For example, you are reading this report in *TUGboat* 39:3, so issues 39:2 and earlier are now publicly available, while the technical articles in this issue remain available only to members (either in print or online at `tug.org/members`), until issue 40:1 is published.

It is our hope that this change will both retain the current members who would have (understandably) dropped with full open-access, and lead more people to renew and join in support of the greater access. If future developments dictate, the policy may be changed again.

We also hope that the greater access will lead to more references to *TUGboat* articles on the forums and mailing lists, and greater visibility generally.

Comments

In addition to the one question above, we had a field where respondents could submit free-form comments. We were grateful to receive dozens of insightful and thought-provoking ideas this way, and would like to relate and respond to a few of them here.

The vast majority of comments, following the numeric results, were a statement of support for TEX and TUG in general, and so any particular *TUGboat* policy would not change that support. Thank you all!

Among people responding "more likely", the general sentiment was support for open access in general, and the concomitant benefits.

Several people asked about getting notifications when a new issue was published. We send out a monthly newsletter, which includes announcements of new *TUGboat* issues. Non-members can subscribe at `lists.tug.org/tex-announce`.

A few people were evidently unaware that it is possible to choose to get *TUGboat* electronically only, or how the fees changed. Indeed, there is a substantial discount on the membership fee for receiving benefits electronically only; see `tug.org/join`. That page also explains possible tax deductions.

On the other hand, quite a few respondents indicated a strong desire for the physical *TUGboat* (and software DVD), belying other respondents who wondered who would want anything on paper (or disc) nowadays. We have no plans to discontinue the physical *TUGboat* (or DVD).

A couple people suggested conducting the survey through the members' area on the web site, to avoid multiple submissions and know if responders were members. We did consider this, but ultimately felt that responders' anonymity was more important to gathering useful results.

And lastly, one eagle-eyed respondent noticed we incorrectly used past tense instead of the subjunctive in the survey text. Does any other organization have such proofreaders?! (We fixed it.) Thank you.

Thanks to all the survey respondents for giving us a foundation for a decision, and making so many thoughtful comments. And a special thanks to all the authors, reviewers, and all contributors to *TUGboat* over the years, who have made it an important resource for the TEX world since TUG's founding.

For further comments or discussion on this (or any other TUG questions), feel free to contact the Board at any time.

◇ TUG Board
`board@tug.org`

TEXConf 2018 in Japan

Norbert Preining

On Saturday 10 November I attended the yearly conference of Japanese TEX users, TEXConf 2018,[1] which this year took place in Sapporo, Hokkaido.

TEXCONF 2018

SAPPORO 2018.11.10

Having attended several international TEX conferences, I am always surprised how many Japanese TEX users find their way to this yearly meeting. This year we were about 50 participants. We had five full talks and two lightning talks, followed by a very enjoyable dinner and after-party.

The first talk was by Takuto Asakura (朝倉卓人), 'llmk — The light LATEX Make'[2] (slides[3]). Takuto gave a short overview of the available TEX make alternatives and why he saw the need for a new tool, which is written in texlua. After some short examples of usage he mentioned a few advanced usage scenarios. He will write an article for *TUGboat* and plans to present `llmk` at the TUG 2019 conference in the USA. As an old-school guy I prefer `make`, which is by far more powerful, but I welcome additions to make building TEX documents easier. My only wish would be a *no-markup-do-your-best* build system — guess I will start writing my own ;-).

Next up was my own talk on 'Continuous integration testing for TEX Live' (slides[4]), where I got into the nitty-gritty details of DevOps for TEX Live — mirroring the Subversion repos into git, and linking them to CI services, as well as using deployments to get binaries back. I hope to have an article about this ready for the next *TUGboat*.

After lunch, Takashi Suwa (諏訪 敬之) presented his work on a new typesetting system with a static type system, SATySFi.[5] With his background in formal verification, Takashi took an interesting approach to typesetting. Due to the complete static typing of the input source, error messages can be much more informative — one of Takashi's biggest complaints with current TEX, but it also makes the input format a bit bothersome in my opinion. For me one of DEK's biggest achievements is the definition of a no-thrills, easy to read and write, input format for mathematics. Takashi has also written a book documenting SATySFi, and I have urged him to make an English translation.

The next talk was by Keiichiro Ishino (石野恵一郎) on 'Breaking paragraphs into lines with the AHFormatter'[6] (slides[7]), a commercial typesetting program targeting businesses with XSL-FO, CSS, XML, ... formatting abilities. It was very interesting to see how commercial products deal with the very same problems we are facing.

Hironori Kitagawa (北川弘典) reported on the state of `luatex-ja`[8] (slides[9]), in particular his work on line adjustments in the presence of inline math formulas, as well as usage of the `luatex-fontspec` sub-package. I cannot repeat it often enough — I consider `luatex-ja` one of the most important packages and it is in daily use on my site.

The day closed with two lightning talks, the first by Keiichiro Shikano (鹿野桂一郎) on the usage of Encapsulated PostScript (`eps`) files in TEX (slides[10]). Unfortunately, he didn't really rehearse his talk and his time was over before it got interesting ;-). Fortunately, we can read his slides online.

The last talk was by Hironobu Yamashita (山下弘展) on 'How to become happy when typesetting Japanese with LATEX' (slides[11]). A very funny and informative talk on the incredible work Hironobu is doing for the TEX community — development of *many* packages, support, and updates of the source code of several programs; the list is long.

I love to attend TEX meetings, and the Japanese TEXConf is in particular always interesting, in particular because TEXies here have a tendency to be rather tech-savvy, one could even say `\expandafter`-manic. This was in fact the biggest complaints during our walk to the dinner location — not enough mention of `\expandafter` in the talks.

It is now nearly ten years that I've been attending the Japanese TEX user meetings, and I think we have come a long way — from a rather separate group of TEX developers and users distributing their stuff on Japanese-only wikis and private pages to a group that is now strongly integrated into our global TEX community (let me just mention the TUG conference in Tokyo[12]) as well as contributing to many projects. Thanks a lot!

◇ Norbert Preining
Accelia Inc., Tokyo, Japan
`preining.info`

[1] `tug.org/l/4A0k8`
[2] `tug.org/l/9JuCN`
[3] `tug.org/l/72koH`
[4] `tug.org/l/5g6hV`
[5] `tug.org/l/y7Nd1`
[6] `tug.org/l/5WRrP`
[7] `tug.org/l/7fLd4`
[8] `tug.org/l/T7tVg`
[9] `tug.org/l/PbYI5`
[10] `tug.org/l/OM4yt`
[11] `tug.org/l/3DIur`
[12] `tug.org/l/Dq5GI`

The Cary Graphic Arts Collection

David Walden

The Cary Graphics Arts Collection (`library.rit.edu/cary`) on the second floor of the Wallace Library at the Rochester Institute of Technology (RIT) is a library on the history and practice of printing. Contained within the Collection is the Cary Graphic Design Archive,[1] "preserving the work of significant American graphic artists from the 1920s to the present". The library has many important holdings and is open to visitors as well as faculty and students at RIT.

Unusual for a library and archive, the Cary Collection includes a print shop with "a working collection of some 20 historical printing presses and more than 2,000 fonts of metal and wood type"[2] of mechanical design spanning from the early 1800s to the present.

The collection curator Dr. Steven Galbraith (in photo below) says that last year 2,000 students came through or used either the library or print shop for classes and projects.

On June 25, 2018, Curator Galbraith gave me a tour of the Collection's facilities, along with Kris Holmes whom I was interviewing (see interview in this issue), in the Collection's Reading Room.

According to the 85-page book *Highlights of the Cary Graphic Arts Collection*, the collection was initially established at RIT in 1969 with the gift, in honor of Melbert Cary Jr., of his library, by the Mary Flagler Cary Charitable Trust (created in memory of Melbert Cary's widow). Melbert Cary's life involved printing in several ways: he was director of an agency that imported metal type from Europe that it sold to printers, he had his own small printing business, and his interest in the history of printing led to a library of 2,300 books on the topic. In the years since the creation of the Collection, the library has grown through other gifts and acquisitions and now contains some 45,000 volumes. The Collection also includes a number of subcollections, including typography specimens, examples of fine printing, books on book binding, and many more (`twcarchivesspace.rit.edu`).

Curator Galbraith is anticipating a major renovation of the RIT Libraries, and he says, "Our goal in the Cary Collection is to expand our facilities to make our collections more visible and accessible to students and researchers."

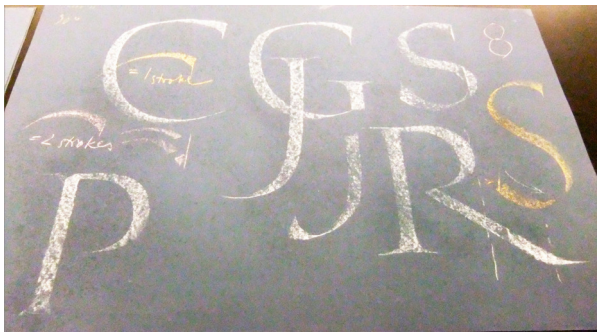
The Collection also regularly presents lectures and exhibits,[3] for instance a 2012 exhibit on the edges of books (photo by Elizabeth Lamark).[4]

In early 2018, Chuck Bigelow and Kris Holmes gave a lecture at the Collection celebrating the 40th anniversary of their first commercial font, Leviathan, a display capitals type for a fine press printing of *Moby Dick*.

Chuck and Kris have long been involved with RIT's graphic arts programs, with Chuck until recently serving as Melbert Cary Jr. Distinguished Professor of Graphic Arts at RIT and Kris teaching type design. Chuck is now resident scholar at the Cary Collection.

Hermann Zapf was an earlier Melbert Cary Jr. Distinguished Professor of Graphic Arts (from 1977).[5] From 1979 through 1988, Zapf taught a summer workshop on type design and calligraphy at RIT. More generally, Zapf significantly influenced RIT's programs related to printing, was an ambassador for the Cary Collection, helped the Collection obtain significant archives, placed many of his own papers and works in the Collection archive,[6] and designed etchings for the glass-surrounded RIT Press space in the Wallace Library (please see `rit.edu/press/history`). At the time of my visit, the Collection staff was preparing for an exhibit on Zapf; it opened in late August — The Zapf Centenary: The Work of Hermann & Gudrun Zapf, 1918–2018, commemorating the 100th birthdays of Hermann Zapf and Gudrun Zapf von Hesse with a retrospective of their influential work and careers. The following drawing,

from Zapf's 1979 RIT summer course, is typical of what he used when demonstrating calligraphy.



Kris Holmes explained, "Many teachers use a chalkboard and they write the letters with a piece of chalk held parallel to the surface, thus creating a 'broad edge'. But Zapf chose to put up sheets of dark paper and write with his chalk directly on the paper. These sheets are usefully less ephemeral than chalkboard samples. They now provide a little window into Zapf's teaching."

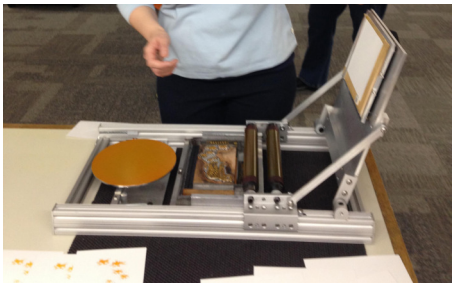
I particularly enjoyed being shown the Arthur M. Lowenthal Memorial Pressroom with its cases of type, various styles of printing presses, and other equipment for trimming, binding, and so on.

The image in the left column of the prior page shows an Albion iron hand press No. 6551 (1891) once owned by William Morris in England and Frederic W. Goudy in New York.[2] An RIT alumnus provided funds to the Collection to buy this press at a Christie's auction. Associate Curator Amelia Hugill-Fontanel was in charge of restoring the press to working order once it was acquired.[7] Of the press room and collection more generally, Hugill-Fontanel says, "Each of the Cary presses demonstrates technological progress in the development of printing machines. We firmly believe that they will be preserved through teaching and limited-edition press work."

According to another story from Curator Galbraith, a fire at Goudy's print shop destroyed much of Goudy's work. However, Goudy had loaned a different Albion press and some cases of type which thus survived the fire; that press is also now in the pressroom along with the type which is known as the "The Lost Goudy Types".

Kris Holmes took the following photo as we walked through the Lowenthal Pressroom. Of the photo she says, "This is the 'Adopt A Font' project. Students commit to cleaning up one of the Collection's wood type fonts. In this case it is one of their beautiful collection of Hebrew wood type. I think this is so typical of the Cary — they find ways of getting students engaged."

Having engineering students on the same campus as the pressroom provides especially interesting opportunities. For instance, in 2015 five students developed an "aluminum hand-operated letterpress, weighing in at 25 pounds and assembled with two Allen wrenches, [that] can produce high quality and repeatable prints"[8] (photo by Amelia Hugill-Fontanel).



In 2016 another set of students built an 18th Century English common press that was then added to the Cary Collection (see videos[9,10]).

Visiting the pressroom and learning how it and the rest of the Collection is used in classes throughout RIT makes me wish I were back in college again. Visit the collection if you are near Rochester.

Notes

[1] library.rit.edu/gda/historical/about
[2] tinyurl.com/morris-press
[3] library.rit.edu/cary/exhibitions
[4] Steven K. Galbraith, *Edges of Books: Specimens of Edge Decoration from RIT Cary Graphic Arts Collection*, RIT Graphics Art Press, 2012.
[5] Others who have held the Cary Professor position are Alexander Lawson (the first Cary professor, library.rit.edu/cary/goudylawson) and Frank Romano (between Zapf and Bigelow, www.rit.edu/news/story.php?id=49409).
[6] library.rit.edu/cary/news/hermann-zapf-tribute
[7] tinyurl.com/amelia-press
[8] rit.edu/news/story.php?id=52390
[9] youtube.com/watch?v=gVbGoQVDYjk
[10] youtube.com/watch?v=JJvFYa3EBn0

Typographers' Inn

Peter Flynn

Font tables

Peter Wilson has rightly called me to account for missing out the fonttable (two t's) package in the description of my experimental fontable (one t) package [4, p 17].

The fonttable package is much more powerful than the one I am [still] working on, and I was so intent on reimplementing the specific requirements of the `allfnt8.tex` file in XeLaTeX to the exclusion of pretty much everything else that I didn't do any justice to fonttable (and a number of other test and display tools).

I am expecting shortly to have more time at my disposal to remedy this and other neglected projects.

Monospace that fits

One of the recurrent problems in documentation is finding a suitable monospace font for program listings or other examples of code. I have whittled my own requirements down to something like this:

1. must remain distinct and legible at small sizes (eg `\footnotesize`) because *a*) you might want code in footnotes; or *b*) you might need a small size on special occasions when you need to fit a listing to the page and keep whole lines on single lines (no wrap).;
2. must *not* have a set wider than one alphabet of the body face lowercase, and preferably narrower; that is,
   `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
   must be the same width or narrower than
   abcdefghijklmnopqrstuvwxyz;
3. must distinguish clearly between zero (`0`) and capital `O`, and between lowercase `l`, digit `1`, capital `I`, and vertical bar (`|`);
4. may be serif or sans-serif. . . ;
5. . . . but must be distinctly different from the chosen body face;
6. must *not* have any characters with an unusual or unconventional design;
7. must be close but not the same in weight to the chosen body face, so that it is distinct in running text but not disruptive, and so that blocks of code have roughly the same greyness as normal text.

'What's wrong with Computer Modern Typewriter?' I hear some people ask. Actually not much, except for the idiosyncratic `@` sign[1] and the set, which is wider than its Roman sibling (see item 2 in the list on p. 171) but only very little. The problem with width is that in listings, it would be nice to have a narrower font so that fewer long lines need wrapping, while remaining readable. Of those listed in Table 1, the three space-savers are thus CMTT, Nimbus Mono Narrow and possibly Luxi Mono scaled down.

Of course, it's possible to scale any font if needed, and it's common if you're using XeLaTeX to use the fontspec package's *Scale=MatchLowercase* option on font specification commands. But while this is important for using the monospace font in running text, listings may need to be in a different size.

Among the various weights, it's clear that Courier is not only wider than most but significantly lighter. Luxi Mono is much closer to the weight of the old Prestige Elite (12–pitch) typewriter face. For the conflicting character forms, Nimbus Mono Narrow has less distinction than most of the others.

It's also a matter of æsthetics, and many people are happiest using what they think looks nice. Personally, I like PT Mono but it's very wide; Libertine Mono reminds me of a golfball I had for the IBM Selectric typewriter once: all the attributes of a serif typeface except the proportional widths; Plex Mono is in some ways very similar in feel to PT Mono

Table 1: Widths of set for some related serif, sans-serif, and monospace fonts

CMR	abcdefghijlkmnopqrstuvwxyz	O0|I1l
CMSS	abcdefghijlkmnopqrstuvwxyz	O0|I1l
CMTT	abcdefghijlkmnopqrstuvwxyz	O0|I1l
PT Serif	abcdefghijlkmnopqrstuvwxyz	O0|I1l
PT Sans	abcdefghijlkmnopqrstuvwxyz	O0|I1l
PT Mono	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Libertine	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Biolinum	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Lib. Mono	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Plex Serif	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Plex Sans	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Plex Mono	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Nimbus Serif	abcdefghijlkmnopqrstuvwxyz	O0|I1l
do. Sans	abcdefghijlkmnopqrstuvwxyz	O0|I1l
do. Mono	abcdefghijlkmnopqrstuvwxyz	O0|I1l
do. Mono N	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Times	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Helvetica	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Courier	abcdefghijlkmnopqrstuvwxyz	O0|I1l
Luxi Mono *	abcdefghijlkmnopqrstuvwxyz	O0|I1l

Times, Helvetica, and Courier (unrelated) are included for comparison as they are a common default.

* Luxi Mono set to one `\magstep` smaller than the others.

[1] When I was having the conference mugs printed for the first TUG meeting in Cork in 1990, the company replaced the `@` sign in the email address with the copyright symbol ©, which they thought 'looked nicer'. Fortunately I spotted this before they were printed.

despite being unseriffed; Courier I have a personal and unexplained dislike for; so it's down to Luxi and Nimbus from this lot — but there are so many others available too ...

Centering (reprise)

Talking of 'projects', as we shall see, the habit of allowing a wordprocessor centering algorithm to auto-center display material, which I have mentioned before [1, 2, 3] has been popping up all over the place, two of them in my own institution within a month of each other — Figure 1 shows them with the text in question reproduced underneath, as it's probably not clear from the images reproduced at this size.

Free Entry with convention passes Tickets €10 without

Free Entry with convention passes Tickets €10 without

Come find your perfect project for this summer with the help of our members!

Come find your perfect project for this summer with the help of our members!

Figure 1: Centered text with linebreaks: transcription plus suggested breaks

LaTeX, like other systems, fits the maximum number of words to the centered line[s], and allows the last line to be short, if necessary. Authors and designers should add manual line-breaks at the logical break-points, especially if there is no proof-reader. In the right-hand example here about a 'Porject [*sic*] Matching Workshop', the choice of yellow text on a brightly-coloured background (not visible in monochrome here) manages to make it so hard to read that no-one noticed.

Afterthought

A year or so ago, the Prime Minister of Pakistan came under suspicion of having forged a document it was hoped would clear his family of wrongdoing in a property transaction [6]. The document was dated 2006, but it was set in Calibri, which wasn't publicly released until 2007, although it was available in test versions of Microsoft Office from 2004, including the new XML version we were all given after the XML conference in Washington, DC, that year. In theory it's possible the Prime Minister's office was represented there...

It's by no means the first time a typeface has caused high-level embarrassment: in 2010 a designer working for a French government agency created a logo using a font called Bienvenue. Unfortunately this font was created privately for France Telecom, and isn't supposed to be available to anyone else [5]. The embarrassment was that the office concerned was France's new intellectual property rights agency HADOPI (*Haute Autorité pour la Diffusion des Œuvres et la Protection des droits d'auteur sur Internet*, no less), set up explicitly to ensure copyright and IP enforcement.

Designers and typesetters are often raided and sued over allegations of using typefaces they haven't paid for, and have been fined some very large sums as a result. I don't think any TeX user would ever do such a thing as use an unlicensed font — especially as there are so many excellent typefaces free of commercial license conditions, but please make sure you're legal. Designing a typeface takes years, and the designers deserve your support.

References

[1] P. Flynn. Typographers' Inn — Titling and centering. *TUGboat* 33(1), May 2012. tug.org/TUGboat/tb33-1/tb103inn.pdf

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

[3] P. Flynn. Typographers' Inn — Afterthought. *TUGboat* 38(1), May 2017. tug.org/TUGboat/tb38-1/tb118inn.pdf

[4] P. Flynn. Typographers' Inn — Fonts and faces and families. *TUGboat* 39(1), Jun 2018. tug.org/TUGboat/tb39-1/tb121inn.pdf

[5] Insider Software. Staying Legal: The Challenges of Font License Compliance. insidersoftware.com/downloads/infusionsoft/StayingLegal-FontLicenseCompliance.pdf, Sep 2010.

[6] B. Kentish. Pakistan's Prime Minister may be brought down by Microsoft's Calibri font amid corruption allegations. *The Independent*, Jul 2017.

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

A beginner's guide to file encoding and TEXShop

Herbert Schulz and Richard Koch

Abstract

A common problem TEX users face when opening and typesetting files is that the text displayed either in the source or in the typeset document or both is not what should be there; characters are scrambled and improper characters appear. This is usually an *encoding* problem — either the editor or TEX or both do not interpret the input correctly.

This document is meant as a first introduction to file encodings. It is definitely *not* meant as an exhaustive document, and deals only with the most common encodings in use today.

While the following document was originally written for distribution with the TEXShop editor (a.k.a. front end) running on Mac systems, other front ends use a similar directive, and the general discussion about file encodings is valid no matter what editor you use.

1 What is a file encoding?

While we usually think of the .tex source file as containing characters, in reality this source, like all computer files, is just a long stream of whole numbers, each (nowadays) from 0 through 255. Computer scientists call these whole numbers *bytes*.

All other computer data must be encoded in one way or another into bytes. The most common encoding of ordinary text into bytes is called ASCII; it encodes most of the characters found on an ordinary American typewriter. For instance, the characters 'A' through 'Z' are encoded as 65 through 90, the characters 'a' through 'z' become 98 through 123. The space character is encoded as byte 32, and numerals, parentheses, and punctuation characters encode as other bytes.

Originally, TEX required ASCII input. While this was sufficient in the United States, it proved cumbersome in Western Europe, where accents, umlauts, upside down question marks, and the like are common; macros were needed to construct those characters and that broke hyphenation. More difficult problems arose when TEX was used in the Near and Far East.

The ASCII encoding only uses bytes from 0 through 127. Thus the door was open to encode other characters using bytes 128 through 255. Many different single-byte encodings now exist to display additional characters using these bytes.

2 Extending the character table

The three most often used extended single-byte encodings on the Mac are MacOSRoman, IsoLatin1 and IsoLatin9.[1]

The MacOSRoman encoding is left over from the days before OS X and, as expected, exclusive to Mac computers. Its use is no longer encouraged.

The IsoLatin1 encoding extends the ASCII encoding with the accented characters used in Western European languages.

IsoLatin9 primarily adds the Euro symbol, €, to the IsoLatin1 encoding along with a few other changes.

2.1 Other encodings used with TEX

Other fairly common encodings include IsoLatin2 for central European languages, IsoLatin5 for Turkish and IsoLatinGreek (also called Iso8859-7) for Greek. Several different encodings are available for Russian and other languages using Cyrillic. Additional encodings are available for Korean and Chinese, but Far Eastern languages use thousands of symbols, so these encodings are not very satisfactory.

2.2 Windows stuff

Windows Latin 1 is a version of IsoLatin1 with some characters in different code locations as defined by Microsoft. Thus, folks running Windows can end up with files in this encoding.

2.3 A crucial flaw

The various encodings were developed independently by computer companies as their products were sold in more and more countries.

Unfortunately (but unavoidably), text files do not have a header specifying the encoding used by the file. Thus there is no way for TEXShop to automatically adjust the encoding as various files are input. Some text editors have built-in heuristics to try to guess the correct encoding, but TEXShop does not use these heuristics because they work only 90% of the time and an incorrect guess can lead to havoc.

3 Unicode

As the computer market expanded across the world, computer companies came to their senses and created a consortium to develop an all-encompassing standard, called Unicode. The goal of Unicode is to encode all symbols commonly used across the world, including Roman, Greek, Cyrillic, Arabic, Hebrew, Chinese, Japanese, Korean, and many others. Unicode even has support for Egyptian hieroglyphics and

[1] We will use the same notation as for the TEXShop encoding directive in this document. See the table on page 176.

relatively recently added support for mathematical symbols.

All modern computer systems, including Macintosh, Windows, GNU/Linux and other Unix, now support Unicode. Internally, TEXShop and many other editors represent characters using Unicode and thus can accept text that is a combination of Roman, Greek, Cyrillic, Arabic, Chinese, and other languages. TEXShop even understands that Arabic, Hebrew, and Persian are written from right to left. To input these extra languages, activate additional keyboards using the System Preferences Keyboard Pane. This Pane changed in recent versions of OS X; in El Capitan, select a keyboard on the left, or click '+' below the list to see a list of additional languages and add their keyboards.

3.1 Unicode representations

Because it has far more than 256 symbols, Unicode defines symbols using much larger integers, using more than one byte. Unicode defines the "internal" structure of these numbers, but gives several different ways to represent the numbers on computers. By far the most popular Unicode encoding nowadays is UTF-8, which uses a sequence of 8-bit bytes, but UTF-16 (using a sequence of 16-bit chunks) and others are also available. (All the Unicode representations are equivalent; the multiple representations exist for historical and other reasons beyond this short note.)

The great advantage of UTF-8 is that ordinary ASCII characters retain their single-byte form in the encoded file. Consequently, ordinary ASCII files remain valid as UTF-8 files. With most single-byte encodings like IsoLatin1, IsoLatin9, etc., any sequence of bytes forms a legal file. If you open such a file with the wrong encoding, the file will be read, but some of the symbols will be wrong. For example, if someone in Germany using IsoLatin9 collaborates with someone in the U.S. using MacOSRoman, and their paper is written in English, they may not notice the mismatch until they proofread the references and discover that accents and umlauts have gone missing.

However, not all sequences of bytes form valid UTF-8 files, because non-ASCII symbols are converted into bytes using a somewhat complicated code. In the previous example, if the German collaborator uses IsoLatin9 and includes non-ASCII characters, such as those with umlauts, in the document and the American collaborator uses UTF-8, TEXShop will report an error when it tries to open the IsoLatin9 file in UTF-8. TEXShop will then display an error message and offer to open the file in a "fallback" single-byte encoding, currently IsoLatin9 (not configurable).

On the other hand, both authors of this document use UTF-8 Unicode as our default encoding, turning that message to our advantage. UTF-8 preserves everything typed in TEXShop, so there are no puzzling character losses. HTML and other code is usually saved in UTF-8, so TEXShop can be used as a more general text editor. Moreover, if a TEX file from an external source is not in UTF-8, we get the warning above. The trick is then to let TEXShop open the file in the "fallback" encoding, IsoLatin9, and examine the file for an inputenc line which tells you what encoding was actually used. Then close the file *without making any changes* and re-open it using the Open dialog and manually choose the correct encoding. Once the file is open with the correct encoding you may add the TEXShop encoding directive line for that encoding and save it for future use.

Using UTF-8 Unicode has become so advantageous that TEXShop 4.00 and later use this encoding as the default, out of the box,[2] encoding.

3.2 Encoding vs. formatting

All of the encoding methods discussed here, including Unicode, are irrelevant to italics, underlining, font size, font color, etc. They just define characters as numbers. It is up to users to specify additional attributes in some other way. For example, when Apple's TextEdit program is used in *Plain Text* mode, a user can change the font or font size for an entire document, but not for individual sections of the document. If the document is saved to disk and then reloaded, the font changes are lost. On the other hand, a word processor like Microsoft Word or Apple Pages has much more control over fonts, font size and the like. These programs output text in a proprietary format readable only by that program, but the file does preserve the extra attribute information.

While all modern computers support Unicode, particular fonts (nearly always) have symbols for only a small portion of the Unicode world. Fonts should have a special character, often a box, to indicate that a character is missing. Thus if you want to write in, say, Arabic or Hebrew, you must choose a font which contains these symbols. Modern computers support a great range of symbols because the computer business covers the world, but it may still be hard to find a font covering obscure Unicode symbols.

[2] If you switch to the latest TEXShop version and have already reset the default encoding in TeXShop → Preferences, your selection will be maintained.

4 Two sides of the story: TEXShop and TEX

Once a user selects an appropriate encoding, the user must configure both TEXShop and the appropriate TEX engine to use that encoding. Different sets of problems arise with these two tasks.

Users in the United States and other English speaking countries can often ignore encodings altogether. The default TEXShop encoding supports ASCII, and TEX and LATEX have supported ASCII from the beginning. So there is nothing to do.

Users in Western Europe must take slightly more care. The current default TEXShop encoding, UTF-8 Unicode, will be sufficient for their needs. But they must configure TEX and LATEX as described below, and carefully choose fonts which support the needed accents, umlauts, and the like. The required steps are easy.

Users in Russia and Eastern Europe must take similar steps, but the authors of this paper are not knowledgeable about correct configurations, so we suggest getting help from friends already using TEX.

Users in the Far East and Middle East, and scholars working with multi-language projects, will need to consult other sources for detailed configurations. These users should certainly examine XETEX and LuaTEX, because these extensions of TEX use Unicode directly and are much more capable of handling languages where Unicode becomes essential. Both XETEX and LuaTEX can typeset almost all standard TEX and LATEX source files, but have additional code for Unicode support. One big problem with these languages is that appropriate fonts must be chosen which support the languages. To simplify that problem, both XETEX and LuaTEX allow users to use the ordinary system fonts supplied with their computer.

5 Telling TEXShop what encoding to use to Load and Save source files

To set the default TEXShop encoding, open TEXShop Preferences. Select the Source tab. In the second column, find the Encoding section. This section contains a pull down menu; select the desired encoding from this menu. Select Western (ISO Latin 9) to get the IsoLatin9 encoding, useful in English speaking countries and Western Europe. You must select Unicode (UTF-8), the current default, or Unicode (UTF-16) if you want to preserve everything you can type into the TEXShop editor. If you pick any other encoding, there may be characters you can type in TEXShop which will be lost if you Save and then re-Load. On the other hand, UTF-8 may not work well with certain LATEX packages, as explained later.

TEXShop has a mechanism to set the encoding of a particular file independent of the user's default choice, or of choices in the Load and Save panels. To set the encoding used to read or write a particular file to UTF-8, add the following line to the first twenty lines of the top of the file:

```
% !TEX encoding = UTF-8 Unicode
```

The easy way to do this is to select the Macro command Encoding. A dialog will appear from which the desired encoding can be selected, and after the dialog is closed, the line will be placed at the top of the file, replacing any existing encoding line.

If such a line exists, the indicated encoding will be used, overriding all other methods of setting the encoding, *unless* the option key is held down during the entire load or save operation.

Many users in Western Europe prefer to set IsoLatin9 as their default encoding so they can easily read files from collaborators, but include the line setting encoding to UTF-8 in file templates used to create files, so that their own files are encoded in UTF-8.

It is also possible to set the encoding used to read a file by Opening the file explicitly from within TEXShop. The resulting dialog has a pull-down menu at the bottom to select the encoding to be used for that particular file.[3] (Note that the "`% !TEX encoding =`" line overrides this command.)

Explicitly Saving a file from within TEXShop produces a Save Dialog with a similar pulldown menu to set the encoding.

Note: you *can't* easily change the encoding of a file. The best thing to do is copy the whole document into a new one and save that with the correct encoding. Using the TEXShop directive before saving the new file the first time is definitely recommended.

6 Telling LATEX about file encodings

Your typesetting engine needs to know the encoding used to save each source file so the input source and the output glyphs are synchronized. For ordinary LATEX, this is usually done by including a command like the following in the header of the source:

```
\usepackage[latin9]{inputenc}
```

Some values for other common encodings are given in the short table following.

This line is not needed when the source encoding is ordinary ASCII.

One valid value for encoding with `inputenc` is utf8. This line works in Western Europe, but not in situations requiring wider use of Unicode (because

[3] Under El Capitan you must first press the Options button to get to the pulldown menu.

TEXShop Open/Save dialogs	TEXShop encoding directive	LATEX `inputenc`
Unicode (UTF-8)	`UTF-8 Unicode`	`utf8`
Western (Mac OS Roman)	`MacOSRoman`	`applemac`
Western (ISO Latin 1)	`IsoLatin`	`latin1`
Central European (ISO Latin 2)	`IsoLatin2`	`latin2`
Turkish (ISO Latin 5)	`IsoLatin5`	`latin5`
Western (ISO Latin 9)	`IsoLatin9`	`latin9`
Mac Central European Roman	`Mac Central European Roman`	`macee`
Western (Windows Latin 1)	`Windows Latin 1`	`ansinew` or `cp1252`

Table 1: Partial encoding list of names in three contexts

the characters are lacking from TEX's usual fonts). When in doubt, it is useful to read the inputenc documentation. To do that, go to the TEXShop Help menu, select Show Help for Package, and fill in the requested Package with inputenc.

Users in Western Europe usually use *four* "related" commands in the header. Here are these four lines for users in Germany.

```
\usepackage[german]{babel}
\usepackage{lmodern}
\usepackage[T1]{fontenc}
\usepackage[latin9]{inputenc}
```

The first of these lines asks LATEX to use German conventions for dates, hyphenation, etc.

The second line tells LATEX to use the Latin Modern fonts. These fonts agree with Donald Knuth's Computer Modern fonts in the first 128 spots, but include additional accents, umlauts, upside down question marks, and so forth used in Western Europe.

The third line tells LATEX the connection between the input characters in the file and the glyphs in the fonts (i.e., the physical representation of the printed characters in the final document).

As explained above, the final line tells LATEX which encoding was used for the source file.

Users interested in more details should consult the documentation for babel, lmodern, and fontenc using TEXShop's Show Help for Package item in the Help Menu. The documentation is interesting, going into considerable historical detail about the evolution of font design in TEX.

7 Encodings understood by TEXShop

Table 1 shows the corresponding entries for some popular file/input encodings used with LATEX in TEXShop.

The 'Open/Save Dialogs' column shows the designation for the encodings in TEXShop's Open/Save Dialogs; you may have to click on the Options button to display the popup menu for encodings.

The 'Directive' column gives the designation used in TEXShop's encoding directive,

```
% !TEX encoding = xxxxx
```

where `xxxxx` is the designator you wish to use. If this line is in place before you first Save your source file, TEXShop will automatically save the file with the designated encoding. TEXShop will also automatically Open the file with that encoding when double clicked. We suggest you create a Template which contains the directive and use that to create new documents.

The '`inputenc`' column gives the optional argument for the LATEX `inputenc` package. As with the Directive, we suggest creating a Template which has the proper `inputenc` line for the corresponding encoding in the directive.

Good luck!

◇ Herbert Schulz and Richard Koch
tug.org/mactex

The DuckBoat — News from TEX.SE: Formatting posts

Herr Professor Paulinho van Duck

Abstract

Prof. van Duck carried on a survey regarding the non-canonical reasons to upvote TEX.SE posts; in the first part of this installment, he will show you the most meaningful results of his research. In the following Quack Guide, you will find some tips & tricks for quickly formatting TEX.SE posts and attaching images to them.

1 Quack chat

Hi, TEX/LATEX friends!

Many things have happened since last time. The most important one is undoubtedly that Paulo Cereda finished his thesis, now he is a PhDuck! It should have been a secret, but how could Paulo keep *that* secret?

All the TEX.SE friends were very happy with it, but soon we missed a reason to make fun of him. So we started reminding him to publish the new version of `arara`, but he also did that: `arara 4.0` is now available, with the coolest manual ever!

So, if you find a new topic to scoff at Paulo, please let me know.

In July, the most highly-anticipated event of the year, the TEX Users Group meeting, took place at the gorgeous location of Rio de Janeiro. I am very glad that the presentation of `tikzducks` was declared the best talk, quack!

However, the occurrence that actually turned the TEX.SE Community upside down was the new site theme, gone live in August. Unfortunately, it is rather awful and less convenient, compared with the previous layout, and few people like it. The Powers (the people who manage all the Stack Overflow site) did not take into account any suggestions made previously by the TEX.SE Community, and it made many people angry. The post which announced the change got (at the time of writing) 69 downvotes — a record — and only 12 upvotes. If you want to know the whole story, just visit the TEX.SE Meta site.

Before going on, let me thank Claudio Beccari, from GUIT, the Italian TEX user group. He appreciated my first article very much and asked me to make an Italian version of it.

He also found an error in my first Quack Guide: the file extension must be specified in the BibLATEX macro `\addbibresource`. In my MWEB example on page 305 in *TUGboat* 38:3, it should have been:

```
\addbibresource{jobname.bib}
```

(extension `.bib` included).

Finally, I would like to thank Ulrike Fischer and her husband Gert, who allowed me to meet Bär, and spend a nice day together with them.

2 Upvoting behavior

Some time ago I suggested to my friend Carla that she post a question/poll on the TEX.SE Meta site about the "wrong" reasons to upvote.

It was welcomed by the Community, many answers arrived, and some results even surprised me, quack!

I will comment on only a little of the feedback here; you can find the complete list at `tex.meta.stackexchange.com/questions/7627/poll-wrong-reasons-to-upvote`.

Of course, all the listed reasons are not *per se* always "wrong", but they are if you upvote *only* because of them.

According to the help pages of the site, voting up is how the community indicates which questions and answers are most useful and appropriate. In particular, the best answers should receive more votes so that good content rises to the top.

I would have expected that the first "wrong" reason to upvote would be "the post contains a stunning image or a beautiful typographical object" and users vote for it *only* because they love the picture, in the same way they put a "like" on a kitten photo in a social network. Many Ti*k*Z posts, for example, get votes due to this reason.

But, contrary to my expectations, it is only third in rank; the gold medal goes to "the post is by one of the top users, I upvoted on trust."

Of course, if the answer is by a top user, it is likely to be excellent. However, it is "wrong" to upvote *only* because it is by a top user, without even reading it, and without reading the other alternative answers, which could be even more refined.

Also, sometimes top users can give a bit overly complex answers to show their skills; there are cases where the same result can be obtained with simpler methods.

There are even (rare) situations when a top user does not understand the question, so the answer is gorgeous, but it does not solve the OP's problem.

So, please remember to upvote the answer, not the answerer, quack!

Going back to our rank, the second place goes to "the answer is the first of the list, it solves my problem, I have no time/will to read the others, even if they could be better."

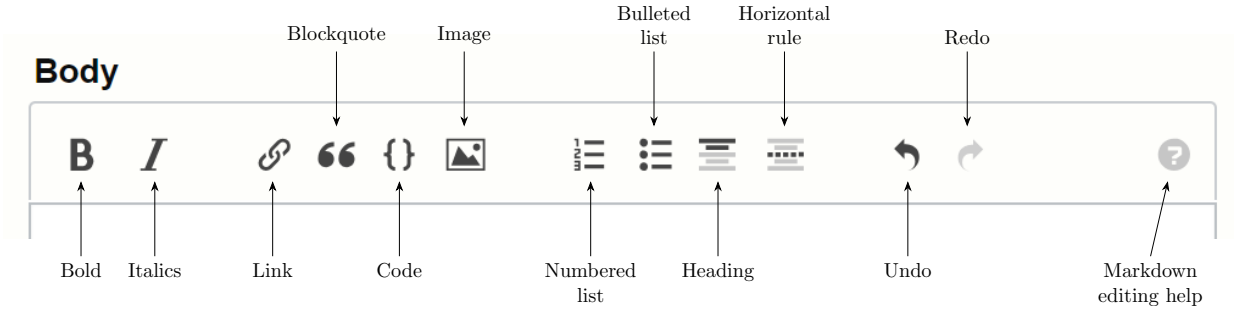


Figure 1: Formatting toolbar buttons

It is also likely that the first answer gets more votes than others simply because every time an answer is added the post goes on the Top Questions list. Indeed, "the post is in the top charts" is "wrong" reason no. 7.

I would add that timing matters on many occasions, for example, an answer posted when it is evening in Europe surely earns more reputation than one posted when it is afternoon in Honolulu (sorry for Hawaiian (LA)TEX friends, maybe it is a punishment because they cook pineapple pizza).

Reason no. 4 is one which I find it difficult to call "wrong": "the post has some duck-related content." I know it could be unbelievable, but there are users who hate ducks, there are also users who eat ducks, I have to pay attention, quack!

Some time ago the duck mania reached a peak, and every time you entered our chat you saw a Ti*k*Z ducks picture; hence some users got bored of it.

Now the situation is quieter, so feel free to use the new package `duckuments`[1] (by Jonathan P. Spratte, alias Skillmon, one of our best users) or the new `example-image-duck` when you build your MWEs.

Recently also marmots (after the nickname of our speediest Ti*k*Z expert) have caught on, and they have their own package `tikzmarmots`.[2] We are also waiting for `tikzlings`, the new Ti*k*Z collection of animals and beings.

Anyway, remember that upvoting is important because it is the way to show best answers to future users and to thank people who posted them, quack!

3 Quack Guide No. 3 How to format a post in TEX.SE

This time my Quack Guide does not strictly concern LATEX, but TEX.SE.

I will try to explain how to format a post on our site and attach an image to it. A very special reader of mine (Peter Wilson) asked me to treat this topic, and how can I not please my readers?

3.1 Formatting the text of the post

TEX.SE uses Markdown, a very simple markup language. Table 1 shows the formatting basics.

The more common "commands" also have buttons in the formatting toolbar placed above the body frame of your post, see Figure 1, and some shortcuts, listed in Table 2.

To apply them, just select the text you want to italicize/bold/etc. and press the specific button or its keyboard shortcut. Please note that the button and the shortcut for formatting code work both for inline code and code blocks.

Unfortunately, TEX.SE does not allow you to attach a `.tex` file. The quickest way to add your MWE is to copy it from your editor, paste it in the body frame of your post, select it and press Ctrl+K or click the curly brackets button.

On the other hand, adding links is allowed. If the link refers to another question from TEX.SE, you can simply copy-paste it, and its clickable title will automatically appear.

For the other links, to make them more readable than the simple site address, you can put a description within square brackets and the url within parentheses:

```
[<link description>](<url>)
```

The description will appear in red, as a clickable link.

The same result can be obtained with reference-style links:

```
[<link description>][<urltag>]
```

putting the tag resolution after an empty line:

```
[<urltag>]: <url>
```

The tag can be a number or a word.

[1] `ctan.org/pkg/duckuments`

[2] `ctan.org/pkg/tikzmarmots`

Table 1: How to format the text of a post on TEX.SE

Markdown syntax	Result
`Use _one_, __two__ or ___three___ underscores or asterisks to get *italics*, **bold** or ***bold italics***.`	Use *one*, **two** or ***three*** underscores or asterisks to get *italics*, **bold**, or ***bold italics***.

`For `inline code` use backticks.`
`Indent four spaces (after an empty line) for:`
 	
`␣␣␣␣Code`
`␣␣␣␣blocks.`
	For `inline code` use backticks.
	Indent four spaces (after an empty line) for:
	
	`Code`
	`blocks.`

`Add two spaces␣␣`
`at the end of a line to have a linebreak.`
 	
`Leave an empty line for a new paragraph.`
	add two spaces
	at the end of a line to have a linebreak.
	
	Leave an empty line for a new paragraph.

`For bulleted lists, leave an empty line and:`
 	
`- use a minus sign`
`+ or plus sign`
`* or an asterisk.`
 	
`For a numbered list, leave an empty line and type:`
 	
`␣1. a space`
`␣2. a number`
`␣3. and a dot.`
	For bulleted lists, leave an empty line and:
	• use a minus sign
	• or plus sign
	• or an asterisk.
	
	For a numbered list, leave an empty line and type:
	1. a space
	2. a number
	3. and a dot.

`Add a > to the beginning of any line`
`> to create`
`> a blockquote.`
	Add a > to the beginning of any line
	> to create a blockquote

`Underline text to have`
`First Header`
`============`
`Second Header`
`------------`
`or use #`
`# First Header`
`## Second Header`
`### Third Header`
	Underline text to have
	First Header
	Second Header
	or use #
	First Header
	Second Header
	Third Header

`Leave an empty line and type`
 	
`---`
`to get a horizontal rule.`
	Leave an empty line and type
	
	to get a horizontal rule.

Table 2: Formatting shortcuts on TEX.SE

Description	Shortcuts
Italics	Ctrl+I
Bold	Ctrl+B
Code	Ctrl+K
Bulleted list	Ctrl+U
Numbered list	Ctrl+O
Heading	Ctrl+H
Horizontal rule	Ctrl+R
Link	Ctrl+L
Image	Ctrl+G

The following three examples all give as a result the clickable word CTAN:

```
[CTAN](https://www.ctan.org/)
[CTAN][1]
[CTAN][ctan]

[1]: https://www.ctan.org/
[CTAN]: https://www.ctan.org/
```

For images, it is the same but with a `!` at the beginning:

```
![<image description>](<image url>)
![<image description>][<imgtag>]

[<imgtag>]: <image url>
```

Of course, usually you do not have a url for your image, but no need to worry about that, the Stack Exchange network has an image hosting platform via `imgur.com`.

To upload your image, just press the specific toolbar button or Ctrl+G, and drag and drop it or click to select it from your computer path. Remember the maximum loadable size of the image is 2 MiB.

The problem is TEX.SE does not allow you to attach a `.pdf` file; you have to somehow transform your output into `.png`, `.jpeg` or another uploadable format. We will see how to do it in the next subsection.

Returning to the formatting methods, a limited subset of HTML syntax can also be used, Table 3 shows some examples. Note that superscript or subscript text can be used for a whole sentence if you want to write it in a smaller font size.

A useful trick is to put an empty comment: `<!>` within two empty lines to separate a list and a code block, otherwise, Markdown does not understand where the list ends and the code begins and makes a mess.

Table 3: Examples of HTML syntax allowed on TEX.SE

HTML syntax	Result
`<kbd>Ctrl</kbd>`	Ctrl
`<sub>subscript</sub>`	subscript
`<sup>superscript</sup>`	superscript
`<s>cancelled</s>`	~~cancelled~~

The syntax of Table 1 does not work in titles of questions; they cannot be formatted, and must be written in plain text only.

In comments, you can use only bold, italics, inline code and links. The rest is not supported, but you can contact the users who commented before you by writing `@username`. This way of pinging, on the other hand, does not work in posts.

For more info, please look at the TEX.SE help page: `tex.stackexchange.com/help/formatting`.

3.2 Creating the image to attach

Now let us see how to create an uploadable image format starting from our `.pdf` file.

If only a little piece of your document is needed, such as a mathematical formula, a Ti*k*Z picture, or a small table, the quickest way is just to make a screenshot and crop the image with some graphics editor, or copy only a subrectangle of the screen and save it as an image. The way to do these depends on your operating system.

You can also transform your `.pdf` into a `.png` file, and crop it, via `imgur.com`, even without an account. From the home page of that site click on "New post", upload your `.pdf`, choose "Edit image" from the drop-down menu which appears if you pass the cursor over your uploaded image, crop it, save it and then download your `.png` by choosing "Download image" from the same drop-down menu.

For cropping, there is also the tool `pdfcrop`, included in your TEX distribution. This calculates the bounding box of each page of your document and generates an output PDF file with margins removed. It could be useful to add `\pagestyle{empty}` to your MWE to switch off page numbering, if page numbers are not relevant for the post topic.

Another very easy tool included in your distribution, `pdftoppm`, converts PDF files to color image files in Portable Pixmap (PPM) format, grayscale image files in Portable Graymap (PGM) format, or monochrome image files in Portable Bitmap (PBM) format.

It creates an image file for *each page* of your PDF file and also has the options `-png` and `-jpeg` to respectively generate a PNG or a JPEG instead of a PPM file.

The syntax is:

```
pdftoppm [options] PDF-file PPM-root
```

For example, suppose you have to convert a document which contains only a little table and has no page numbers, named `yourdoc.pdf`. If you run:

```
pdfcrop yourdoc.pdf
pdftoppm -png yourdoc-crop.pdf yourdoc
```

you will get a document with your table only, no white space around it, in PDF (`yourdoc-crop.pdf`) and PNG (`yourdoc.png`) formats.

If your `documentclass` is `standalone`[3] or your output is not affected by the class of the document, you can use the `convert` option of `standalone` to transform it into an image format (`.png` is recommended, but others are also supported); see Section 5.6 of its documentation.

The `standalone` conversion is done by an external image converter program, which you need to install, and when you compile your document you have to use the `-shell-escape` option. By default, ImageMagick's conversion program is used.

The tools of ImageMagick can also be used directly and can be useful if you need to convert more than one piece of a document, for example running:

```
convert -density 300 yourfile.pdf yourfile.png
```

you will get a `.png` file for each page of your `.pdf`. Of course, the `convert` command has many options to improve the quality of your image, for more info see `www.imagemagick.org/script/convert.php`.

If your document has many pages and it is useful to attach all of them to your post, the `pdfpages` package may help you. It can arrange more than one page of another PDF document on one sheet of paper.

Suppose you have a document of four pages, let us call it `duckument.pdf`. You can create a new one-page document in this way:

```
\documentclass{article}
\usepackage{pdfpages}
\begin{document}
\includepdf[pages=-,nup=2x2]{duckument.pdf}
\end{document}
```

and then include this one page document (see Figure 2) to your post with one of the methods shown above.

[3] `https://ctan.org/pkg/standalone`

Figure 2: Example of `pdfpages` output.

For more info, see the posts "How does one add a LaTeX output to a question/answer?"[4] on Meta and "Compile a LaTeX document into a PNG image that's as short as possible"[5] on the main site.

4 Conclusions

I hope TEX.SE formatting rules don't worry you any more. However, if you are in trouble, remember:

***All we need is* duck!**

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

[4] `tex.meta.stackexchange.com/questions/2781/how-does-one-add-a-latex-output-to-a-question-answer`.

[5] `tex.stackexchange.com/questions/11866/compile-a-latex-document-into-a-png-image-thats-as-short-as-possible`.

Managing the paper trail of student projects: `datatool` and more

B. Tomas Johansson

Abstract

It is described how the paper trail for final year undergraduate projects can be handled using the package `datatool` and a master file of student data. In particular, it is shown how to generate a list of students and supervisors, a randomised mark sheet and timetable for project presentations, and how to sum the marks. To achieve this, the `ifthen` and `lcg` packages are used as well.

1 Introduction

At our institution, being responsible (or coordinator) for a final year undergraduate report writing module typically involves keeping track of around a hundred students, their project titles, supervisors and examiners. Moreover, various documents have to be produced, for example, a list of students with accompanying project titles and supervisors, a mark sheet as well as a timetable for the oral presentations, and a document having the final marks, to mention only a few. Naively setting up those documents in LaTeX by copying in student data, one finds that as the term goes by, students drop out or are added, supervisors turn up informing you that a project title has changed, or it is suddenly agreed that the timetable for the oral presentations should not be in name order but randomised. Starting feverishly to make changes to each document, most of us soon lose track of which data one has changed for what student and in what document.

A remedy is given by the package `datatool` [4]. Then only a master file of student data is needed, and documents are generated by reading from that file. Focus here is on basic constructions relevant for managing undergraduate projects; examples of handling marks of students and printing them out can be found in [4].

A csv-file with student data first needs to be constructed. It is a text file with comma-separated items on each line (other separators are also allowed). We'll give examples using this sample `students.csv`:

```
FirstName,Surname,Title,Supervisor,Examiner
Joe,Allen,Prime Numbers,Gert Li,Abbie Wan
Deeksha,Bhai,Inverse Problems,Alice Ince,Ma Ren
Gina,Gil,Algebraic Notes,Gert Li,Bob Al Turner
```

The first row consists of key words to be used when constructing documents, and the remaining rows are filled with student names and their data. For simplicity, only the students' names are split into first name and surname. One can easily add more keywords to the file such as `ReportMark` and `VivaMark`, and fill in the corresponding marks.

To generate a list of students, their supervisors and the titles of the projects, one can proceed as follows. In the preamble of a `.tex` file put

```
\usepackage{datatool}
\DTLloaddb{students}{students.csv}
\newcounter{nrstudents}
```

The command starting with `\DTLloaddb` has the effect of storing the database from the csv-file in a variable named `students`. The counter `nrstudents` will be used to number the students.

The lines below generate a table with student names and their corresponding supervisors, also numbering each student:

```
\begin{tabular}{rll}
 & Student & Supervisor\\
\DTLforeach{students}
  {\firstname=FirstName,\surname=Surname,
   \supervisor=Supervisor}
  {\stepcounter{nrstudents} \thenrstudents.
   & \firstname \space \surname
   & \supervisor\\}
\end{tabular}
```

Here, the command `\DTLforeach` goes through each line in `students` and stores the surname, first name and the supervisor's name of that line in the variables `firstname`, `surname` and `supervisor`, respectively. The keywords to pick out the data are given in the first line of the file `students.csv`. The number $1, 2, \ldots$, is in the first column thanks to the counter `studentnr`. A basic table is the result, as below; the design of such tables is a separate issue not dealt with here.

Note that if, for example, the name of the examiner should also be present in the table, then simply adjust to include `\examiner=Examiner` in the `\DTLforeach` command, and put `\examiner` in a separate column.

	Student	Supervisor
1.	Joe Allen	Gert Li
2.	Deeksha Bhai	Alice Ince
3.	Gina Gil	Gert Li

To get a corresponding list of titles, use

```
\setcounter{nrstudents}{0}
\noindent\\
\DTLforeach{students}
  {\title=Title}
  {\stepcounter{nrstudents}
   \thenrstudents. \title \\}
```

where again `\DTLforeach` is the key part. The result:

1. Prime Numbers
2. Inverse Problems
3. Algebraic Notes

If changes are made to the data in `students.csv`, it is only necessary to run the file having the above

commands to obtain an updated list of students, supervisors and project titles. This is a clear advantage compared to copying in the data in the file itself, especially as the number of files grow.

A typical question is how many projects a supervisor has, to check that there is a fair work load. To count and print out the number of projects assigned to a supervisor, do

```
\def\sumpr{0}
\DTLforeach[\DTLiseq{\supervisor}{Gert Li}]
  {students}{\supervisor=Supervisor}
  {\DTLadd{\sumpr}{\sumpr}{1}}
\sumpr
```

Here, a sum variable `\sumpr` is defined and declared to be zero. The data in `students` is sifted through `\DTLforeach`, where `\supervisor` is defined equal to Gert Li via `\DTLiseq`. If the conditional statement `\supervisor=Supervisor` is true, add one to the variable `\sumpr` (done via the command `\DTLadd`). The number of projects for the given supervisor is then printed with `\sumpr`.

Rather than counting the number of projects for each supervisor, it can be helpful to sort the data with respect to the name of the supervisors (it would have been more realistic to split supervisor names into first name and surname but to keep it simple a list is obtained here sorted with respect to the first name of the supervisors). Sorting is done with

```
\DTLsort{Supervisor}{students}
```

The data in `students` is now sorted and using the commands above, a table can be generated with the rows in alphabetical order of the supervisors; it is left for the reader to try.

For the oral presentations, staff need a mark sheet to score each student. This mark sheet should contain the name of the student, project title and time of the presentation, and have a table for scoring: the level of the content, mastery of the subject, quality of the slides, and presentation skills. Moreover, an overall mark should be given. A complication is that the order of the presentations shall be randomised.

The package `lcg` [2] generates random numbers. Put in the preamble of a `.tex` file for the mark sheet,

```
\usepackage{lcg}
\usepackage{ifthen}
\usepackage{datatool}
\DTLloaddb{students}{students.csv}
\newcounter{hour}\setcounter{hour}{1}
\newcounter{minutes}
```

The counters keep track of the time of a presentation. Assume that each presentation is 10 minutes, with a 10 minute break after 50 minutes. The hour is set to one to have an afternoon session starting at 1 pm.

To generate the mark sheet, first a column `Random` of random numbers is appended (on the fly) to the database. The command `\rand` from the package `lcg` [2] is invoked to generate random numbers. The data is then sorted with respect to this new column,

```
\DTLforeach{students}{}
  {\rand \DTLappendtorow{Random}{\arabic{rand}}}
\DTLsort{Random}{students}
```

It does not matter much to us if some random numbers are equal, the mixing in the database is still sufficiently far from name order. Then continue with

```
\DTLforeach{students}
  {\firstname=FirstName,\surname=Surname,\title=Title}
  {\noindent\\
   Student name: \firstname \space \surname \\
   Project Title: \title\\
   Time: \thehour.\theminutes 0 pm \\
   \addtocounter{minutes}{1}
```

If a presentation is to happen for example at 1.20 pm then `\thehour=1` and `\theminutes=2`. Since the presentations are 10 minutes each, the counter `\minutes` has to be increased by 1 (adding 0 to it manually).

The next part of the mark sheet is a table for scoring. This does not contain any real complication and is only included here for completeness

```
\begin{center}
\begin{tabular}{|l|c|c|c|c|}
\hline
& Poor & Fair & Good & Excellent \\ \hline
Level of content & & & &          \\ \hline
Mastery of subject & & & &       \\ \hline
Slides & & & &                   \\ \hline
Presentation skills & & & &      \\ \hline
\end{tabular}
\end{center}
Please give overall mark between 0 (lowest) to
100 (highest):
\begin{center}\thicklines
\framebox[.98\textwidth][c]{
  \parbox{.95\textwidth}
    {Additional Comments:\noindent\\[0.3cm]}}
\end{center}
```

A conditional statement is written at the end of the file using the `ifthen` package [1], checking whether five talks in a row have been given (that is, if `\theminutes=5`). If five talks have been presented, the counter for the hour is increased by one and the counter for the minutes is reset to zero. Otherwise nothing additional is done. This is coded as

```
  \ifthenelse{\equal{\theminutes}{5}}
    {\addtocounter{hour}{1}\setcounter{minutes}{0}}
    {}
}% end group from \noindent above
```

and this ends the construction of an elementary mark sheet for scoring oral presentations. A part of the file is shown at the top of the next page.

A timetable for the presentations also needs to be generated. Put the same commands in the preamble as for the mark sheet, and keep the randomisation.

Student name: Gina Gil
Project Title: Algebraic Notes
Time: 1.40 pm

	Poor	Fair	Good	Excellent
Level of content				
Mastery of subject				
Slides				
Presentation skills				

Please give overall mark between 0 (lowest) to 100 (highest):

Additional Comments:

Figure 1: Simple mark sheet example output.

Students are traversed writing out the data in a table and increasing the time of the presentation:

```
\begin{tabular}{lr}
\DTLforeach{students}
  {\firstname=FirstName,\surname=Surname}
  {\firstname \space \surname
   & \thehour.\theminutes 0 pm --
   \addtocounter{minutes}{1}
   \thehour.\theminutes 0 pm\\
```

Following this, if five consecutive talks have been given, the text "Break" and the time of the break are stated, otherwise a new line is started,

```
\ifthenelse{\equal{\theminutes}{5}}
  {\\ \setcounter{minutes}{0}
   Break \thehour.50 pm --
   \addtocounter{hour}{1} \thehour.00 pm
   & \\ \\}
  {\\[-0.4cm]}
}% ends \firstname... group above
\end{tabular}
```

This finishes a timetable for oral presentations, and part of such a file is this:

Gina Gil	1.40 pm – 1.50 pm
Break 1.50 pm – 2.00 pm	
Joe Allen	2.00 pm – 2.10 pm
Deeksha Bhai	2.10 pm – 2.20 pm

The automatic process for the randomised mark sheet and timetable takes away the painstaking job of manually assigning students and their timeslots. Having to make possible changes in the files makes the manual approach even less amusing. Instead only the csv-file needs to be updated (data for that file can often be generated from the student administration).

Let us mention one further construction. Expand the above csv-file with two columns named `ReportMark` and `VivaMark`, and fill in the corresponding marks. That is, a mark for the written report and a separate mark for the oral presentation. The total mark is the weighted sum of these two marks, with the written report counting for, say, 90% and the oral presentation for 10%. A file reporting the student names, marks and the total mark is to be constructed. In the preamble, put

```
\usepackage{datatool}
\DTLloaddb{students}{students.csv}
```

The remaining lines are

```
\begin{tabular}{rccc}
Student & Report Mark & Viva Mark & Final Mark\\
\DTLforeach{students}
  {\firstname=FirstName,\surname=Surname,
  \reportmark=ReportMark,\vivamark=VivaMark}
  {  \firstname \space \surname
   & \reportmark & \vivamark
   & \FPeval{\result}
           {round(0.9*\reportmark+0.1*\vivamark,0)}
     \result \\}
\end{tabular}
```

Here `\FPeval` (a wrapper in `datatool` of a command from the package `fp` [3]) stores in `\result` the value, first rounded to zero decimals, of the weighted sum `0.9*\reportmark+0.1*\vivamark` (more advanced calculations of marks and how to write them directly into the csv-file is given in [4, Sections 6.6–9]). An example of the outcome is

Student	Report Mark	Viva Mark	Final Mark
Joe Allen	68	82	69
Deeksha Bhai	60	65	61
Gina Gil	28	38	29

Other documents can be generated similarly with `datatool`, for example, a certificate for each student with their individual name and final mark, and an attendance sheet with the name of a student and supervisor filled in (to be used to record meetings with the supervisor). Personalized e-mail messages can also be produced.

Limitations of `datatool`, such as being slow on sorting, are stated in the documentation [4]. For our moderately sized documents, no problems have been experienced.

To conclude, involving the package `datatool`, managing the paper trail of student projects becomes manageable.

References

[1] David Carlisle, The `ifthen` package, `ctan.org/pkg/ifthen`
[2] Erich Janka, The `lcg` package, `ctan.org/pkg/lcg`
[3] Michael Mehlich, The `fp` package, `ctan.org/pkg/fp`
[4] Nicola L. C. Talbot, The `datatool` package, `ctan.org/pkg/datatool`

◇ B. Tomas Johansson
School of Mathematics, Aston University
Birmingham, B4 7ET, UK
`b.t.johansson (at) fastem dot com`
ORCID 0000-0001-9066-7922

Interview with Kris Holmes

David Walden

Kris Holmes is one-half of the Bigelow & Holmes design studio. She has worked in the areas of typeface design, calligraphy, lettering, signage and graphic design, screenwriting, filmmaking, and writing about the preceding. The Kris Holmes Dossier, a keepsake for the April 2012 presentation of the Frederic W. Goudy Award to Kris, reviews her career to that point.

The interview is a chronological oral history interleaved with discussions of examples of Kris Holmes' work. The interview took place on June 25, 2018, at the Cary Graphic Arts Collection at the Rochester Institute of Technology (RIT).[1]

1 Youth, calligraphy, and lettering

David Walden, interviewer: A few years ago, you told me you were raised on a fruit ranch in the San Joaquin Valley of California. Please tell me a little bit about your family, the ranch, and your education through high school.

Kris Holmes, interviewee: The farm that my parents lived on when I was born was a 90-acre farm on Zediker Avenue in a little town called Parlier, California. The nearest hospital was in Reedley, California, so that's where I was born. I have five brothers and sisters. My parents came to California from Oklahoma as part of the dust bowl migration, and they worked their way to eventually being farm owners in the San Joaquin Valley. So my early school years were spent at a little school called Riverview Elementary School, and then Reedley High School. It was a very nurturing environment. I had very sincere teachers, who encouraged me to work toward a college scholarship. My parents were not that enthusiastic about education. I think they assumed I would do what my sisters did, which was marry a local guy and stay in the area. But I was just born a curious person and I worked really hard in high school, and I ended up getting a very nice scholarship to Reed College in Portland, Oregon. So that would've been in the spring of 1968 when I got the notice from Reed, and in the fall of 1968 my brother drove me up to Portland with my bicycle in the back of the car. And that was the beginning of my adult life.

D: During your youth, did you do sports or have hobbies?

K: I didn't do sports too much. I had many hobbies. I loved to draw, I loved to do art, I loved sewing — I sewed almost everything that I wore, as did many of my girlfriends. Nobody in that area had much money so all of us were taught to sew and we all enjoyed sewing. In fact, we had little competitions to see who could sew a dress for the least amount of money. We would recycle our mothers' dresses and things. We kind of made a fun life for ourselves without money. We had televisions but other than that, we didn't have much entertainment. Everybody worked on their family farm, and then we just did kid stuff the rest of the time. It was a nice way to grow up.

D: When we talked by e-mail a few years ago, you mentioned your teacher, Roland Jenkins, who introduced you to the American transcendentalists, and you said that had an influence on you. Can you tell me a bit about that influence and how it went on?

K: Mr. Jenkins was one of my — I get a little emotional thinking about him — he was a teacher at Reedley High School and every teacher there got one period a day which was their free period, and he was a smoker so he liked to go to the faculty lounge and smoke during his free period. One year he had a group of students who were trying for college scholarships, and so he decided that he would use his free period to teach our first conference-style class on American Literature. And we read everything; we read Stephen Crane, we read *Walden*, we read Thomas Wolfe, we read *Moby Dick*; we just spent the whole semester reading all of these magnificent American authors. And because it was a conference-style class, we talked about what we had read and we each presented papers that focused on something we loved. Mr. Jenkins introduced all of us to the idea of independent thinking. To read *Walden* at age 16 is a perfect time to read it because you're kind of thinking that way, anyhow, and this idea of a guy that built his own house in the woods and lived deliberately is such a powerful idea, especially to a young person. Mr. Jenkins is the best example of what I mean about teachers that were so encouraging and that really changed everybody's life.

D: How did you find Reed College? You had the whole California college and university system to choose from, and Fresno State couldn't have been very far away.

K: Nope, I could've driven over there. Well, I had Plan A, Plan B, Plan C, and Plan D when I applied to college. Plan A was Reed College because two of the girls that were my best friends at home had an uncle who taught there, Professor Wiest. He taught psychology at Reed and they went to visit Reed and they came back with a glowing description of a place where everybody's smart and you just sit around tables and talk about things, and people wear serapes to class and you go barefoot, and they just thought it was heaven. So I did some more research on it and I felt that it was a place that would be very different than the San Joaquin Valley, which is certainly true, but a place that would be maybe more like Mr. Jenkins' class. I think he affected my life in that way, too. My Plan B was UC Santa Cruz, which at that time was called the "Poet's Campus." My Plan C was UC Berkeley. My Plan D was Fresno State. I was going to go to college, one way or another. But luckily I got a nice scholarship to Reed, and so I was on my way to my first choice.

D: You have said you went there in 1968.

K: I went there in the fall of 1968, and I went there for two and a half years; and then I left. It was going to be a temporary leave of absence, but it turned out to be permanent. I left Reed, I think, for kind of the same reason that Steve Jobs said that he left Reed. At that time, you got a scholarship when you went in but the tuition kept getting higher and higher. At some point, if you came from a working class family or a farm family, it was very, very hard for them to keep up with tuition hikes, even with loans and jobs and everything. It was just so expensive that I started thinking that I wanted to leave and just start working.

D: You were studying liberal arts at Reed?

K: Yes. I was majoring in literature when I went there. However, in my second semester, I took an energizing and intelligent Modern Dance class from Judy Massee, the dance professor, and quickly became devoted to studying modern dance. Then I met Lloyd Reynolds, and that changed everything for me.

D: It was a calligraphy course, presumably?

K: That's right. I had always liked to draw, and I especially had always liked drawing lettering. But when I got to Reed, Lloyd Reynolds was this luminous presence at Reed College, and everybody took his class. If you were a chemistry major, you took his class; if you were pre-med you took his class; if you were an art major you took his class; and interestingly enough, many of the people that I know who took his class way back in 1968, when I talk to them today, they still sit down and practice calligraphy sometimes. He really had this lifelong influence on so many of us. When I was a freshman, I was only there a few weeks and somebody said, hey, you know you can't get into Lloyd Reynolds' class but he teaches a little private workshop on Tuesday afternoons. Reynolds would stay a couple of hours after class to teach people that couldn't get into the class. I ran right over there, and somebody loaned me a pen, and then Lloyd loaned me a better pen, and he just sat down and put it in my hand and showed me how to do calligraphy. So that would've been the fall of 1968, so 50 years ago.

D: And his motivation? What do you think drove him to help so many people?

K: He wanted to change the world with calligraphy. He was very discouraged by the fact that you had to go to an art museum to see art, and that it was all kind of run by somebody else. He thought art should be in the hands of the people. He was a fan of the philosopher Ananada Coomaraswamy and William Morris, and so he wanted to give us art that we could do with our own hands. To make a beautiful laundry list, he used to say, is as important as anything you see in a museum. And I think he succeeded in that; he certainly succeeded in presenting the whole world to us through the history of writing and the history of calligraphy.

D: I have looked up on the Web many things you've said, and you say somewhere that what you learned from Lloyd Reynolds, you use in everything you design. Please say a few more words about that.

K: When Chuck [Charles Bigelow[2]] and I design a typeface, we start out with just a blank place there on the screen or on the paper. In our case, we like to start with paper. So where do you start? Since both Chuck and I studied with Lloyd Reynolds, we have this common language, and we start with the letter forms that he taught us. We get out calligraphy tools and we sit there and we draw, and we talk to each other; for instance, "Okay, this is a correct lower case 'e' for the Renaissance, how does it need to be different for modern technology?" For us, what we learned from Reynolds is the common ground that we start from. And you know, I don't think we've ever argued about a design decision. We make a decision, based on our common ground and if we disagree we say okay, let's try it both ways. It's very

smooth sailing for designing for us, and Reynolds really is in everything.

D: You have said that Reynolds followed the first canon of Chinese brush painting; "heavenly breath's rhythm vitalizes movement." What does that mean?

K: I wish I could say it in Chinese. Well, the other thing that Reynolds used to say is a quote from an old jazz tune, "it don't mean a thing if it ain't got that swing." And that's a pretty good translation of "heavenly breath's rhythm vitalizes movement." And what it means is that these letters aren't still. You look at any one of these letters that we see around us, and what you're really looking at is the path of the moving hand of the person that did it. Even in modern type, although we don't do it by writing in a stroke, those characters are outlines based on stroked letters and it gives them a certain vitality. It's like when somebody sees your handwriting they say, "oh, that's Dave's handwriting; oh, that's Kris' handwriting." How do they know that? Because it's been vitalized by some energy in yourself. So to me, that's what that phrase means.[3]

D: I've also read that you studied calligraphy and brush writing with Robert Palladino at Reed.

K: I did. What happened is that the summer after my freshman year when I really had just been taking that little workshop with Reynolds, Reynolds decided to retire and so he needed to choose somebody to take over his classes and he chose Robert Palladino.[4] Robert Palladino had been a Trappist monk and he had gone, on Reynolds' recommendation, to Iowa to study with Father Edward Catich from whom Palladino learned the art of brush written Roman capitals.[5] Catich wrote a very inspiring book called *Origin of the Serif*, and he was the person who carefully examined the actual inscription and figured out that the characters from the Trajan column were not drawn as outlines on the side of the column, they were actually first written with a brush like this one, which is a sign painter's brush.[6] I brought my brush so you can see that it has a broad edge like a calligraphy pen, but it's a brush. It's called a Bright's brush.

Robert Palladino had studied with Catich; and also because of his studies in the priesthood, he could read Latin. So Reynolds chose him to take over his teaching at Reed College, and I took two years of classes with him. We stayed in touch and in the 80s Chuck arranged for Palladino to give a workshop at the Imagen Corporation in Silicon Valley. Steve Matteson and Tom Rickner, now of Monotype, remember that Palladino workshop with happiness.

D: You said that you left Reed because you needed to work. What work did you find?

K: Some pretty awful jobs, really. [Laughs.] And actually, briefly, I had a job at Hallmark Cards out in Kansas City, doing lettering for them.

D: Remotely?

K: No, I lived there for two months, but didn't like it very much so I moved back to Portland, scraped by doing calligraphy jobs, and I went back to Portland State University to try and get a teaching certificate. This turned out to be a good thing, as this was how I met up with Chuck again.

I had first met Chuck when I was a sophomore at Reed, in a mime class taught by an Italian mime named Carlo Mazzone[-Clementi]. I was taking the class, and Chuck had already graduated but he came down to Reed to take it too. So that was where I met him, and for the first five years that we knew each other, we were just part of a great big group of friends at Reed or in Portland who all had common interests in theater, dance, art, ... So, we had known each other about five years, and there I was in Portland again trying to make a living, and I went down to a monthly magazine called the Oregon Times. And Chuck was the art director there and I said, "Well, do you need any help?" It turned out that he needed help, so we started working together once a month putting this magazine together and one thing led to another, and pretty soon we were an item, which delighted all of our friends. Everybody said, "Oh, that's great." It was because I had to scrape together a living that I met Chuck again, so something good came out of it.

D: But then, if I have the chronology right, you left Portland and went off to New York City.

K: That's correct.

D: And Chuck was still in Portland?

K: Yes. Because of the inspiring teaching of Judy Massee, I had this idea in my head that I really wanted to be a dancer and I thought if I didn't go and take a shot at it I would always regret that, so I moved to Manhattan for about nine months, although I always I felt like I had left something behind in Portland; that was Chuck. I immediately saw that I could never be a professional dancer; you had to be much better technically than I was. But I did some good things in New York. I got a job at Harcourt, Brace, Jovanovich as a paste-up girl where I learned about publishing, and then I took Ed Benguiat's lettering class at night. Ed Benguiat was the teacher

who kind of took me from stroke-based calligraphic letters to drawn letters.

D: Can you say a word about the distinction between calligraphy and lettering?

K: Yes, I can. People say that calligraphy literally means "beautiful writing," but to me it also means efficient writing. When you write something calligraphically, every major part of a letter is a single stroke. For instance, if I were calligraphing an "R", I would do the left vertical stroke, then the serif on the bottom left of that stroke, then the bowl, then short horizontal stroke, then the diagonal leg and, bam, this complex letter is done in just five strokes.

See the animation by Holmes at
`tug.org/interviews/holmes-BrushCapR.m4v`

But if I was going to hand letter an "R", I would take a pointed instrument, not an instrument with a broad edge, and I would draw the outline in many short careful stokes. It is a much slower process. This is how I draw our typefaces, because they're not calligraphy, they're outlines. As you can see — I'm mimicking it on the table — it takes a long time to do that.

Figure 1. Of this image, Holmes says, "In the background is that same brush written R. The outline is of a Lucida Bright capital R. Obviously it is different, but the differences are only the technical alterations that we make as type designers. The basic letterform remains."

Calligraphy was really developed for the reproduction of books before the invention of printing, and it's a very, very efficient way to write. Also, of course, completely flexible. With type, you only have the choice of the characters that are in your font that you're using. But calligraphically, if the spirit moves you, or the layout demands it, you can put a swash on a letter.

D: A digression, please. In high school, I spent three years doing mechanical drawing and therefore, architectural or mechanical drawing lettering. What is that? I thought of it as lettering.

K: Did you use a template?

D: No, we had to learn to letter and we used that funny architectural style with the long tails on the "R"s, "H"s would have a little horizontal squiggle, . . .

K: Did you do it freehand? You just went one, two three?

D: Freehand. One down, two around, and three the long tail.

K: I would say that's calligraphy. It's not based on a historical style but it's based on modern architectural lettering, and you're doing it by hand, and you're doing it very fast and efficiently; I would say it's more calligraphic than lettered.

D: That's an interesting distinction, thank you.

K: I don't know if everybody agrees with me on that. [Laughs.]

D: It's your interview.

K: Okay!

2 Bigelow & Holmes

D: As I understand your chronology from Chuck, after his mentor Jack Stauffacher [see "Remembering Jack Stauffacher" on `bigelowandholmes.typepad.com`[7]] invited him to San Francisco to attend slide lectures by three Swiss type designer-teachers, it came to him that he could be in the type design business, too, but he would need a partner.

K: Yup.

D: And he called you in New York. Was this a surprise that you were going to become a partner? I mean, you knew Chuck already. Had you anticipated this in any way?

K: Well, I knew Chuck already, and I knew that I really would like to figure out some way to be back in Portland with him, but I didn't want to go back to scraping together a living as I had been doing. As I was saying, the whole dance thing in New York wasn't working out, and I took the class from Ed Benguiat, and Chuck had this idea that I would move back to Portland and we would start our studio, Bigelow & Holmes. And that's exactly what we did. I came back to Portland, I got a job teaching part time, and we opened a little studio on Southeast Salmon Street in Portland.

D: Teaching part time where?

K: Portland State University, and the Museum Art School in Portland.

K: I think Chuck probably told you the story of how we got our first commission and he got a grant from the National Endowment to add the phonetic characters for Native American languages to a typeface called Syntax, designed by Hans Ed. Meier. We did that job; and then, Andrew Hoyem was looking for somebody to design special initial letters for his big limited-edition printing of Moby Dick. The Cary Collection has a copy of that Moby Dick here, so you can look at that if you want.

D: Let's step back a second to the addition to Syntax. Is this it (Figure 2) here in this little 4-page keepsake brochure you gave me?[8]

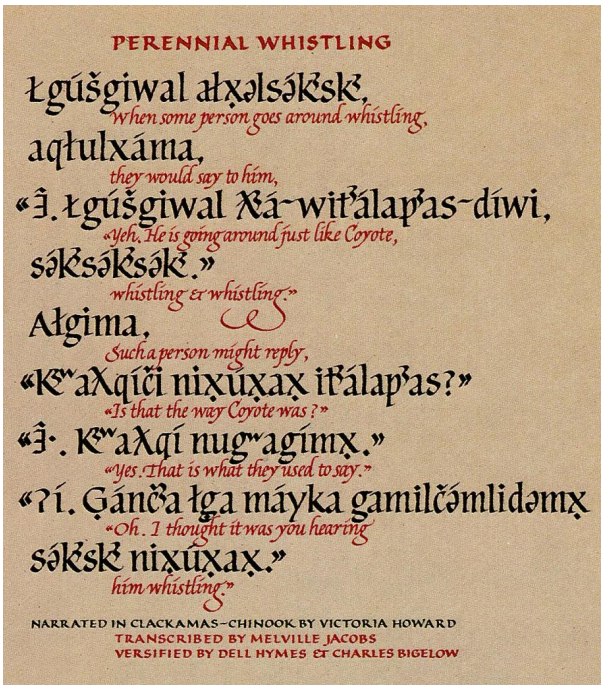


Figure 2. Calligraphy relating to Victoria Howard, from a keepsake given out by Bigelow & Holmes at a 1983 ATypI seminar.

K: This is not the Syntax typeface; it is my calligraphic research for that project. We were going to design the special phonetic characters to go along with Syntax; and Hans Meier, the designer of Syntax (Chuck wrote to him and he wrote back), was very enthusiastic and willing to help. He was really one of the people that opened the door for us. So I researched these characters by developing a calligraphic hand that included all the special characters, and this was it.

D: And both lines in each pair of lines are yours?

K: The whole thing is all hand done. The lines written in black are done in a special calligraphic hand that I developed for this project. This was done first, before we did the font with the special phonetic characters. Hans Meier's Syntax design is a very pure sans-serif, different from my calligraphy, but handwriting the characters was a way to understand them better. And that's a way that I like to work whenever I come across a project that I need to do something new with. When we were going to design the Greek characters to go with Lucida, the first thing I did was get out my pen and look at Greek manuscripts and figure out how to write that with a pen; because once I can write it with a pen, or marker, or a brush, I feel completely secure. It's like learning a language. I think, "okay, now, I can go from there." Otherwise, you're just copying what some other type designer did. This way I'm going right to the source of the characters, and it's a really efficient way to work. (By the way, this calligraphic hand I used for the Syntax study that later became the inspiration for a typeface called Sierra [Figure 11].)

D: While we're on this page, let's digress momentarily and look at the other pages in the 4-page keepsake. In it you have three examples of pieces that the brochure says are from the series you did of calligraphy based on women poets, I guess throughout history. Please say a few more words about that. The example we were just looking at says it is from text from Victoria Howard (c. 1865–1930) as transcribed by Melville Jacobs — I guess from Victoria Howard speaking in the Clackamas-Chinook language. How many pieces were there in that series about women poets?

Go to `tug.org/interviews/holmes-bigelow-clackamas.mp3` to hear Chuck Bigelow reciting the text from Figure 2 in Clackamas-Chinook (also known as Kiksht), as he typically does when showing that figure in talks.

K: I think there were five or six total, I'm not sure. I started that project because we were working with these stories by Victoria Howard. And I thought well here's this brilliant woman but illiterate in English, so her brilliance as a great storyteller was nearly unknown and ephemeral. Then I realized there are a lot of other women like that. Peig Sayers [another page in the keepsake brochure] is well known in Ireland; illiterate in Irish, very poor, but a fabulous storyteller in Irish. People said you'd sit and listen to her tell stories, and you'd just walk out the door, and you didn't even know where you were, you had been

so lost in her stories. And also there were women troubadours. So I thought I'd do a series of pieces of calligraphy that were based on the work of women who got no little or no credit — because little was preserved of their work. I thought I would preserve it in the best possible way, by calligraphing it.

D: And in each of these three examples — Victoria Howard (c. 1870–1930), Peig Sayers (1873–1958), Lombarda (c. 1190) — both pairs of lines are yours?

K: Everything.

D: So here in the Peig Sayers piece (Figure 3) is your interpretation of Irish lettering?

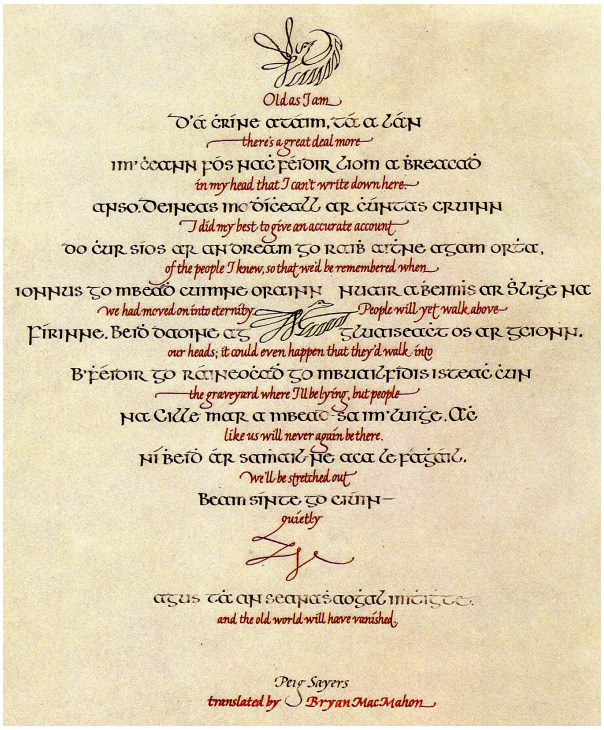


Figure 3. Calligraphy for the Peig Sayers page of the 1983 ATypI keepsake.

K: That's my interpretation of the formal hand used in the Book of Kells. The English is in chancery cursive. In the Lombarda piece [the third example in the 1983 ATypI keepsake], the capitals hand is an original hand I developed, a hand that is based on Roman square capitals, which is a historical style. This is a pretty good example of how I work because you see me copying historical styles and then inventing something that's new.

D: In each of these cases your practice was to find some originals, try to figure out how it was written, and get comfortable with writing it?

K: Yes. And I just did that for the piece that I contributed to the book The Cary Collection did in honor of Hermann Zapf, which we can show you before you leave, and I can send you a copy of it (Figure 4).

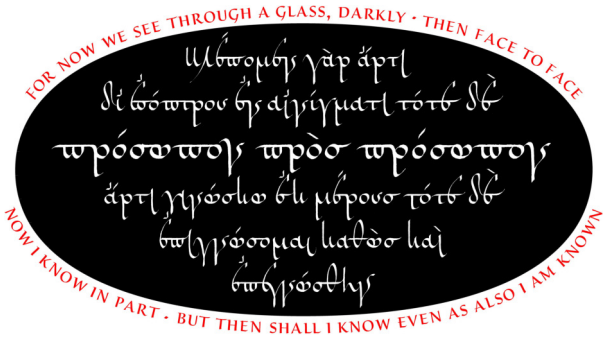


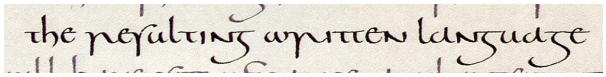
Figure 4.

This is my favorite thing to do with calligraphy — to learn a new hand, not just copying it letter by letter, but to learn it so I can just sit down and write it out.

D: And this is your calligraphy in this statement by Friedrich Neugebauer, on the first page of the keepsake?

K: That's my calligraphy. I think those characters were originally written on wax tablets. They are the informal cousins to the beautiful formal letters on the Trajan Inscription.

D: And you've done the calligraphy using a style from history, a style that, for instance, has these extreme "r"s and "s"es is the word "resulting".



Very interesting.

K: I found an image of a wax tablet written in this style, and I sat down and learned it. By the way, I taught a class at University of the Arts in Philadelphia, a one-week workshop, and I had my students do the same thing. I brought in manuscripts, everybody picked out a manuscript and they just sat there for 12 hours straight learning how to write it. And then they did their own personal piece based on that.

D: It says in this quote from Friedrich Neugebauer (in your calligraphy) that the words express a design philosophy of Bigelow & Holmes. I transcribed it into my writing so I could tell you what it says.

K: [Laughing] Say it.

D:

we as individuals
and lettering artists can
help to preserve our oral
tradition by choosing texts
that are meaningful to us and
that shed light onto our lives.
the resulting written language
will have significance and integrity.
this approach to work and art
could be a new beginning:
a reaffirmation of our
own age-old linguistic traditions
and the establishment of
positive directions for our
future use of language.

K: Yes. I'm so glad you reminded me that we had said that was our statement, and we still stand by that. You know, I was just talking to Chris Myers, who is the head of that program at University of the Arts, about why calligraphy is now considered kind of an artsy craftsy thing. To Chuck and me it's not; to us it's the root of everything that we do. That statement by Neugebauer really says it much better than I ever could. That is our philosophy.

D: Before I interrupted you a while back, you began to talk about the capital letters font Leviathan.

K: Yes, for that *Moby Dick* project.

D: You gave me another handout, which was a keepsake from a recent exhibit here at the Cary Collection (Figure 5).

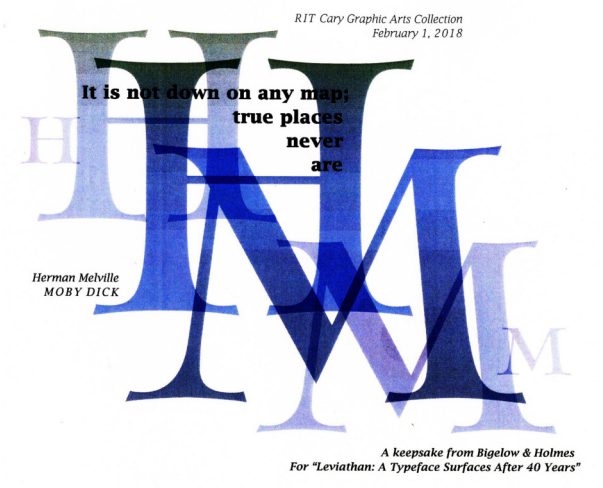


Figure 5. Cropped from a keepsake from Bigelow & Holmes for "Leviathan: A Typeface After 40 Years" (2018).

K: Yes. The exhibit was in February [2018], to celebrate the 40-year anniversary of this typeface. The Cary Collection put on a presentation and they put it right here, at this place on the table, their copy of the Moby Dick book; and Chuck and I gave a talk about how we designed the set of initial letters. It was a beautiful event.[9]

D: What led you to decide to have a 40-year anniversary of this?

K: I think it was our close relationship with the curators here at the Cary. You know, we mentioned that gee, it was 40 years ago that we did this. And the associate curator here, Amelia [Hugill-Fontanel], said, "Hey, let's have a 40-year celebration." And soon after we did that, Steve Matteson came out from Monotype because he was releasing his new revivals of Goudy types that he has done, and The Cary Collection had a similar celebration. The Cary Collection is kind of getting to be a place where you can announce a new type or celebrate an old typeface. There is a nice lettering and type community around here, so it's a really beautiful event in the evening. Everybody gets together.

D: Perhaps it's a more reflective phase in your career?

K: Maybe. I don't think that this means we're moving into a reflective phase; Chuck jokes that thirty-five years ago, everybody used to ask him about the future, now they ask him about the past.

3 Working and learning: Cambridge, MA, San Francisco Bay Area, Hawaii, UCLA

D: Let's go back, if you don't mind, to your education. We started at Reed, you did some studying in New York City; you didn't mention it but I read that you were at the School of Visual Arts. I guess that's where Ed Benguiat was. Then in 1979, you studied here at RIT — a brief summer course, I believe.

K: It was a two-week summer course. Hermann Zapf came from Germany and taught for two weeks. Those [big blue pages of letters on the next table being prepared for an upcoming Zapf exhibit at the Cary Collection] are samples Zapf made as he taught (Figure 6).

We came to Rochester, and that was another turning point in our careers — to meet Zapf and watch how he worked, and hear him talk about his work. It was just a great experience. So we've always thought of RIT as a stronghold of good lettering and good typography.

D: And, somewhere along the line, you finished your Bachelor's degree at Harvard Extension.

Figure 6. Photo taken by Chuck Bigelow during 1979 course by Zapf.

K: I finished it in 1982. Chuck was teaching at RISD and I had a job at Compugraphic up in Wilmington, Massachusetts. We wanted to live together, and so we found a place that was kind of in the middle, though it turned out to have torturous commutes for each of us. I would work all week at Compugraphic; then on Tuesday and Thursday nights I hightailed it down to Harvard Square and took classes there, and finished my degree. That was another great experience in my life; wonderful professors; I studied linguistics with Calvert Watkins, Islamic Art with Sheila Blair. I had a great life drawing teacher, so I was drawing all the time; and finished my degree just before Chuck got his job at Stanford, so I finished it up just before we left.

It was fun living in Cambridge. In the end, Chuck left his job at RISD and I left my job at Compugraphic, and then we were really in heaven. We had a little studio in our apartment on Irving Street, near Julia Child. We'd see her in the nearby market. We spent every evening working on ideas for typefaces.

D: Your type design at Compugraphic was for phototypesetters?

K: Yes. Compugraphic was the first place in the U.S. to install the Ikarus system, which is why I was interested in working there.

D: And you knew they had that already and therefore sought out the job?

K: I had sought out the job earlier. I wasn't sure what I wanted to do, but then they installed that Ikarus system just before I went to work there. So I was able to kind of do my analog drawing job and then sneak over and see how the digital system was working.

D: Did you already know the people in Germany who had done the Ikarus system?

K: Peter Karow at URW. We didn't know him yet.

D: Back to your education. Somewhere along the line you ended up at UCLA studying for an MFA.

K: Yes I did, years later. But in 1982, by the time I finished my degree at Harvard, I had already gotten a commission from Hell, the inventor of digital typesetting, to design some original typefaces for them.

D: How did they find you?

K: They found me because the Hell-Digiset typesetter was being introduced to the American market, and they wanted people to design some new types that would show off the high quality that it had for a typesetter at that time, and they wanted high quality versions of some historical typefaces. They had heard about Chuck and wanted him to be their American typographic consultant. Chuck was going to go on a job interview with Max Caflisch, who was Hell-Digiset's European type consultant. And before he went on the interview Chuck said, "look, they're looking for type designers, why don't you just put together your portfolio and come with me." And I said, "it's your interview, I can't go." He said, "Oh come on, it'll be fun." [Laughs.]

So he brought me along on the interview and Max Caflisch was kind of surprised to see a second person there, but then I pulled out of my portfolio all these letters that I had learned to do from Robert Palladino based on the Trajan Inscription. At that time, it was a very, very unusual skill and Max Caflisch looked at it and said. "Oh! This is interesting." So that was how I got the invitation to present proposals for original typefaces, and they also hired me to do a revival of Baskerville and a revival of Caslon especially for their digital typesetting machine. So that was how I got that commission, which I was still working on in 1982 when we moved to California.

D: And at that time you were communicating by airmail?

K: Yes. That is when I started a technique that I still use. What was happening is they had the Ikarus system at Hell in Kiel, Germany, and I was doing the drawings. I sent a first set of drawings on paper but they said that the size wasn't right, and I realized that this is a piece of paper traveling over damp atmosphere and dry atmosphere, and the paper was changing size and the digital constraints were too tight for that. So I started drawing on dimensionally stable Mylar, and it's a very nice surface to draw on, you can get a really beautiful fine line. I would draw the typefaces, sometimes based on material Max Caflisch had sent me, sometimes based on material that I had had blown up, or just my original sketches

from specimens in the Harvard Houghton library. I would make the drawings, I would make a nice blue-line copy of them, and then I would mail the originals to Germany, and they would digitize them on their Ikarus system there and then send me the bitmap versions for me to correct. It was a long, slow process but it was the only way to do it at that time.

D: How did you feel about doing a re-enactment of Baskerville or Caslon, as opposed to doing an original typeface?

K: I felt really good about it. You know, I learned so much. I had big blowups of the characters that were at the size I was going to draw out. But I also had my little printed specimen of the type and I had a magnifying system, almost like a little microscope. I would sit there and look through that microscope at the original printed samples, and then draw; look and then draw; and I did that for months and I learned so much that I felt just fine about doing that because there are so many decisions that you need to make. For example, Baskerville cut a different type for every size that was going to be printed. Well, if you're going to do a new Baskerville now, which size do you go by? What size is your model? You can't just say I'm going to use the 12 point model. That might be one of the sizes Baskerville or his punch-cutter didn't even cut so there are a lot of decisions that have to be made. I didn't even call it a redrawing, I would call it a revival.

D: Revival. Back to UCLA, this was after you were at Stanford? While you were at Stanford?

K: Chuck was at Stanford from 1982, and then we worked in the Bay Area for I think 14 years. And then, everything was going fine and we decided we wanted an adventure and we moved to rural Maui for four years. We lived up on the side of Haleakalā, the volcano, and we studied the native plants in Hawaii. We were still drawing type and doing business with people, but we were just doing it from rural Maui. So we studied native plants, I studied traditional hula and chanting, and we studied Hawaiian language and music; so we just did a few years of study.

D: But always running your studio.

K: Always running the studio. We thought we could do it from afar because of the Internet, but it was hard to do on Maui because they had an antiquated electrical system, so whenever there was a storm it would blow out our fax machine; and it turned out to be very difficult to run our business from Maui at that time.

D: I heard from Karl Berry, who I guess heard it from Chuck, that in Hawaii you grew roses …

K: Yes. Chuck had a vast collection of, I think, over 600 different kinds of roses. He was studying rose fragrances.

D: It was just part of the adventure.

K: Just part of the adventure. We've always been avid gardeners; you may have seen my short film *La Bloomba* (Figure 7).

D: [I had not, but now I have; it is here: `tinyurl.com/holmes-bloomba`]

K: That was my thesis film for UCLA and it was based on time lapse video I did of flowers opening and every flower in that movie is a flower from our garden. We just always loved gardening, both of us have. It's kind of our mutual hobby.

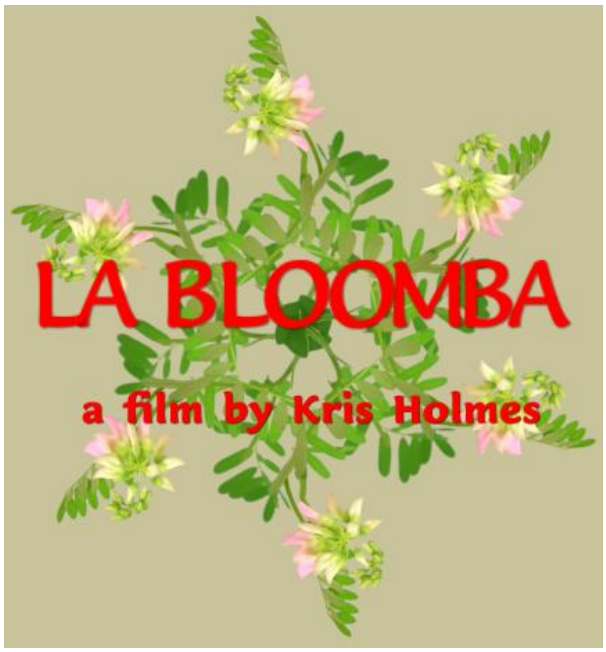


Figure 7. Title slide from *La Bloomba* film. Kris says, "… a special version of Lucida Casual which I prepared so I could animate the title from light to bold lettering, like all of the flowers blooming in the film."

D: So you went to Stanford, you were in the Bay Area for 14 years, you spent four years in Hawaii, and now do we get to UCLA?

K: Now we get to UCLA. By the end of about two years on Maui, we had really done everything we wanted to do there. It's a very isolated lifestyle. And so we started thinking about what's next, and Chuck said, "you know, I've always wanted to write screenplays," and I said, "I've always wanted to be an animator." So we decided that we would each apply to UCLA film school, we'd probably get rejected and

that would be the end of that. But we both got accepted into film school and so we moved to Los Angeles to be students again, which was one of the greatest experiences of my life.

It was just such a wonderful place; my department was headed up by Dan McLaughlin, he used to work for Charles and Ray Eames at their studio. Everybody was so supportive, the students were brilliant, everybody was working on great projects; it was an exciting atmosphere. Chuck was able to audit a linguistics class on the Mayan language K'iche', in which the greatest surviving epic of Native American literature had been written in its native script. It was just such an exciting campus to be around, and we absolutely loved it.

D: Meanwhile, your studio is at home and you're working.

K: Yes, and we're working. In fact, near the end of my first year at UCLA, Sun Microsystems' Java group asked me to design a Devanagari font used for Hindi, Marathi, and Sanskrit languages; so I had to stop coming to class for a few weeks while I worked on that. But it was okay; you know, it worked out in the end. So that was a great, great experience, going to film school.

D: Yes. But you said that you studied animation and Chuck studied screen writing, but the literature I found on you says that you wrote an award-winning screenplay, so you did some screen writing as well?

K: I did. I thought, "well, here I am at film school, why not try everything?" At the film school you could take classes in anything; it's just that you had to work hard and keep up. I took a class in costume design taught by Deborah Lynn Scott who had designed the costumes for the Titanic movie, I took screenwriting with Richard Walter, who taught Academy Award winners. I wrote a screenplay about Nikolai Vavilov, who was a botanist working under Stalin, who ended up starving to death in prison. It won a Sloan Foundation prize, for screenplays about science.

4 Operating a design studio

D: When you formed your company, way back in 1976, what was the image of how the two of you were going to work together; and how has that evolved over time?

K: Oh boy, I don't know that we had an image of how we would work together. I think that what we wanted is we wanted to be able to design type. I'm not sure that we have an image of that now. We just kind of take each job as it comes along. And the thing is that both of us can do everything required, so that I could do the whole thing by myself and Chuck could do the whole thing by himself. When we get a new job, we say "okay, who's really interested in this; who wants to do this; you want to do sketches, okay, how 'bout you do this, let's try this." Every job is a new, a new definition of what Bigelow & Holmes is. And we enjoy that.

D: I don't know where what I am about to say is from, but whoever wrote it says, "as principal artist at Bigelow & Holmes, Holmes is responsible for creation of over 100 digital typefaces, including conception, research, drawing, computer input, digital editing and production management." What does Chuck do?

K: [Laughing.] Chuck does some of that, too. Chuck does some of all of that.

D: You both do it all.

K: We both do it all. We do divide up some of the tasks of our business; I'm the president and accountant, he's the vice president and lawyer, stuff like that. He does much more writing than I do, which is fine by me. I do more calligraphy.

D: But I was interested to find papers written by you in your dossier that you gave me. Very interesting.[10,11]

K: Thank you.

D: This person, whoever it is, says you've designed 100 digital typefaces. I found other people saying 70, 300,[12] ... — different numbers in different things I read.

K: It's even worse than that because now, when we design something, we can do multiple master setups (Figure 8), so I can design poles, two or three poles, and then I can interpolate a hundred weights in between, if I want to. I'm not even sure how I would come up with a count. There are individual fonts, and families of fonts, and so on. A few years ago, we established a little online store to offer simple versions of our Lucida designs, and we put around 300 of our fonts on it, in different families like Lucida Sans, Lucida Casual, with different styles, and many weights in each style.

D: Perhaps it is not a relevant measurement anymore.

K: Maybe you could say I've done so many families of fonts. I have no idea what that number is. I don't have time to stop and count them all.

Weight Name	W3C	lowercase alphabet
UltraThin	100	abcdefghijklmnopqrstuvwxyz
ExtraThin	150	abcdefghijklmnopqrstuvwxyz
Thin	200	abcdefghijklmnopqrstuvwxyz
ExtraLite	250	abcdefghijklmnopqrstuvwxyz
Lite	300	abcdefghijklmnopqrstuvwxyz
Book	350	abcdefghijklmnopqrstuvwxyz
Text	375	abcdefghijklmnopqrstuvwxyz
Normal	400	abcdefghijklmnopqrstuvwxyz
Thick	425	abcdefghijklmnopqrstuvwxyz
ExtraThick	450	abcdefghijklmnopqrstuvwxyz
Dark	500	abcdefghijklmnopqrstuvwxyz
ExtraDark	550	abcdefghijklmnopqrstuvwxyz
Bold	600	abcdefghijklmnopqrstuvwxyz
ExtraBold	650	abcdefghijklmnopqrstuvwxyz
UltraBold	700	abcdefghijklmnopqrstuvwxyz
Black	800	abcdefghijklmnopqrstuvwxyz
ExtraBlack	900	abcdefghijklmnopqrstuvwxyz
UltraBlack	1000	abcdefghijklmnopqrstuvwxyz

Figure 8. A series of weights of lower case Lucida Sans. Theoretically, in the multiple master approach to font development, a few master designs can be done, e.g., UltraThin, Normal, and UltraBlack, and the other weights can be derived by interpolation.

D: Back to your job history, in a sense; once you started the company. It seems to me that you do three different things. You teach sometimes . . .

K: Yes.

D: . . . so really that's kind of individual; your company sometimes does work for hire, . . .

K: Yes.

D: . . . and then sometimes you do things on speculation and then license. Are those the three categories?

K: Those are three categories. We also do consulting; Chuck does more of that than I do. But I'll give you an example. Sometimes we do a typeface on commission, for instance, Apple Chancery. Apple was developing the TrueType GX program, which would enable a font to do what is now called character substitution in OpenType. Apple wanted a typeface to show off its new technology, so we designed Apple Chancery for them. It had all kinds of character substitutions: the lower case "e" has a regular "e", a descending "e", an "e" with a swash at the end, an "e" with a swash at the beginning; depending on where this "e" fell in the text, you would use different versions of the character. When we finished that project, it came up to a character count of 1001. I said, "I want to call it Scheherazade because of "The Thousand and One Nights" which I had read to prepare for Sheila Blair's class at Harvard." But Apple said, "no, Scheherazade won't fit on a drop down menu," so they called it Apple Chancery. So that was an example of a company saying we want you to do something for a specific reason.

D: And then they own it.

K: That's correct. In fact, we're embarking upon a similar project here with the Cary. We're doing a revitalization, a revival, of Baskerville. So it will be the second time that I've done this.

D: And why does Cary want to do that?

K: For their own use. We are going to design the typeface, and they'll be able to use it for labeling and cards and signs and posters and everything. They will own it, and we won't sell it through our store or any other way. On the other hand [years ago] for Lucida, Chuck had the idea that it would be nice to have serif and sans serif matching families, and because digital typesetting was just emerging, we wanted to design something that would look good at low resolution, like 300 dpi. So we just did it, and it's ours, and we license it. But we did it on speculation; nobody paid us to do that; it was just an idea that we had.

D: Yet you keep adding to it.

K: Oh yes, we do.

D: So somebody must be buying it.

K: Yes, somebody's licensing it; they're not buying it.

D: How do you sell something like that — mostly through your website, or through Myfonts, or . . .

K: No, we license mostly to corporations that want to bundle the font in their equipment or their software. We also license a large set of Lucida fonts to the TEX Users Group for mathematical typesetting.

D: I see. How does dealing with a big giant like Microsoft compare with dealing like with a tiny thing like TUG?

K: It's not that different. It's not like you make your presentation to all of Microsoft. You're usually only dealing with a core set of people, a group of two to five or six people. You get to know them well and they know you. So it's not as different as it might seem. But sometimes the core group of people leave and you get a new core group of people, and you just hope they're as good as the original set.

5 How Kris works

D: If I can digress again back to one of these handouts, was this (Figure 9) in the Illiterate Women's Poet series, too, or is this different?

K: This is by an ancient Greek woman poet, Anyte. She was literate. It was part of the research I did for

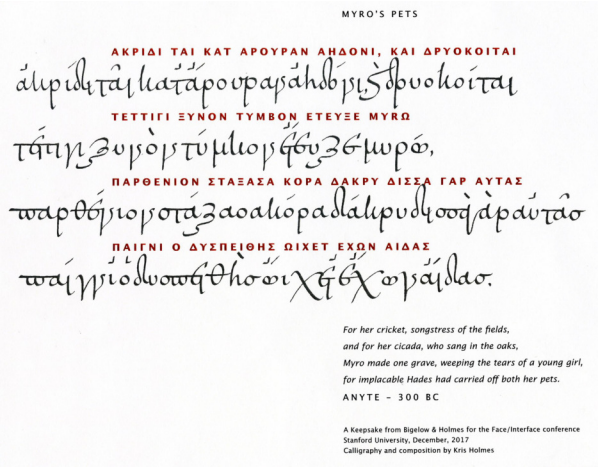


Figure 9. A keepsake from Bigelow & Holmes for the Face/Interface conference, Stanford University, December, 2017.

the Lucida Greek project. I was in London and I was in a little shop, and there was a book called *Greek Literary Hands A.D. 400–1600*. It wasn't professional calligraphy, it was handwriting by scribes who were preserving literature; and there was a very lovely script in there. So I did the thing I love to do. I sat down with the script and a piece of paper, and I figured out how to use it and how the letters change as they're next to each other. And then I found that very charming poem by Anyte and so I wrote it out. You know, Lloyd Reynolds used to say that "I calligraphed it" sounds pretentious. "Just say, 'I wrote it out'." And the older I get, the more I think yeah, he's right; "I wrote it out."

D: What you described is really interesting.

K: And the capitals typeface on there is Lucida Grande, so that keepsake actually has a font mixed in with the calligraphy.

D: That's the red?

K: Yes, that's the red.

D: A question back, I guess this design [below] was for Hell?

Why did they want that?

K: They didn't know they wanted it. Hell was going to have a meeting in Basel where they wanted me to show them the beginnings of the redrawings of Caslon and Baskerville, and then they wanted me to make proposals for two totally original typefaces that would show off the resolution abilities of the DigiSet. I went over to the Houghton Library in Cambridge and I was looking at some manuscripts, and there was a manuscript that had a lower case "p" that [gesturing] came down like this, and then it just sort of looped back up like this. All done in one stroke so that the stem was like a double stroke (Figure 10).

Figure 10. The original sketch Kris did in the Houghton Library.

And I thought gee, what if I designed a font where every character worked that way. That would really show off their abilities with the typesetter. So I did it. I just sketched it out and put together a keyword. And I did the finished art by cutting the characters in Amberlith, because I didn't have any other way to proof the characters at that time. Now, you just type it on the screen, but I couldn't do that at that time. So I cut the characters out of Amberlith, and I put together a presentation (Figure 9) that was the beginning of the typeface Isadora. The other typeface I presented was based on the Chinookan calligraphy, and it was called Sierra, and I put it together the same way.

By that time, we had already taken Hermann Zapf's class at RIT and I had worked on an article for *Fine Print* about Zapf Chancery.[10]

So we went to the type review meeting in Basel and, of course, I was just shaking with nervousness. There at the type review meeting was Hermann Zapf, Max Caflisch, and a couple of other people; and we sit down, and so I said, "this is my proposal for Isadora." Hermann Zapf said, "First of all, Kris, I'd like to thank you for that beautiful article in *Fine Print*. You know more about that typeface [Zapf Chancery] than I do." I could tell that one of the other people in the group was not crazy about my proposal for Isadora, so I was really getting nervous; I thought they're going to turn Isadora down. But then, Hermann Zapf picked it up, looked at it, and said "Yes, this is top quality, we'll take it." And that was it. I was on my way thanks to Hermann Zapf. I don't think I would've gotten the commission otherwise, but he was so influential that the other people in the meeting just went along with the idea, and he also said that he liked Sierra, so I did that one as well.

D: Wow!

K: Yeah, I know. [Laughs.] Isn't that a cool story?

D: Great story, but I need you to explain what a keyword is, and what is Amberlith?

K: When type designers design a font, they don't start with A and go straight through to Z. The first step is to design an "n" and an "o". Once you have those two letters looking the way you want, you have a lot of information — x-height, stem weight, round weight, serif structure, bracket structure. So in designing Isadora, I first designed an "n" and an "o". The next step is to design a keyword — meaning a word that contains letters that give you more information about the shapes of all the letters (Figure 11). You choose a keyword that contains a cap and a lowercase (to determine how the stem weights need to differ), an ascender and descender (to fill out your vertical parameters), a diagonal (to determine the weight for that) and "two story" letters like "e" and "a". I always say that once you have a keyword that really works, your design is at least 75 percent complete.

Isadora started with drawings, as do all of my typefaces. But how to take those big drawings and see what they will look like at reading size. In those days for me, the most efficient way to do that was to put a piece of Amberlith over the drawing and cut the image in it. Amberlith is clear acetate coated with a photo-opaque translucent film. I did thousands of these Amberlith cuttings in my early years as a designer. Once you have a good Amberlith cutting, you can simply photocopy it into a black and white image and paste letters together. You can then hang the big images on the wall for a first look and then have them photo-reduced down to see your design in a size closer to the size it might be used at.

Figure 11. Presentation of characters cut in Amberlith.

As you might imagine, this was a tedious process. But over the years I got very good at cutting those Amberliths — people often thought they were cut by some kind of machine. It was really the only way to proof my keywords at that point. In designing Isadora for Hell in Germany, I would do the above keyword process and then send the final keyword drawings off to Germany to be digitized by their crew. They would mail the bitmap images back to me, and then I could go forward and design the whole typeface. So a keyword can be a way to either illustrate a completed typeface, or, more importantly for me, a way to test my ideas.

D: A big part of your world has been Lucida. There is so much; we can point to all kinds of articles and examples.[11,13] But is there something you'd like to say about it? I'd certainly like to hear why you chose the name.

K: Chuck chose the name Lucida because the typeface would be made of light. I think the guiding light of Lucida is that we wanted to design something that was legible, that's always our number one priority. Every time we look at a letter, is it really legible? We never skimp on that. Gary Munch, a former student of Chuck's and now a respected type designer and now dear friend, described Lucida to me as a workhorse typeface. I thought, I like that; it is a workhorse typeface and that is what we wanted. We didn't want something too fancy; we didn't want something unapproachable; we wanted something based on traditional pen-drawn letterforms but something that was modern and clear.

D: Do you ever just get tired of one more Lucida font or typeface? Or does it remain interesting? You get to choose what you do so presumably you choose it because you want to do it.

K: There's always a new challenge, like variation. We have Lucida Handwriting, for which the basic height and weight measurements match Lucida, but in every other way it's totally different. I don't really get tired of it, and it's very handy to always have this basic set of measurements so I know what I'm dealing with — even if I don't always match it, at least I have a starting point.

D: Did or do the issues of the "font wars" — all the different type formats, . . . , Type 1 and Type 3, and OpenType, and all of that — affect you when you're designing a font or is that some kind of a post processing problem?

K: I don't think it affects me one bit when I'm designing a font. You know when I'm designing a new font, I actually kind of live in isolation. I never look

at other typefaces because I don't want them to get in my mind's eye. So even just looking at that lettering over there, it's kind of getting into my mental visual space. So when I'm working on a new design, I don't look at other people's designs. But I'm always thinking. Chuck once asked Adrian Frutiger [designer of Univers and many other typefaces], what do you think about when you're doing a new design? And Adrian Frutiger said the most perfect thing, "I think about what it will look like in the mind of the reader." I'm thinking about that. I'm thinking about how it's going to work with the technology I'm designing for. I'm thinking what it's going to be like to look at. But the business end of things, *pfft*, absolutely not thinking of it at all. How could you?

D: I don't know, I don't do type design.

K: [Laughing.] Well there's so many twists and turns in the business end of things. Really good products get dropped; really bad products get not dropped; and so you just can't think of that.

D: In our correspondence a few years ago, you noted that you and Chuck are one of the few design teams that have worked with both phototypesetting and digital.

K: Yes.

D: Is the design process different for any of that, or is it still you sit there with your paper in isolation?

K: [Laughs.] Well, I think it's different in that I'm thinking how the technology will affect the finished image because, you know, the way something looks printed in phototype is going to be very different from something printed digitally. And we have not only worked in photo and digital type, we did that first typeface Leviathan for metal type and letterpress printing.

D: So that does have to affect what you draw.

K: It does have to affect what I'm doing. But the basic design process is sitting there in isolation.

D: Since Ikarus, what other design systems have you used?

K: I want to say that I am very sorry to have had to give up Ikarus. I used it for I think about 27 years, during which time I think I had a total of two crashes. It was very, very stable; and it had a very, very high resolution. Everybody kept saying, "oh, you need to move on to Fontographer" or something. But I was very happy with the accuracy that I could achieve with Ikarus and the solidity of the system. So I was sorry to give it up. There was an Ikarus on Mac for a while and we used that. But once we got into the big Unicode character sets we really had to move to FontLab, and now maybe we're moving on to Glyphs.

D: Glyphs, that's the name of a system?

K: Yes, G-L-Y-P-H-S. It's a font design and editing application that I think many type designers are moving to now, it's a really excellent system. I'm just always in the middle of a job so it's hard to pull up stakes and learn something new, but at some point I'll get around to it.

D: Let's go to another piece of this literature that you gave me last night, which is this brochure from Imagen, "Imagen Presents Lucida: the First Typeface Design for Laser Printers."[14] Somewhere in something Chuck told me he said that Michael Sheridan designed and produced this. Am I right? This was produced by Michael Sheridan?

K: Yes, that's right. He was Director of Typography for Imagen and previously had worked for Grant Dahlstrom at the Castle Press, which produced finely printed books.

D: And your role in it was . . . ?

K: Was to design the font. I didn't do any of the book design, or production, or anything.

D: Okay. In that year, it talks about 11 different sizes of fonts; 6, 7, 8, 9, 10, 11, 12, 14, 18, 24, and a Roman that's something bigger than that, 36. They weren't scalable fonts back then or what?

K: It sounds like they were still rasterizing the fonts, doesn't it? You'll have to ask Chuck about that detail.

> Chuck explained: At the time, mid-1984, Imagen used bitmap font technology. No laser printers had scalable outline font technology until the Apple LaserWriter with PostScript outline fonts was launched in March 1985.
>
> At B&H, we drew the Lucida characters at a large size, around 166 mm, and digitized them as scalable outlines with Peter Karow's Ikarus system, which used Hermite cubic curves. We designed only one master size, intended to work well at around 10 point, plus or minus a few points. For instance, at 10 point at 300 dots per inch, the vertical stems are almost exactly 4 pixels thick.
>
> From Ikarus, we output scalable outlines in a circular arc & vector format. Again, just one master design. Imagen scan-converted our Ikarus arc/vector outlines to bitmaps, which they hand-edited. In a way, then, the original Imagen Lucida fonts were hand adjusted for each print size, because the bitmap

editors made visual decisions about details of pixel placement as they edited the fonts.

D: It also says in here, "this is for low and modest resolution output devices". So how have things changed for Lucida now that we have higher resolution output devices, on practically everybody's desktop?

K: It means that we can design fonts a little differently. If you look at the original Lucida in that booklet, you'll see that the hairlines are quite thick compared to the stem weight. So, let's see, the stems were 16 mm on a drawing size of 166 mm; the hairlines were like half, 8 mm — something like that — of the stem weight. And that's because you're working at low resolution, so you don't want that hairline to fall apart. You've seen characters where they're just breaking up. We didn't want that to happen, so we made nice thick serifs, nice thick hairlines. For a higher resolution machine, you can make very thin delicate hairlines, and very thin delicate serifs and the whole face will have a slightly lighter look to it. So the different resolution really does change the design.

D: Have you gone back to any of the earlier Lucida fonts or typefaces and redone them?

K: Well that's kind of what Lucida Bright is; Lucida Bright is a redrawing of the original Lucida, but for high resolution. It has rounded bracketing on serifs; thinner hairlines, thinner serifs. So I would say Lucida Bright is a redoing of Lucida. I ran into a girl at Wells College, which has a very nice book arts program, and she had printed letterpress a whole book in Lucida Fax. And I said, "gee, we never thought it'd be printed letterpress." And she said, "oh, it was just perfect for what I wanted to do, so I just had polymer plates made and printed the whole thing in Fax." You never know how your typeface is going to be used, and this is something that bothers a lot of type designers — "Oh gee, they're spacing it too tightly" or "they've taken the italic and slanted it even more." But it never really bothers me. I feel like, "okay, you bought your font; if you want to change the styling that's fine." I don't approve when people go back in and actually change the individual characters, which happens a lot.

D: I guess I don't understand the distinction. How do they change the font to, as you say, slant it more? What do they do?

K: Using Word styling, you can slant an alphabet, so they start with an italic and then think, "I need to slant it more." That's something you do within the word processing.

D: At a high level, where it just does the same thing to everything.

K: Exactly. I see it all the time, and it is a choice made by the layout designer. What I object to are changes made to individual characters, sometimes called glyphs, right in the source code of the font, especially if someone calls it a new design.

D: Have you seen where typefaces that you have designed have influenced other peoples' typefaces, other than messing with your typefaces?

K: [Laughs.] I think so. I think there are many humanist san serifs that were probably influenced by Lucida. I think Lucida's big x-height has been a big influence. You see typefaces with a big x-height all the time. I think our idea of including non-Latin alphabets in with the Latin alphabet — a lot of people have worked in that direction but we were the first designers to coordinate Latin and non-Latin to the extent that we did. We wrote a paper about it, and a recent scholarly paper talks about that, 23 years later. I do think we've been influential.

D: I think you've answered this already but let me ask it explicitly. You have your calligraphy skills, and you do your type design work, and presumably calligraphy influences type design?

K: Oh absolutely. That's always where we start. It's this tool that influences it.

D: The brush?

K: No, it's not even the brush; that's why I brought all of these things (Figure 12).

Figure 12. Four calligraphy tools. See `tug.org/interviews/holmes-4-tools.mp4` for a video.

Here we have a brush [rightmost] but the thing that's distinctive is this broad edge; it's called a broad edge tool. This is a marker [second from right], and the thing that sets it apart from the marker that you're writing with is it's not a point, it's got a

broad edge. Here's a really big marker [second from left]; same thing, the broad edge. Here's a pencil [leftmost]; same thing, the broad edge. So it's this broad edge that is really the basis for our way of thinking about type, and it's because now you're writing not with the point like you are with your marker, you're writing with a line so it's almost like a 3D quality in the character. The tool automatically creates the thicks and the thins in the letter. I almost feel like this tool does 50 percent of the work for me, and then my hand just kind of automatically pushes it around based on my study of historical calligraphy. That's my starting point, then I have a nice, calligraphed — to use that word Reynolds hated — nice calligraphed letter.

Whenever Chuck and I discuss lettering between ourselves, we always end up getting out one of these pens and say, "if you go like this and like this, that's how it needs to work right there." And every decision big or little is based on this tool. When I say "based" on, I mean that we go a long way from this tool and sometimes we use a slightly different tool. For example, Lucida Handwriting, that was based on some sketches I did with a really old marker, a pointed marker, but it was wearing out, so it was almost like half brush, half pen. So sometimes we move to a different tool and certainly go far afield from this analysis, but that's really the root, right there.

D: Forgive me for not completely grasping, but let's say you're designing something; you draw a letter; then do you somehow digitize that? Or are you mostly working on a screen rather than with a brush?

K: The way I work is that I start with my sketches and my rough things with a brush or with a pen; and then I put a piece of paper on top of that, and I've done it to the size that I want or I blow it up to work it to the drawing size that I want, about 166 mm tall. Then put a blank piece of paper on top, put down some guidelines, and then I do an outline around that sketch, or around that calligraphic letter. I have an illustration that shows this perfectly (Figure 13). And this is where I get to go far afield. So when I make a serif with this brush, well it's a very delicate little thing — you can see up there [on a wall poster] on that first "R" you can barely see the serif — but for type, you don't want a serif that thin, it would break off. So that's where I get to make some major changes with my pencil and I just beef up that hairline a little bit, beef up the serif.

D: And somehow that gets into a computer.

K: Yes it does. When I was using the Ikarus system we actually had a little Aristograph tablet, a little electronic board with a puck on it and we would digitize that way. Now what I do is that I scan the drawing into a background and I fit splines on it on a screen (Figure 14).

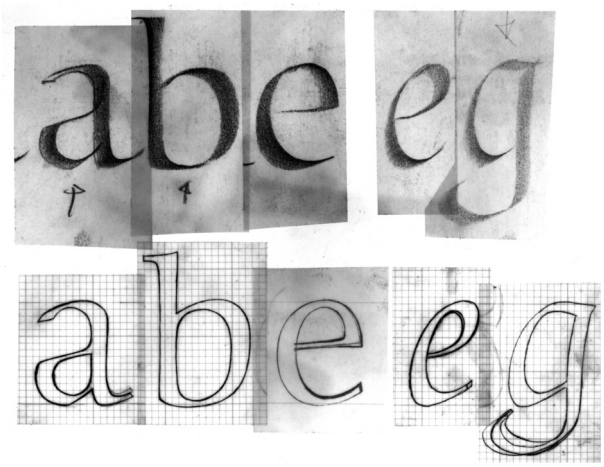


Figure 13. Illustration of the process. The typeface is Sierra — based upon her calligraphic studies for the Syntax Phonetic project. The tool she used was a broad edged pencil (leftmost in Figure 10), a carpenter's pencil.

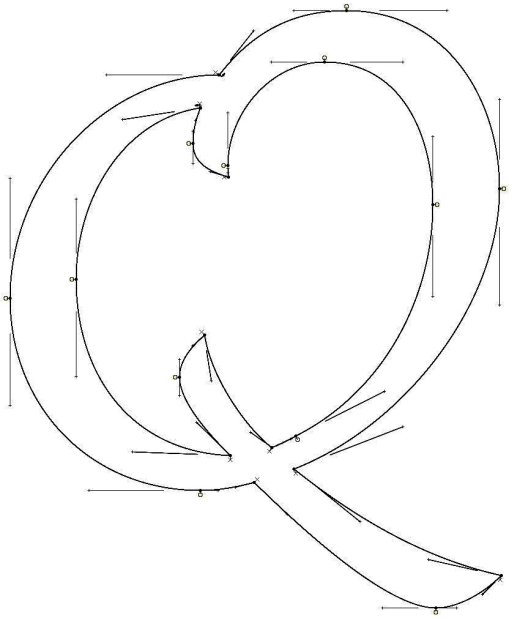


Figure 14. The letter Q from Lucida Handwriting showing chosen spline points.

D: So this is in Glyphs?

K: I've actually been doing this part of the job in Illustrator. You can do it in FontLab or Glyphs or related applications, but I kind of like working in Illustrator for the first pass because you get a really nice background image in that. When I say fitting splines,

I mean that type designers have a very specific way that we lay out the spline outlines on a character. A designer can take a letter and just autofit splines in Illustrator, and that's fine if you're doing a lettering job. But autofitting gives you way too many points and they are positioned illogically, not recognizing the structure of the characters. But for something that we do, which may be a system font, it's going to have hints put on it that adjust the outlines to fit raster grids; it's going to be used at lots of different sizes, so we have a very specific way that we arrange those points so that shapes and the structure are logical. All the extreme points are marked, in the X and Y axes, and the stems and bowls, x-heights, base lines, capital heights, and so on, are marked, so when you put hints on them, the rasterizer can adjust those to the output raster easily.

D: How does what you have in Illustrator get into Glyphs?

K: Okay. So now I have the calligraphic sketch, and then I do the real fine line drawing on a piece of Mylar, then I scan that, put it in the background in Illustrator, I fit the splines in Illustrator, so I have a nice spline-based outline. I just simply copy and paste that into a window in FontLab or Glyphs.

D: Ah ha!

K: And if I set everything up right, to all the exact measurements, it all goes very smoothly.

D: How long does it take you to do a letter?

K: I don't know, I don't even want to know. When I was doing really fast jobs for somebody, I would time it based on half an hour a letter. But you know, I don't know anymore because sometimes you spend … like a "w", which is always so difficult to do; I might spend two or three hours on a "w". But you know, a capital "I", well, not so much. And sometimes it takes years to finish a typeface just the way I want it.

D: Chuck talks about harmonized families of Lucida; serif, sans serif, and typewriter. What does harmonized mean?

K: I can explain that. I have a little lecture that I give when I teach type design. For a type design to be effective and legible, you have to achieve a balance of sameness and differentness. Think of a ransom note; really hard to read, right? It takes you a long time to struggle through it. Or think of plain block lettering where you have a square and every letter fits into the square; also very, very difficult to read. So on the one hand, the letters are too different in the ransom note; and too similar in plain block lettering. It's not as bad as a ransom note, but you wouldn't want to read a whole book like that because the letters are too much alike. So when you design typefaces, you're not just designing a beautiful "a" and then a beautiful "b" and so on, you're designing a whole system of characters that work together — that are different enough so that you can tell an "o" from an "e" but are alike enough so that your reading experience is smooth.

Then between different weights and styles of a typeface, it means that you have to have something that's holding those together. Maybe the stem weights are matching. Maybe the slant of the italic is matching. Maybe the thick/thin ratio is matching. But you want certain aspects that are matching so that your eye can go smoothly from, say, a sans serif to a script font without that horrible jarring effect that you get from the ransom note. So that's harmony. Does that make sense?

D: It does make sense, and I can see it in my mind's eye that reading a book where the titles are in some different font from the text, if it's too different, it's jarring.

K: Yes, you've seen that, I'm sure.

6 TUG, teaching, conclusion

D: Since this is an interview for the TEX Users Group, I need to ask you something about TEX, LATEX, Metafont, Computer Modern, and the TEX community more widely. How do you see any of that and its relevance to the world at large, or its relevance to you?

K: Well, to be frank, I don't know that much; I couldn't design a metafont. I greatly admire it because I think it opened up this whole world of typesetting mathematics, which I wouldn't have even understood was a problem, except mathematicians have told me how it used to be and that it was a problem. I will tell you that at some point, back in the early 1980s, when we had a studio in San Francisco on Vandewater Street, I picked up a copy of Knuth's — I guess it's his Metafont book — and I stayed up all night reading it. I was just completely enchanted that you could … the way he lays everything out. You just start from the beginning and you just move through. I thought "wow, this is great; I'm going to start designing everything as a metafont now." But then I found out that actually it's a lot more complicated than using Ikarus so I never followed up on it, but I'm enchanted by the idea. I think Knuth has done a lot of good with this system. It is just one of those things I'm probably not going to get around to in this lifetime. [Laughs.]

D: Why is Bigelow & Holmes bothering with a relationship with TUG? TUG is such a minor entity.

K: 'Cause we love you guys. [Laughs.] No, it's really true; because we like and respect you. We've always enjoyed working with Karl Berry over many years, and the same with others in TUG. Why work with people if you think it's going to be a problem? With you guys, we know it's not going to be a problem. And if there's a problem, we can work it out. You're all such interesting people. I mean, this is why we're independent. You know, I don't think Chuck or I, either of us, would want a job that was unhappy. We want to be really happy in what we do, and part of being happy is hanging around with people where we enjoy their company. Not only just on a professional level, but it seems we all have personal things that we have in common. I guess it's like the big group of friends that Chuck and I were part of when we first met. You're part of our big group of friends that we have as adults. And you do a good job.

D: Well, we could go with this interview for a long time, and we can add to it as I sort this out, and you say you have some other materials. That would be great.

K: Yes, I have illustrations.

D: Before we stop for today, I would like to ask a few final questions. For instance, you teach; you've taught a lot. Why do you teach?

K: I've taught a lot. But I don't actually teach anymore.

D: Why did you teach?

K: I felt it was my duty. You know, I had good teachers and people who passed on to me this amazing knowledge that I could never have gotten from a book when I started out, or now. And so I felt that it's my duty to pass on what I've learned; and also, when you're teaching, you also learn from your students. One thing I've learned is to not be so uptight about type design. [Laughs.] You know, I was teaching type design here at RIT and the first year I thought, "Oh wow, they're not doing everything [right], they're not restricted enough." I had taken it too far on a perfectionist side, but I could loosen up a little bit like them, and that was a very good feeling.

D: Have you also mentored people who thought about going into type design?

K: Yes, and we've mentored people who did go into type design.

D: What do you recommend for them? How do you tell them to do this?

K: I think in the past, we've mostly told them by example. We teach them how to do the craft, and we just conduct our lives as we do, and they learn. Actually, at this point, I wouldn't really recommend that anybody go into type design exclusively because the business model at this point is just so . . . ; it's not really something that you could make a living out of very easily; it would be a very hard living. So these days more often I recommend that people become general designers, maybe specializing in lettering design, and specializing in font design. I know that when I was teaching type design here, a lot of my students were graphic designers and they would write to me and say, "oh, I put my typeface into my portfolio, and when I pulled that out, 'Wow!', The recruiter was so excited, they said, 'you designed a typeface?! I don't know how to do that!' " So it was a nice addition to their overall portfolio.

I talked to Hermann Zapf years ago at a meeting, and he said he doesn't teach calligraphy and lettering any more. He said, "If I ever taught again, I would teach craftspeople like bakers to do nice lettering on cakes, or woodworkers to do good lettering." He said that he was disillusioned with the typeface design business.

D: What should an amateur typesetter, who may not know much about type, but uses what's there, tries to put a hyphen in the right place, tries to remember when you're supposed to use slanted instead of emphasis or italic. What do you recommend for such a person; what's the minimum they should know?

K: That's a good question. I think you should know the difference between roman, italic, oblique. You'd be amazed how many people don't know the difference between italic and oblique.

D: I'm one of them.

K: You know the difference. Italic is a different letter form, while oblique is just a slanted roman. You know it, you just don't know that you know it. [*Lucida italic* vs. *Lucida oblique*.]

I think you should also know about the different slants; you should know about weights, what's a normal weight, what's a bold weight, what's a light weight. You should know some basic styles: this is an old style, this is a sans serif, this is a slab serif; and you could buy a simple book about typography. There are several now. Alexander Lawson wrote a good book about type that explains all of these things, the title is *Anatomy of a Typeface*. Jan Tschichold's *Asymmetric Typography* is another great book that kept me up all night reading.

And then I think that you should just look at things and read things, and just ask yourself questions about what I'm reading. There's your handwriting on that page; you laid that handwriting out in a certain way for a reason. You want it all on parallel lines so it's easier to read, you made it a certain size in relation to the page. If you were going to translate that into a typeset piece, what would you do? Well, maybe you'd make the letters a little smaller because type is easier to read than handwriting. You'd have to look at the space between lines. Just think about what you're doing and what it means. What kind of feeling does flush left give as opposed to a centered piece of type or a justified column (flush left and right). I think just thinking about what you are saying is a good start.

Also finding something that you really want to do a good job on. You know, I think that Chinookan calligraphy really turned me around as a designer because I wanted to do a good job. So I worked really hard and thought it all through from the very basics, and I invented that new script that I was using, because it really meant a lot to me. So I think that rather than just doing alphabet after alphabet or something, just find literature that you love and typeset it, and see how it feels. It'll feel great.

D: I will try your advice.

D: I have to go back to a prior question I forgot to ask. Last night at dinner, you and Chuck were talking about the Go typeface (Figure 15).[15] It's a free typeface apparently.

K: That's correct. Free and open source.

```
return func() *DenseMatrix {
return func() *DenseMatrix {
return func() *DenseMatrix {
return func() *DenseMatrix {
```

Figure 15. Go font examples: (top to bottom) Go mono, Go regular, Go medium, Go medium with color.

D: But that presumably means that somebody paid you to do it so it could be free.

K: It was a commissioned typeface, by the Go language people at Google. They said that they wanted a really nice font to bundle with the Go language or "Golang". One of them was Rob Pike, who co-invented the Plan 9 from Bell Labs operating system, which used Lucida fonts. But for Go, which is free and open source and has good handling of TrueType fonts, they wanted something nice for people to use that could also be free and open source.

D: Final question: what do you see in your future?

K: I think I'd like to just be able to continue what I have been doing all along. Designing for Bigelow & Holmes. Honoring my wonderful, generous teachers. Encouraging sincere young designers. And putting a little swing into things. [Laughs]

D: Thank you so much for taking the time to participate in this interview. I'm honored to meet you in person.

K: It's my pleasure.

Notes and references

[1] See the article on the Cary Graphic Arts Collection elsewhere in this issue (pp. 169–170).

[2] Yue Wang, Interview with Charles Bigelow, *TUGboat*, volume 34, number 2, 2013, pp. 136–167, tug.org/TUGboat/tb34-2/tb107bigelow-wang.pdf

[3] For more about Reynolds' teaching and legacy, see reed.edu/calligraphy/history.html

[4] Laura Lindquist, The Music of the Words, *Reed Magazine*, August 2003, reed.edu/reed_magazine/aug2003/features/music_of_words/index.html

[5] Robert J. Palladino, Inscribed in Stone — The Masterful Work of Edward M. Catich, *Reed Magazine*, March 2010, tinyurl.com/reed-magazine-2010-catich

[6] artlegacyleague.blogspot.com/p/about-catich.html

[7] Also see the interview of Charles Bigelow in *IEEE Annals of the History of Computing*, vol. 40, no. 3, pp. 95–103.

[8] The image in Figure 2 was cropped from an image that was 50 percent wider and somewhat taller on a 5.5x9 inch page of the keepsake brochure. Other images in this interview have also been cropped from their original presentations.

[9] See library.rit.edu/cary/exhibitions/leviathan-typeface-surfaces-after-40-years which shows the letter C from the font, used in "Call me Ishmael".

[10] Kris Holmes, ITC Zapf Chancery, *Fine Print*, January 1980, pp. 26–29, scroll down to the last part at tinyurl.com/holmes-chancery.

[11] Charles Bigelow and Kris Holmes, The Design of a Unicode Font, *Electronic Publishing*, volume 60, number 3, September 1993, pp. 289–305, cajun.cs.nott.ac.uk/compsci/epo/papers/volume6/issue3/bigelow.pdf

[12] There are many example fonts at luc.devroye.org/fonts-26292.html

[13] Charles Bigelow, A Short History of the Lucida Math Fonts, *TUGboat*, volume 37, number 2, 2016, pp. 154–160, tug.org/TUGboat/tb37-2/tb116bigelow-lucidamath.pdf

[14] tug.org/interviews/holmes-imagen-lucida.pdf

[15] Go mono and Go regular also come in italic, bold, and bold italic; Go medium also comes in italic. For more about the Go fonts, see tug.org/interviews/holmes-bigelow-gofonts.pdf. The code sample in the figure was provided by the Go language developers.

◇ David Walden
tug.org/interviews

[*Editor's note:* Many of the illustrations use color; for the full effect, please see the online version of the article at tug.org/TUGboat/tb39-3/tb123holmes-walden.pdf.]

Science and history behind the design of Lucida

Charles Bigelow & Kris Holmes

1 Introduction

When desktop publishing was new and Lucida the first type family created expressly for medium and low-resolution digital rendering on computer screens and laser printers, we discussed the main design decisions we made in adapting typeface features to digital technology (Bigelow & Holmes, 1986).

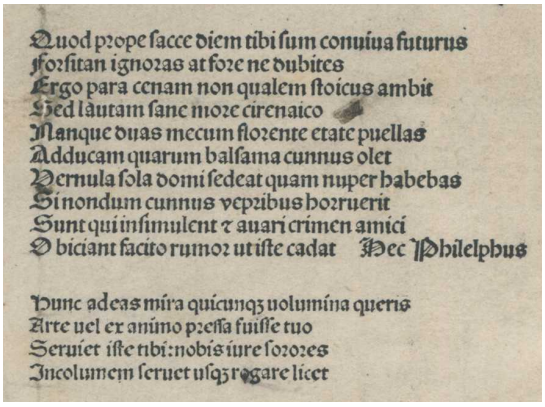
Since then, and especially since the turn of the 21st century, digital type technology has aided the study of reading and legibility by facilitating the development and display of typefaces for psychological and psychophysical investigations. When we designed Lucida in the early 1980s, we consulted scientific studies of reading and vision, so in light of renewed interest in the field, it may be useful to say more about how they influenced our design thinking.

The application of vision science to legibility analysis has long been an aspect of reading research. Two of the earliest and most prominent reading researchers, Émile Javal in France and Edmund Burke Huey in the US, expressed optimism that scientific study of reading would improve the legibility and economy of written and typographic forms.

> "The object [of study] is the characters in use. We shall have to investigate their size, their form, their spacing." (Javal, 1878)
>
> "We therefore should seek to improve legibility without reducing the number of letters on the page." (Javal, 1905)
>
> "Certainly the letter-forms that have come down to us through the ages have never been pruned to meet the reader's needs, though the writer and printer have made conservative changes for their own convenience. There is not the slightest doubt that forms can be devised which will be much more legible than these ancient traditional symbols." (Huey, 1908).

A few years later, Barbara Roethlein, in her M.A. thesis at Clark University, formalized the questions to be asked of typographic legibility:

> "Every reader has observed that all of these variants of letter-forms are not equally legible — an observation which raises the theoretical question: What are the factors upon which legibility depends? And the practical question: How should one proceed if one set out to improve the legibility of printed letters?" (Roethlein, 1912)


Quod prope facce diem tibi fum conuiua futurus
Fortitan ignoras at fore ne dubites
Ergo para cenam non qualem ftoicus ambit
Sed lautam fanc more cirenaico
Nanque duas mecum florente etate puellas
Adducam quarum balfama cunnus olet
Vernula fola domi fedeat quam nuper habebas
Si nondum cunnus vepribus horruerit
Sunt qui infimulent t auari crimen amici
O biciant facito rumor ut iſte cadat Hec Philelphus

Nunc adeas mira quicunq; uolumina queris
Arte uel ex animo preſſa fuiſſe tuo
Seruiet iſte tibi: nobis iure forores
Incolumem ſeruet uſq; rogare licet

Figure 1: Earliest known type specimen sheet (detail), Erhard Ratdolt, 1486. Both paragraphs are set at approximately 9 pt, but the font in the upper one has a larger x-height and therefore looks bigger. (See text.)

Despite such early optimism, 20th century type designers and manufacturers continued to create type forms more by art and craft than by scientific research. Definitions and measures of "legibility" often proved recalcitrant, and the printing and typographic industries continued for the most part to rely upon craft lore and traditional type aesthetics. Moreover, the craft of type punch-cutting involved visual knowledge that vision science had not yet encompassed. For five centuries, type punch-cutters — type designers before the term — carved extremely tiny forms that had to be effortlessly recognizable by the greatest number of readers, and as well be visually pleasing to the casual glance. Renowned punch-cutters like Garamond, Granjon, Van Dijck, Bodoni, and others, though not scientists in the modern sense, were cognizant of some of the most refined aspects of visual perception.

2 Body size and x-height

The most universal feature of type is size. The ability to compose type at nearly any size is taken for granted today, but not in the early years of typography. The laborious creation of many sizes of type, the punches cut by hand, was the life work of highly skilled artisans over generations and centuries.

In the incunabula era, printing through 1500, very early books were printed in single sizes and styles of type, but later printers did employ a broader range of sizes. In 1486, Erhard Ratdolt, a German printer established in Venice, printed the first known type specimen sheet, showing 14 different typefaces (fig. 1). Ten were gothic rotunda fonts in sizes from 36 to 9 point (Ratdolt, 1486).[1]

[1] These measurements in point sizes are rounded to integers. The actual body sizes are a few fractions of points bigger or

A remarkable feature of Ratdolt's range of rotunda fonts is that at three of the sizes, 18, 13, and 9 point, he displayed two versions, one with a large x-height relative to the body, and one with a small x-height. Ratdolt's larger x-height versions look substantially bigger to us than the smaller x-height versions at the same body size. Ratdolt left no explanation, but we may reasonably suppose that visible differences between the different x-height versions appeared the same to the printer and his readers in the 15th century as they do to us today. A side observation is that Ratdolt's gothic fonts, as with most gothic types of the era, had larger x-heights than the roman types produced by printers in Italy at the time, yet the roman style soon replaced the gothic in Italian printing, and thence proceeded to do the same in French and eventually English and Dutch printing.

During the 16th century, average type sizes in use decreased by a few points. The main economic factor was cost of paper. Smaller type sizes enabled smaller page sizes, less paper, cheaper editions, and a larger market. Other factors have been suggested. One is greater production and usage of eyeglasses, to make smaller type more legible for older readers or others with vision difficulties. Another is technical improvement in the methods of punch-cutting and type casting, including improvements in metallurgy to produce harder, more durable type. A religious reason may also have been a factor during the Protestant Reformation and the Counter-Reformation: smaller type enabled books to be more economically printed and transported, and, if the contents were proscribed by religious authorities, easier to conceal.

But, as roman type body sizes decreased in the 16th and 17th centuries, their x-heights increased. For example, in 1569, Robert Granjon cut a "Gros Cicero" ("Big 12 point") in the style of Garamond but with a bigger x-height that made it look almost as big as the next larger body size, the St. Augustine (14 point) (Vervliet, 2010; M. Carter, 1985). In the 17th century, Dutch punch-cutters and typefounders continued the trend toward bigger x-heights, and in the 18th century, Pierre Simon Fournier cut alternative faces with large, medium, or small x-heights, in several sizes. He called certain of his large x-height, slightly narrow faces "in the Dutch style" (Fournier, 1766).

It has been said that Granjon's Gros Cicero was the eventual model for the Monotype face "Plantin" of 1913, which may have been the starting point for Times New Roman of 1931 (Carter, 1985).

Commenting on the longstanding trend to larger x-heights, Stanley Morison lamented the lack of documentation on the "development of type design consciously viewed as a means of reducing the real space occupied by the letters while maintaining their apparent size" (Morison, 1968).

In an influential essay on the "optical scale" in typefounding, Harry Carter (1937) pointed out that types intended for different reading sizes were traditionally designed differently. In particular, types for newspapers and other continuous texts composed at small sizes often had abbreviated descending strokes or "tails", as well as shortened ascending strokes, to increase the x-height fraction in relation to the body size. By "x-height fraction" we mean the portion of the total body height occupied by the x-height.

Although it had been evident for 500 years that larger x-height fractions made type appear bigger, in the 1980s we did not know of studies that proved that types with bigger x-height fractions were actually more legible in terms of speed of reading or degree of comprehension. Apparently, when type looked bigger, that was good enough to persuade printers and readers of its value, but several early 20th century legibility studies focused on the minimum sizes that were easily readable.

Javal (1905) stated that nine point type was most used for books and newspapers in France; the 9-point type in his book had an x-height of 1.5 millimeters. Huey (1908) recommended a minimum x-height of 1.5 mm for fast reading. Roethlein (1912) tested 10-point fonts, the majority of which had x-heights in the range of 1.4 to 1.5 mm (insofar as the heights could be determined).

Miles Tinker's *Legibility of Print* (1963) summarized decades of meticulous legibility research by Tinker and Donald Paterson on type size and legibility. Using body size in points as their measure, they found that type sizes of 10 and 11 point were read most quickly. By the early 1980s, many of the types tested by Tinker and Paterson a half-century earlier were no longer in common usage, but our measurements of those types and sizes in catalogs indicated that the x-heights averaged 1.5 mm.

Those early assertions of minimum type size for fluent reading were confirmed in a series of rigorous psychophysical reading studies by Legge et al. (1985, 2007), which found the "critical print size" below which reading speeds decrease markedly, but above which increases in type size do not appreciably increase reading speeds. Legge et al. measured

smaller. Typographic point systems were not promulgated until the 18th century and not stabilized until the 19th and 20th centuries (Ovink, 1979).

RQENbaegn (Lucida Bright)
RQENbaegn (Nimbus Roman)
RQENbaegn (Nimbus Mono)

Figure 2: Lucida, URW Nimbus Roman (Times design), and URW Nimbus Mono (Courier design) compared at 10 point.

the physical size of printed type and the distance at which it is read, defining psychophysical size as degree of visual angle subtended by the object — in this case x-height — at the retina of the reader's eye. Legge (2007) states that across a range of studies, the critical print size is around 0.20 degrees. This is equivalent to a 1.4 mm x-height read at 40 centimeters. For example, with familiar Times Roman, critical print size would be 9-point type read at a distance of 16 inches. Greater reading distances need larger type sizes for easy reading; closer distances allow reading at smaller sizes, as when teenagers easily read small text on smart phones at distances of 12 inches or less.

When we designed Lucida in 1983–1984, we were not aware of Legge's studies of print size. We determined Lucida's large x-height fraction by a less rigorous method. Some ergonomic recommendations of the early 1980s specified that 20 to 24 inches was a suitable distance for reading text on a computer monitor, and although there were other recommendations, they all suggested that reading distance for screen displays should be greater than average reading distances for text on paper. When we viewed 10-point printed samples of popular typefaces like Times Roman and Courier at 20 to 24 inches, they seemed too small. A 12-point size seemed easier to read. But, as with paper in Renaissance printing, computer screen area was expensive, so simply enlarging type size on screen was not an ideal solution. We thought it would be better to give Lucida a big x-height, so that when set at a 10-point body size, it would look as big as Times or Courier at 12 point. (This article is set in 9-point Lucida Bright.)

We made the Lucida x-height fraction 53% of the body size. In other words, when Lucida is set at 10 point, its x-height is 5.3 points high. In comparison, the x-height fraction of Times Roman is 45% of body sizes and that of Courier nearly the same. Thus, at 10 point, Lucida appears bigger than Times or Courier, by roughly 17%, although the visual impression is affected by average letter widths. Times on average is narrower than Lucida, and Courier, a monospaced design, is wider than Lucida.

At 10 point, Lucida read at a distance of 16 inches has a visual angle of 0.26 degrees, well above the critical print size found by Legge et al. At a distance of 20 inches, 10-point Lucida has an x-height of 0.21 degrees, still above critical print size.

We wondered about drawbacks to such a big x-height. As ascenders and descenders are decreased in length in order to increase the x-height of the font, there must eventually be a stage at which ascenders and descenders are too short for readers to distinguish letter pairs like b/p, d/q, h/n, v/y. We did not know at what point such illegibility would occur. Some 32 years later, in an elegant study of design proportions and legibility, Larson & Carter (2016) tested different x-height fractions of a single typeface design and found that, indeed, beyond a certain point, reduced descenders impaired letter recognition.

Another reason for our choice of a big x-height was related to digital screen resolution. In the early 1980s, computer screens had resolutions around 72 to 75 pixels per inch, too low at text sizes to render more than a pixelated impression of letter shapes. Before deciding on the final forms and proportions of Lucida high-resolution outline characters, we hand-sketched bitmaps of letters at various resolutions on graph paper, to study how high resolution forms devolve into minimalist pixelations at low resolutions. The x-height portion always seemed more important for letter recognition than the ascenders and descenders; observations going back to Javal and Huey supported that view.

Later, using interactive bitmap editing tools, we produced hand-edited bitmap font sets, named "Pellucida", for screen displays (Bigelow, 1986). These had somewhat larger x-heights than the outline Lucida high-resolution fonts, and were used as user interface fonts on the DEC VAXstation 100, the Tektronix Smalltalk workstation, and in the operating system Plan 9 from Bell Labs.

A side trip to the future: in 2011, one of us (Bigelow) co-authored a review article with Legge, with illustrations of x-height and letterforms created by Kris Holmes (Legge & Bigelow, 2011). That paper reviews reasons in favor of x-height as the main indicator of perceived type size, cites historical, practical, and laboratory evidence to explain the "critical print size" and other aspects of type size in relation to reading. It should be noted, however, that some reading scientists and typographers favor capital height as an indicator of legibility, notably Arditi (1996) and the German DIN 16507–2 standard of 1999.

3 Open spacing

Lucida Sans (including Lucida Grande), and the original Lucida seriffed faces have slightly more space between letters than most modern types. In particular, Lucida Sans has more inter-letter spacing than popular sans-serif typefaces in the "neo-grotesque" style dating from the 1960s and 1970s, e.g., Helvetica and its clones. In the 1970s, there was a fad for very close or "sexy" letterspacing in seriffed as well as sans-serif typefaces intended for advertising typography. This was partly based on a hypothesis that we read by word shapes, not letters, which led to assaults on readers with dense tangles of crowded words. That hypothesis has since been discredited by further research (Pelli et al., 2003).

Lucida took a different approach. Its letterspacing was influenced by the open spacing of early roman typefaces, like Jenson's Venetian romans from 1470 to 1480, which remained legible despite the "noisy" environment of rough paper, easily worn types, and uneven pressures of early printing technology. The spacing of early roman typefaces tended to equalize the apparent space between letters with the space inside letters, an aesthetic practice believed to contribute to legibility, followed by later type punch-cutters and type designers through the 20th century. Equalized spacing was a visual judgment, not an exact measure of distance or area.

Related to the concept of "optical scale", types intended for small sizes often have slightly wider inter-letter spacing, which can be seen in contemporary as well as historical types.

Another influence on Lucida letterspacing was a tremendously influential paper by Campbell & Robson (1968) which showed that the human visual system is more sensitive to certain spatial frequencies — alternating light and dark band patterns — than to others. Peak sensitivity occurs around 3 to 6 cycles per degree of visual angle, and becomes less sensitive as frequencies increase, that is, as the alternating bands are more tightly packed. At higher spatial frequencies, contrast between light and dark stripes must be increased for better perception. Campbell and Robson demonstrated this as a contrast sensitivity function, "CSF" (Ohzawa, 2008).

Type printed on light paper with black ink is generally high-contrast, but type rendered on the phosphors of cathode-ray-tube screens by a soft scanning spot is lower in contrast and fuzzier, so we tried to adjust the horizontal spacing frequency of Lucida characters at 10 and 12 point to fall near the peak visual sensitivity range of 3 to 6 cycles per degree. At 12 point, Lucida Sans has a vertical stem

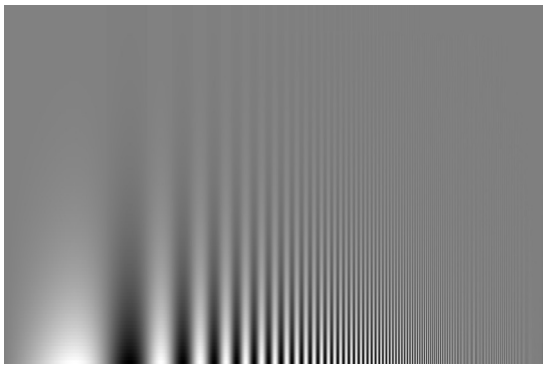


Figure 3: The Campbell-Robson contrast sensitivity function (from Ohzawa, 2008). Perception of the bands typically shifts as you view the image at closer or farther distances or scale the image larger or smaller.

spatial (stem) frequency of roughly 5.5 cycles per degree of visual angle, and at 10 point, the frequency is around 6.7 cycles per degree — not quite ideal with reference to the Campbell & Robson CSF, but reasonably close.

The contrast sensitivity function concerns visual acuity, but a different aspect of letter spacing is the reader's ability to recognize objects, namely letters, that are closely juxtaposed, as in standard typographic text strings. Herman Bouma called this "interaction effects in letter recognition" in Bouma (1970) and "visual interference" in Bouma (1973). Bouma found that the ability to perceive fine details is impaired when contours are close to the details to be recognized. In particular, recognition of letters is impaired when flanking letters are close by, and impairment worsens the farther the letters are from the fixation point of central vision. This effect is now commonly called "crowding" (Pelli et al., 2007; Levi, 2008). Bouma's observations caused us to think that generous spacing could ameliorate some problems in recognizing type on screens. We already believed that the tight letter spacing of popular grotesque faces was a hindrance to reading at small sizes, and Bouma's research tended to reinforce our impressions.

Crowding is the difficulty of recognizing letters near each other. It has two main factors: (a) the closer the letters are to each other; and (b) the farther off-center they are on the retina. (The "center" being the small, high-acuity region called the "fovea".) In reading, our central vision fixates briefly on words or letters and then jumps several letters ahead to fixate again, and so on. During fixation, the more peripheral the letters are — that is, the farther they are from central vision — and the closer they are to

each other, the harder they are to identify. The more crowded the letters, the slower the reading.

Although wider letterspacing may seem to improve recognition of text at a given type size at a certain distance, wider letter spacing also expands the whole text string, driving subsequent letters or characters further toward peripheral vision, where crowding becomes progressively worse.

In the era when we designed Lucida, texts would be read on computer screens at greater distances than on paper, and thus the type would look smaller and its inter-letter spacing tighter. Therefore, we made the spacing slightly wider, hoping it would reduce the risk of crowding and make reading easier despite the greater reading distance.

We also noted that a little extra spacing avoided some localized problems when errors in rasterization and fitting caused adjacent letters to accidentally merge, on screen or in print. This often happened with a popular grotesque sans-serif in early laser printers; the letter 'r' often collided with a following 'n' to make a spurious 'm', turning "fern" into "fem", "warn" into "wam", and so on.

The trade-off of loose letter spacing was that at larger text sizes on paper, Lucida text seemed airy compared to densely fitted grotesques. The higher resolution LCD and LED displays of modern smart phones, tablets, and laptops have made some of these Lucida adjustments unnecessary, but Lucida fonts still perform well on high resolution screens and e-ink readers at small sizes.

Much later research doubted that more space between letters ameliorates crowding. "There is no escape," declared Pelli et al. (2007). So, did generous letter spacing of Lucida make it more legible? Anecdotally, yes. Lucida (Sans) Grande functioned well as the system screen fonts on Macintosh OS X for 13 years at sizes ranging from 9 to 14 point, and users complained when the system fonts were changed from Lucida to a grotesque sans-serif. Monospaced Lucida Console has been a terminal and programming font in Windows operating systems since 1993. But, does generous letter spacing actually improve reading speed or comprehension? Perhaps, at best, only for certain sizes, reading distances, and readers. A recent paper by Xiong et al. (2018) compared legibility of fonts intended to ameliorate effects of macular degeneration and found that interletter spacing was a beneficial factor.

4 Open counter-forms

Counter-forms are the spaces inside letters; some are totally enclosed as in 'b', others non-enclosed as in 'c'. A few letters have both enclosed and non-enclosed counters, as in roman 'a', 'e', and 'g'. Enclosed counters can clog up in printing, so in giving Lucida a large x-height, we also made the enclosed areas relatively big. Wherever possible, we opened up counters in 'a', 'c', 'e', 'g' in the roman styles (fig. 4, left). In the italics, we adopted a chancery cursive style with a characteristic counter-form for *a*, *d*, *g*, *q* in one orientation, and a rotationally contrasting counter in *b* and *p*, to help distinguish letters easily confused. We thought of widening the letters as well, but this would have reduced economy of fitting, because we were also increasing the spaces between letters.

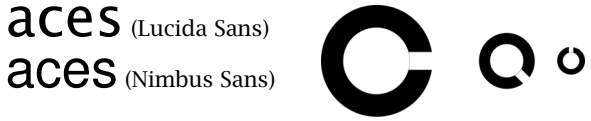


Figure 4: counter-form comparisons of Lucida and URW Nimbus Sans (Helvetica design), left; Landolt C, right.

The nearly enclosed counter-forms of 'c' and 'e' in "grotesque" style faces, while stylish at big sizes, appeared to close up the gap (also called "channel" or "aperture") separating the two terminals of 'c', and the eye and lower terminal of 'e', which tend to get blurred, get clogged or blurred, making them confusable with 'o'. There is a vision test that uses a circular figure called the "Landolt C", devised by a 19th century Swiss ophthalmologist (fig. 4, right). The Landolt C is a circular ring with a precisely cut gap equal to the thickness of the ring. Test subjects are asked to name the position of the gap but do not need to name letters. It resembles, in a rigidly geometric way, the aesthetic of several Swiss grotesque sans-serifs.

5 Distilled humanist letterforms

Because we were designing Lucida for text sizes and, often, coarse resolutions, we tried to distill the letter shapes to minimalist forms that we felt would be recognizable under most imaging conditions. We wanted Lucida typefaces to be without distracting details, essentially transparent as conveyors of information. Lucida Grande, Lucida Sans, and original Lucida seriffed have forms and thick-thin proportions derived from pen-written letter shapes written and read in the 15th century by Italian humanists, whose handwriting was the model for the first roman typefaces. A fellow typographer commented that Lucida is a "workhorse" design. We took that as a compliment. Lucida true italics are somewhat showier, exhibiting traces of the fast Humanist handwriting styles still called "cursive" or "running".

We had studied Humanist handwriting as students of Lloyd Reynolds and other calligraphy teachers, including Hermann Zapf. The Humanists based their writing on what they thought was the most legible ancient handwriting, written by scribes in the court of Charlemagne 600 years earlier. Early Humanist letterforms were simple and unadorned, crafted to be easy to write and easy to read, even by older scholars with declining vision, in an era when eyeglasses were rare and expensive. The Humanist style was therefore extensively "user tested" in two different historical eras. Of course, nearly all roman types descend from one or another era in the long evolution of type forms that began with Humanist bookhands in the 15th century, but the "humanist" sans-serifs pioneered by English lettering artists Edward Johnston and Eric Gill, and later refined by Swiss designers Hans Ed. Meier and Adrian Frutiger, followed the Renaissance style, not the 19th century English machine-like "grotesques". Influenced by the calligraphic teaching of Lloyd Reynolds and impressed by Meier's elegant Syntax and his persuasive reasoning (Schulz-Anker, 1970) that the humanist forms were inherently more legible, we followed the humanist aesthetic in Lucida Sans.

6 Differentiated details

A problem at low resolutions is that letters begin to look alike because there isn't enough information to distinguish shapes quickly and easily. Type styles that assimilate forms, like geometric and grotesque sans-serifs, are particularly prone to this problem, especially along the upper region around the x-height, where traditional typefaces rely on details of shaping to differentiate letters.

For example, in an ostensibly simple sans-serif 'n', there is a white cut or crevice where the arch joins the left stem. This cut, along with the square corner of the left stem, keeps 'n' from being confused with 'o'. At low resolutions, these differentiating details can get obscured or lost, so we lowered the arch join, cutting more deeply into the shape. This also tended to increase the thickness of the arch, further distinguishing 'n' from 'o'. H. Carter (1937) noted both that the 18th century punch-cutter Fleischman made low cuts in the joins of h m n, and that Times Roman emphasized the strong arches of those same letters, so our decisions on these features had historical precedents as well as contemporary technical reasons.

We cut off terminals of curved strokes and diagonals vertically, to align with the vertical axes of digital rasters. However, we kept the serif-like terminal on 'a', to differentiate it from other letters. We tried to use the elegant Humanist 'g' with closed lower loop, but our bitmap tests showed that the letter shape did not survive at low resolutions and small sizes, so we settled on the "grotesque" style 'g' in Lucida Sans and Lucida Grande. As in Aldine humanist typefaces, we drew ascenders taller than capitals, to distinguish lower-case 'l' from capital 'I', and also to de-emphasize capitals slightly so that all-capital composition like acronyms, common in high-tech prose, and texts with frequent capitals, as in German orthography that capitalizes nouns, did not unduly interrupt the pattern of text. At lower resolutions, the distinguishing difference in height between capital 'I' (Eye) and lowercase 'l' (el) was neutralized, so for some purposes in later Lucida designs, we added serifs to capital I. These can still be found in Lucida Grande in Apple OS X, but are not the default forms.

7 Adjusted contrast of thick/thin strokes

We observed that in early laser printing and screen displays, thin hairlines were often "broken" by white gaps because of errors in rasterization. Such breaks made letters difficult to recognize and text annoying to read, so we thickened hairlines and serifs to avoid breakage and drop-outs. In the original Lucida seriffed faces, the thickness ratio of main stems to hairlines was 2 to 1, much thicker than in a face like Times Roman, giving Lucida a low contrast and less bright look. For Lucida Sans and its twin sibling, Lucida Grande, we used a thin-thick contrast of roughly 3 to 4, echoing the ductus of pen-written Renaissance roman and italic hands. As noted above, this also helped distinguish 'o' from 'n' because of the difference in thickness between the arch of 'n' and the curve of 'o' at the x-line.

In terms of laser-printer technology, the slightly thinner "hairlines" (which weren't very hair-like) of Lucida Sans helped keep the text from darkening too much on write-black laser printers. Aesthetically, we wanted to give our sans-serif more graphical modulation in its thick-thin contrast than in the stolid sans-serifs.

8 Regularization & repetition

We drew Lucida by hand but digitized it with the Ikarus software system developed by Peter Karow at URW in Germany. We edited the digital outlines to achieve precise regularity of base-line, x-line, capital line and other alignments, and to ensure that repeatable letter elements like stems, bowls, and serifs were digitally identical. This made it easier for software to recognize and adjust outlines, as was first achieved in Ikarus modules, and later implemented

in the “hints” of PostScript Type 1 and “instructions” of TrueType font rendering technologies.

Following research done by Philippe Coueignoux (1975) in his MIT PhD dissertation, we experimentally decomposed Lucida letter shapes to a small set of repeatable component parts from which all the letters could be assembled, in case extreme data compression was needed. This intriguing and instructive sort of data reduction turned out to be unneeded in the dominant commercial font formats for Latin fonts, so we didn’t pursue it further.

Although regularization and repetition remain popular approaches in type design more than three decades later, there are sometimes objections to the homogeneous look. Against this tendency, in 1992 we explored an opposite path, freely written forms of expressive letters in Lucida Handwriting, a connecting, casual script that we see often, especially in France, on Parisian bistro awnings, French perfume bottles and other Gallic expressions of charming exuberance.

9 Weight

The ratio of x-height to vertical stem thickness in Lucida normal weight fonts is 1 to 5.5. This is slightly heavier than many seriffed text faces. Although Times Roman has about the same stem to x-height ratio, it has thin hairlines and serifs that lighten the overall tone. The normal or regular stem weights of several popular grotesque sans-serifs are slightly lighter than Lucida Sans normal or regular weights, but the corresponding hairlines are slightly thicker, so the weight of Lucida seems comparable.

When we designed the first Lucida fonts, we chose a slightly dark weight to compensate for erosion around the edges of black letters on white background-illuminated screens and on write-white laser printers, which visually reduce weight, making text look weak in small sizes. The slightly dark weight made Lucida well adapted to most screen displays for almost 30 years, but printing on 300 dot-per-inch write-black laser printers had a slightly darker tone than we desired. When common printer resolutions increased to 600 dpi, this darkening tendency was mostly alleviated, because the percentage of weight added by write-black laser technology was reduced at the higher resolution. A fortuitous outcome of our choice of stem weight was that at 10 point, our target size, the main stems were four pixels thick when printed at 300 dots per inch, enabling thinner strokes to be 3, 2, or 1 pixel thick and a greater gamut of thickness modulation.

In 2014, we developed more than a dozen additional weights, ranging from UltraThin (1:22) to UltraBlack (1:2.3). With Ikarus we interpolated and extrapolated digital outlines of the hand-drawn weights, and with FontLab we hand-edited the results. The interpolations needed mostly minor editing but the extrapolations needed extensive editing. Both the interpolations and extrapolations first required the outlines to be edited so their spline point structures were isomorphic, that is, having the same numbers and kinds of points in the same orders. The whole process involved several iterations.

UltraThin ExtraThin Thin ExtraLite
Lite Book Text Normal Thick
ExtraThick Dark ExtraDark Bold
ExtraBold UltraBold Black
ExtraBlack UltraBlack

10 1984: First showing

Lucida was first shown at a meeting of the Association Typographique Internationale (ATypI) in London, September 1984, in the form of a type specimen chapbook from Imagen Corporation, a Silicon Valley laser printer manufacturer that was the first to license Lucida fonts. The Lucida booklet was designed by Michael Sheridan, Imagen’s type director whose appreciation of fine typography stemmed from his prior experience working at Grant Dahlstrom’s Castle Press, a Pasadena, California printing firm renowned for fine typography and printing. (The booklet is available at `tug.org/interviews/holmes-imagen-lucida.pdf`.)

Today, new and original typefaces are released in an unceasing flood, so it may be hard to recall that three decades ago, there were nearly none. As typography shifted from analog to digital technology in the 1970s and 1980s, typefaces for digital typesetters and printers were, with very few exceptions, digitizations of existing typefaces from previous eras of metal or photo-typography. (Among the few instances of original designs for high-resolution digital typesetters were the Marconi (1976) and Edison (1978) type families intended for use in newspa-

pers, designed by Hermann Zapf for the Hell-Digiset firm, which had invented and demonstrated the first digital typesetter.) In the article "Digital Typography", Bigelow and colleague Donald Day wrote (1983) that the initial, imitative phase of digital typography would eventually be followed by a creative phase of original design, but that had not happened by 1984. So one more reason we developed Lucida was to show that original digital designs could be effective and successful.

References

Arditi, A. (1996). Typography, print legibility, and low vision. *Remediation and Management of Low Vision*, Cole, R., ed., 237–248.

Bigelow, C. (1986). Principles of Type Design for the Personal Workstation. *Gutenberg-Jahrbuch*, 61, 253–270.

Bigelow, C. & Day, D. (1983). Digital typography. *Scientific American*, 249(2), 106–119.

Bigelow, C. & Holmes, K. (1986). The design of Lucida: An integrated family of types for electronic literacy. *Text Processing and Document Manipulation*, 1986, 1–17.

Bigelow, C.A. & Holmes, K. (1993). The design of a Unicode font. *Electronic Publishing*, 6(3), 289–305.

Bigelow, C.A. & Zanibbi, R. (2015). Analysis of typographical trends in European printing 1470–1660: Comparison of automated methods to palaeotypographical approaches. Presentation, American Printing History Association Conference: Printing on the Handpress & Beyond, 2015.

Bouma, H. (1970). Interaction effects in parafoveal letter recognition. *Nature*, 226(5241), 177.

Bouma, H. (1973). Visual interference in the parafoveal recognition of initial and final letters of words. *Vision Research*, 13(4), 767–782.

Campbell, F.W. & Robson, J.G. (1968). Application of Fourier analysis to the visibility of gratings. *The Journal of Physiology*, 197(3), 551–566.

Carter, H. (1937). Optical scale in type founding. *Typography* 4. Reprinted in *Printing Historical Society Bulletin*, 13, 144–148, 1984. issuu.com/letterror/docs/harry_carter_optical_scale_in_typefounding

Carter, M. (1985, December). Galliard: A revival of types of Robert Granjon. *Visible Language*, 19(1).

Coueignoux, P. J.-M. (1975). Generation of Roman printed fonts, Ph.D. dissertation, Massachusetts Institute of Technology. http://dspace.mit.edu/handle/1721.1/27408

Fournier, P.S. (1764). *Manuel typographique utile aux gens de Lettres (etc.)*, Vol. 1. L'auteur. (1766: Vol. 2.)

Huey, E.B. (1908). *The Psychology and Pedagogy of Reading*. The Macmillan Company.

Javal, E. (1879). Essai sur la physiologie de la lecture. Annales D'Oculistique 82, 242–253. [English translation: "Essay on the physiology of reading", Ciuffreda, K.J. & Bassil, N. (1990, October). *Ophthalmic and Physiological Optics*, Vol. 10.]

Javal, E. (1905). *Physiologie de la lecture et de l'écriture.* Félix Alcan.

Larson, K. (2004). The science of word recognition. microsoft.com/typography/ctfonts/WordRecognition.aspx

Larson, K., & Carter, M. (2016). Sitka: A collaboration between type design and science. In *Digital Fonts and Reading*, Dyson, M. & Suen, C.Y., eds., 37–53, World Scientific Publishing Co.

Legge, G.E. (2006). *Psychophysics of Reading in Normal and Low Vision*. Lawrence Erlbaum Assoc., CRC Press.

Legge, G.E., & Bigelow, C.A. (2011, August). Does print size matter for reading? A review of findings from vision science and typography. *Journal of Vision*, 11(5), 8. jov.arvojournals.org/article.aspx?articleid=2191906

LiVolsi, R., Zanibbi, R., & Bigelow, C. (2012, November). Collecting historical font metrics from Google books. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 351–355, IEEE.

Morison, S. (1997). Letter forms: typographic and scriptorial: two essays on their classification, history and bibliography (Vol. 45). Hartley & Marks Publishers.

Ohzawa, I. (2008). ohzawa-lab.bpe.es.osaka-u.ac.jp/ohzawa-lab/izumi/CSF/A_JG_RobsonCSFchart.html

Ovink, G.W. (1979). From Fournier to metric, and from lead to film. *Quaerendo*, 9(2), 95–127.

Pelli, D.G., Farell, B., & Moore, D.C. (2003). The remarkable inefficiency of word recognition. *Nature*, 423(6941), 752.

Pelli, D.G., Tillman, K.A., Freeman, J., Su, M., Berger, T.D., & Majaj, N.J. (2007, October). Crowding and eccentricity determine reading rate. *Journal of Vision*, 7(2), 20. jov.arvojournals.org/article.aspx?articleid=2122073

Ratdolt, E. (1486). bsbipad.bsb.lrz.de/nas/einblattdrucke/300001993_0_r.pdf

Rubinstein, R., Bigelow, C., Baudin, F., Lynch, E., & Levy, D. (1985). Proceedings of the typography interest group ACM CHI'85. *ACM SIGCHI Bulletin*, 17(1), 9–15.

Schulz-Anker, E. (1970). Syntax-Antiqua, a sans-serif on a new basis. *Gebrauchsgraphik*, 7, 49–56.

Tinker, M. (1963). *Legibility of Print*. Iowa State U. Press.

Vervliet, H.D.L. (2010). *French Renaissance Printing Types: A Conspectus.* The Bibliographical Society, The Printing Historical Society, Oak Knoll Press.

Xiong, Y.Z., Lorsung, E.A., Mansfield, J.S., Bigelow, C., & Legge, G.E. (2018). Fonts Designed for Macular Degeneration: Impact on Reading. *Investigative ophthalmology & visual science*, 59(10), 4182–4189.

◇ Charles Bigelow & Kris Holmes
lucidafonts.com

TEX Gyre text fonts revisited

Bogusław Jackowski, Piotr Pianowski,
Piotr Strzelczyk

1 Introduction

The collection of the TEX Gyre (TG for short) family of text fonts, an extensive revision of the freely available 35 base PostScript fonts, was released by the GUST e-foundry in 2006–2009 [4, 6]. Having finished this task, the GUST e-foundry team started to work on the math companion (in the OpenType, OTF, format [7]) for the TG text fonts [5]. Work on the math companion was finished two years ago. It resulted in the broadening of the repertoire of glyphs that could be used not only in math mode but also in text mode in technical documents. Hans Hagen, indefatigably coming up with interesting ideas, proposed to migrate the relevant glyphs to the text TG fonts. Needless to say, we seized on Hans's suggestion.

The first step was to decide which glyphs are to be migrated (and/or improved). Obviously, the list of candidates grew and grew. All in all, about 1000 glyphs were designated to be added, mostly geometrical and math symbols. A math companion, so far, was provided only for serif fonts, thus the consistent enhancement of the repertoire of the sans-serif fonts was a working test for our font generator — cf. Section 2 below.

We started with two fonts — the serif TG Pagella and the sans-serif TG Adventor. The results were satisfying. Now we are ready for the next step: to enhance similarly the rest of the TG family (TG Chorus, which is hardly suitable for technical texts, needs an individual approach). We believe, however, that we're over the hump. Below, we describe the most difficult and thus most interesting (to us) aspects of this stage of the TG project.

2 The MetaType1 engine

The scheme of the new workflow for the authors' MetaType1 software is depicted in Figure 1.

The main change in the engine consists of the replacement of several components (AWK plus Perl plus T1utils) by Python code with the FontForge library (finally, the library is available both under Unix and Windows). However, the FontForge library does not allow for sufficiently detailed control over the contents of the AFM and PFM files being generated, necessitating additional steps for fine tuning these files (dashed arrows in Figure 1).

First published in *Die TEXnische Komödie* 3/2018, pp. 11–20. Reprinted with permission.

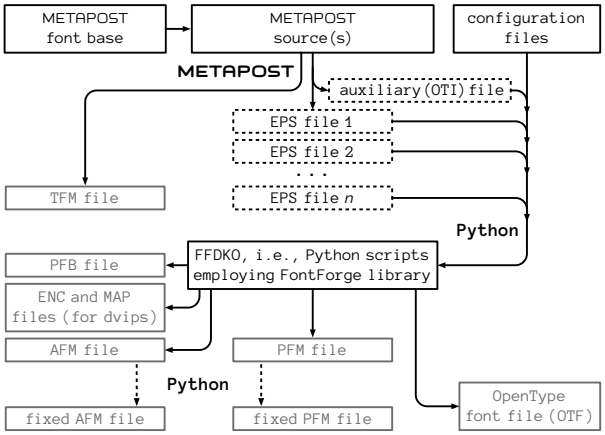


Figure 1: New MetaType1 engine: working scheme

The converter from Type 1 fonts to MetaType1 sources, implemented in AWK plus T1utils, has not yet been rewritten. We plan to rewrite it in Python with the FontForge library and enhance it to also process TrueType and OpenType files.

Of course, MetaPost is still the main module for generating glyph shapes. However, instead of spreading the auxiliary information into several output files (including EPS files), a single auxiliary output file, containing all the information needed for further processing, is generated. We will refer to this file as an *Olio Typographic Information* file, OTI. (*Olio* is a traditional name for a potpourri; it appears, e.g., in Robert Burns's *Address to a Haggis* — "French ragout or olio"). An OTI file is a container of "assorted bites and fragrances", indeed. Below is a fragment of an OTI file for TG Pagella Regular.

```
FNT FAMILY_NAME TeX Gyre Pagella
FNT HEADER_BYTE49 TeX Gyre Pagella
FNT GROUP_NAME TeX Gyre Pagella
FNT STYLE_NAME Regular
. . .
FNT WEIGHT Regular
FNT ITALIC_ANGLE 0
. . .
GLY A CODE 65
GLY A EPS 165
GLY A ANCHOR INBAS ALT.ogonek 623 -143
GLY A ANCHOR INBAS BOT_MAIN 392 -143
GLY A ANCHOR INBAS TOP_MAIN 392 819
GLY A WD 778 HT 692 DP 0 IC 6 GA 392
GLY A HSBW 778
GLY A BBX 15 -3 756 700
. . .
FNT FONT_DIMEN7 0.83
FNT DIMEN_NAME7 (extra space)
FNT FONT_DIMEN22 2.5
FNT DIMEN_NAME22 (math axis)
FNT HEADER_BYTE72 234
```

Each line of the OTI file contains either global information, concerning the whole font (prefix FNT), or local, concerning a given glyph (prefix GLY followed by the glyph name). We will not dwell too much on the details of the structure of OTI files as it will be documented elsewhere.

3 The glyph repertoire

As mentioned above, one of the important reasons for the "face-lifting" of the TG text fonts was our efforts on TG math fonts. Many symbols do not need the mathematical extension of the font structure (the MATH table in OTF files), but still prove useful in typesetting technical texts; for example, mathematical symbols (operators, relational symbols), arrows, geometrical symbols, etc. — see Figures 2 and 3. The number of glyphs grew from circa 750 to more than 1600, and may grow further in the future (see Section 5).

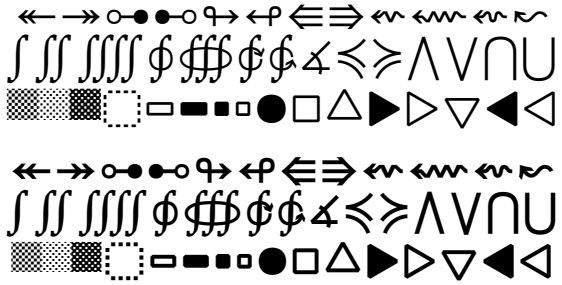


Figure 2: Sampling of added glyphs: TG Pagella regular (top) and bold (bottom)

The symbolic glyphs in the TG math fonts were designed only for regular serif variant fonts. The code, however, turned out to be flexible enough that with a few changes it was possible to generate bold and sans-serif variants.

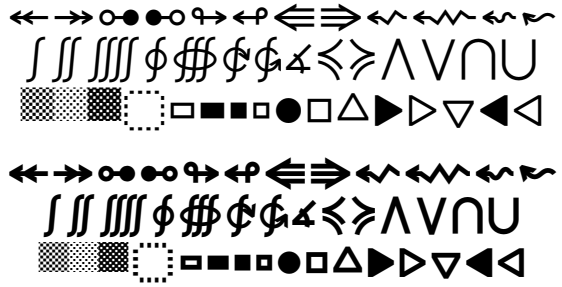


Figure 3: Sampling of added glyphs: TG Adventor regular (top) and bold (bottom)

Apart from enriching the repertoire, many glyphs were amended, due to, among other reasons, employing FontForge which, by default, minutely checks glyph outlines. For example, a tilde in TG Adventor was drawn from scratch, axes in several glyphs were corrected and so on.

$f(x) = 1/x$ $\quad$ $f(x) = 1/x$
$(x+1)(x-3) \geq 0$ $\quad$ $(x+1)(x-3) \geq 0$

Figure 4: Default math-oriented glyphs (left) vs. old glyphs produced by the OTF `ss10` feature (right)

Math-oriented glyphs existing already in the text fonts have been replaced with slightly different forms, better suited for math formulas. The old forms can be reached, if required, by using the OTF mechanism called features [2, 7, 8], namely, the 'stylistic set' feature `ss10`. Moreover, the Pagella Greek alphabet was taken from TG Pagella Math, that is, from Diego Puga's excellent Mathpazo with the kind permission of the author who agreed to let us use a fragment of his font under the GUST Font License (GFL [3]). The latter change involves significant change of the metric data. We are generally very reluctant to introduce such changes, but believe that the elegance of the Mathpazo Greek alphabet justifies that decision. Some glyphs from the Greek alphabet of TG Adventor (programmed in MetaType1) required improvements which also implied changes in metric data.

Rolling with the punches, we decided to abandon our initial idea of full compatibility with the metrics of the renowned Adobe 35 fonts [1]. The reason is twofold: first, Adobe metric data is, as we pointed out in the documentation of the TG fonts [4], itself inconsistent in several cases; second, preserving full compatibility makes sense only when the relevant metric files are used for previewing PostScript files to be printed on a printer with built-in Adobe Type 1 fonts. The TG fonts might have been used for such previewing, but, as it turned out, they have not (either in Ghostscript or in TEX Live; for example, the URW replacements for the Adobe 35 are typically used). Eventually, we decided to tune the TG metric data according to our experience whenever required. We believe that we will manage to avoid such changes in the future.

4 The font structure

The structure of the OTF fonts has been enhanced with the "backward compatible math style" feature (`ss10`) mentioned above and, moreover, with the mechanism of *anchors*, although the name "snaps" seems to us to be more accurate. Anchors enable putting accents precisely over glyphs. Roughly speaking, the anchor mechanism can be considered the analogue of the TEX `\accent` mechanism. Anchors, however, are implemented in a much more intricate

way: three features, obscurely documented in [2, 8], namely, `ccmp` (glyph composition / decomposition), `mark` (mark positioning, precisely, accent-to-base or mark-to-base positioning), and `mkmk` (mark-to-mark positioning, or, in other words, accent-to-accent positioning)[1] are used for this purpose, and yet the OTF anchor mechanism turns out insufficiently efficacious.

We were surprised by the complexity and laboriousness of the implementation of such a simple concept. Having read the explanations below, the reader and our virtual successors should feel forewarned and thus be less surprised.

"Anchors" or "marks" are actually pairs of numbers (planar points); the features `mark` and `mkmk` are supposed to position two glyphs in such a way that the respective anchors of the accent and accentee coincide. The former feature is used to position accents over or below base glyphs, the latter to position accents over or below accents. In the TG fonts, following common practice, only so-called combining accents (a subset of the block of combining diacritical marks [9]; that is, zero-width glyphs, protruding entirely to the left) are used for accenting and, thus, are equipped with anchors. In order to reduce the amount of anchor data, we decided to use as anchored accentees only accentless Latin letters plus letters "welded" with cedilla, horn, ogonek, and, additionally, l·, Ŀ, ł, Ł, ø, and Ø.

The `ccmp` feature enables the transformation of the input stream, namely: replacing glyphs and assembling a series of glyphs into a composed character or disassembling a composed character into a series of glyphs. The respective substitutions, in principle, must be defined in the font. Some engines, however, know better and perform such substitutions even if the font lacks relevant data. For example, Microsoft Word replaces 'i' (U+0069) followed by a combining top accent, say 'caroncomb' (U+030C), by a single glyph 'icaron' (U+01D0), provided that the latter is available in a given font; no further information, in particular, no `ccmp` feature, is required. Similarly, XeTeX joins accents with the base glyph into a single glyph, provided that the assembled form is present in the font; otherwise, accents are placed using anchors. This behaviour cannot be turned off — XeTeX simply uses system libraries which know better...

In the TG fonts, the `ccmp` feature is used to disassemble accented glyphs (but not glyphs with cedilla, ogonek, or horns) and to join into a single glyph letters followed by combining cedilla, ogonek, or horn (provided that the resulting glyph belongs to the repertoire of the font); otherwise, anchors are used. Moreover, `ccmp` is used to replace certain base glyphs and accents by their alternative forms; for example, 'i' and 'j' in the vicinity of top combining accents are replaced by their dotless forms, while top combining accents following an uppercase letter or ascender are replaced by their 'high' (flattened) variants.

The process of accenting using anchors, seemingly a trivial task, is, in fact, quite sophisticated. The Unicode standard recommends that if a text processor is being fed with a stream of text data containing a glyph, having assigned a Unicode slot, which is followed by a series of combining accents, then the text processor may position these accents over the main glyph [10], provided that the font contains the relevant positioning information. A typical example of the application of the anchor mechanism involving the `ccmp`+`mark`+`mkmk` features (as implemented in the new TG fonts) is depicted in Figure 5.

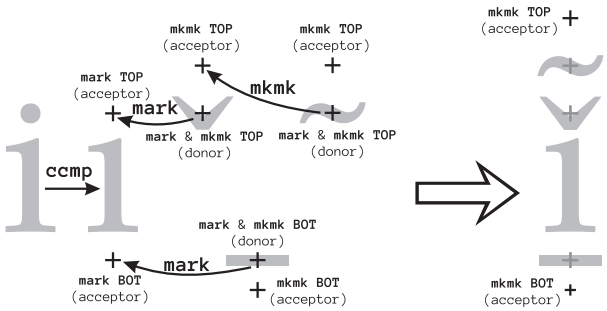


Figure 5: Anchor mechanism scheme — an example (explanations in text)

In the picture, feature names written in a small size denote the type of anchor (mark), large ones denote application of the respective features, labels 'TOP' and 'BOT' are defined by the user; the assumed input string is: 'i', 'macronbelowcomb', 'caroncomb', 'tildecomb' (that is: U+0069 U+030C U+0331 U+0303). The anchors have descriptors given in braces: *donor* and *acceptor* (taken from physical chemistry).

The process of accenting works here as follows:

- first, the `ccmp` feature enters the scene: the letter 'i', when followed by a combining upper accent, is replaced with 'dotlessi';
- next, the `mark` feature acts: the 'caroncomb' glyph is placed over 'dotlessi' in such a manner that its 'TOP' donor anchor coincides with the 'TOP' acceptor anchor of the glyph 'dotlessi'; as a result, both anchors become inactive;
- next, the `mark` feature enters once again: the 'macronbelowcomb' glyph is placed below 'dotlessi' in such a manner that its 'BOT' donor

[1] There is yet one more anchor feature `mset` (mark positioning via substitution) meant for handling peculiarities of the typesetting of Arabic texts.

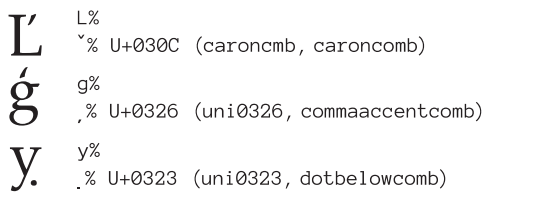


Figure 6: Peculiar positioning of certain accents (TEX source — right; the result — left)

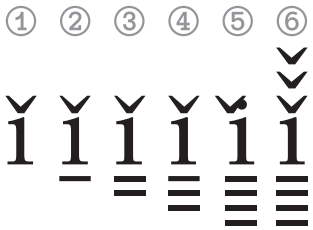


Figure 7: Troublesome replacement of 'i' by 'dotlessi' (explanations in the text)

anchor coincides with the 'BOT' acceptor anchor of the letter 'dotlessi'; as a result both anchors become inactive;

- finally, the `mkmk` feature intervenes: the 'tildecomb' glyph is placed above the newly placed 'caroncomb' in such a manner that its 'TOP' donor anchor coincides with the 'BOT' acceptor anchor of the 'caroncomb' glyph; as a result, both anchors become inactive;
- the resulting assembled glyph still has two active anchors, 'TOP' and 'BOT', that could be used by the `mkmk` feature, provided that the relevant glyphs appear in the input stream (immediately after 'tildecomb' in this case).

As one can see, the process of assembling glyphs using anchors is fairly complex. It should be admitted, however, that it enables handling such peculiarities as replacing a caron glyph with a comma-like variant if glyphs 'l', 'L' or 'J' are to be accented with caron, replacing a comma accent by a turned comma accent above 'g' (normally comma accent goes below a letter), or a singular positioning of a dot below accent at a letter 'y', as shown in Figure 6.

Unfortunately, not all cases of practical importance can be reliably handled. A notable example is the replacement of letters 'i' and 'j' by their dotless forms: the result depends to a large extent on the order of the glyphs in the input stream. In Figure 7, six cases are shown with different orders of glyphs in the input stream, namely (here, *i* stands for the letter 'i', *c* stands for 'caroncomb', and *m* stands for 'macronbelowcomb'): 1. *ic*; 2. *imc*; 3. *immc*; 4. *immmc*; 5. *immmmc*; 6. *icccmmmm*. Observe a malpositioned caron in case 5 — it is the result of our "design decision". The replacement 'i'→'dotlessi' is performed only if the top accents occur close to the letter 'i', preferably immediately after it. The OTF feature specification permits contextual replacements, that is, a certain number of bottom accents may precede the top one, but the preceding sequences must be enumerated explicitly. We decided to limit the length of the context to three glyphs (case 4 in Figure 7). If more bottom accents intervene between the letter 'i' and the top accent, the replacement is not performed and the glyphs are just overlapped (case 5 in Figure 7; as mentioned, combining accents have zero width and protrude to the left). Some fonts define longer contexts (for example, Charis SIL), but we decided that for practical purposes three is enough.

In order to avoid such situations, we recommend that the top accents go first, then the bottom accents (case 6 in Figure 7). The problem with our recommendation is that the order can be reversed by a text processing agent: according to the Unicode Standard recommendation, the bottom accent should go first and "canonical ordering behavior cannot be overridden by higher-level protocols" [11]. Some text processing agents apply the algorithm defined in [11] at the phase of reading the Unicode stream. In general, a typesetter cannot rely safely on the text processor. Even in TEX, the same text may be processed differently depending on the implementation.

In TEX, selected features, such as `ccmp`, `mark`, `mkmk`, etc., can be switched on or off on demand. Not all text processors offer such a possibility. A notable example is Microsoft Word which has these features switched on by default (it is not obvious whether it makes use of the Unicode ordering algorithm). As was mentioned, not all engines (in particular Microsoft Word, but also XeTEX) obey rules coded in the features `ccmp`, `mark`, `mkmk`. Incidentally, Figure 7 was created using LuaTEX.

In our opinion, the complexity of the implementation of anchors, resulting in a variety of approaches and implementations, is caused by the oversimplified mechanism of the OTF specification: the only allowed operations on a glyph are (re)positioning and substitution which is directly related to the OTF table structure and the basic tables, namely, `GPOS` and `GSUB`. The former operation is restricted merely to shifting, the latter to one-to-one, one-to-multiple and multiple-to-one replacements (which excludes reordering). Replacements can be either explicit or contextual, which adds complexity and does not help too much. In particular, fairly aged, not to say fossil, regular expressions are not allowed in contextual replacements.

5 Plans for the future

The next step (besides obvious cleaning of the sources, both Python and MetaPost) will undoubtedly be extending in a similar way the remaining TG text fonts, both sans-serif (Heros) and serif (Bonum, Cursor, Schola, and Termes). TG Chorus, as a chancery font, is not suitable for such an extension. We consider naming the stylistic features used in the TG fonts — it needs consideration, however; wrong names may likely introduce mess rather than order.

Having gathered experience with the text fonts, we would like to revisit the TG math fonts, with attention paid to sidebearings and math "staircase" kerns.

Moreover, we plan to remove all non-Python modules. As was mentioned, the path MetaType1 sources → OTF and Type 1 fonts is governed by Python; the reverse path, OTF and Type 1 fonts → MetaType1 sources, currently employs AWK and T1utils, thus, it cannot be used for converting TTF and OTF fonts to MetaType1 sources. We believe that the employing of FontForge (as a Python library) is the remedy.

We have no clear answer to the question of whether "small figures", accessible by features `subs` (subscripts), `sups` (superscripts), `sinf` (scientific inferiors) `numr` (numerators), and `dnom` (denominators), should be included in the text fonts; in math fonts math sub- and superscripts can be used instead. If we include these glyphs, then the next question arises: do we need special figures for small caps, `smcp`, other than, traditional in the TEX realm, old-style figures, also dubbed nautical? And do the small figures need variants commonly used for "normal" figures, that is, `lnum` (lining figures), `onum` (old-style figures), `pnum` (proportional figures), and `tnum` (tabular figures)? We are somewhat reluctant to add such a hodgepodge to an already intricate font structure.

6 Acknowledgements

We are indebted to all people and TEX groups that have supported our font enterprises. Almost all the GUST e-foundry projects were kindly supported by the Czechoslovak TEX Users Group CS TUG, the German-speaking TEX Users Group DANTE, the Polish TEX Users Group GUST, the Dutch-speaking TEX Users Group NTG, TUG India, UK-TUG, and, last but not least, TUG. In a few cases, GUTenberg, the French-speaking TEX Users Group, supported us too.

The exceptional, personal thanks we owe to our friends who have kept our spirits up for many years and tirelessly encouraged us to work on fonts: Hans Hagen, Johannes Küster, Jurek Ludwichowski, Volker RW Schaa, Jola Szelatyńska, Ulrik Vieth — hearty thanks! All trademarks belong to their respective owners and have been used here for informational purposes only.

References

[1] Adobe Systems Inc. Adobe metric files. `ftp://ftp.adobe.com/pub/adobe/type/win/all/afmfiles/base35/`

[2] Adobe Systems Inc. Feature file syntax. `adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html`

[3] GUST e-Foundry. GUST Font License. `gust.org.pl/projects/e-foundry/licenses`

[4] B. Jackowski, J. M. Nowacki, and P. Strzelczyk. TEX Gyre fonts collection. `gust.org.pl/projects/e-foundry/tex-gyre`

[5] B. Jackowski, P. Strzelczyk, and P. Pianowski. TEX Gyre math fonts collection. `gust.org.pl/projects/e-foundry/tg-math`

[6] B. Jackowski, P. Strzelczyk, and P. Pianowski. GUST e-foundry font projects. *TUGboat* 37(3):317–336, 2016. `tug.org/TUGboat/tb37-3/tb117jackowski.pdf`

[7] Microsoft Corp. OpenType Font Format, ver. 1.60, ISO/IEC 14496-22. `microsoft.com/typography/otspec160/`

[8] Microsoft Corp. Registered features. `microsoft.com/typography/otspec/featurelist.htm`

[9] Unicode Consortium. Combining diacritical marks. `unicode.org/charts/PDF/U0300.pdf`

[10] Unicode Consortium. The Unicode Standard 10.0.0; chapters 2.3 Compatibility Characters, 2.11 Combining Characters, 2.12 Equivalent Sequences and Normalization. `unicode.org/versions/Unicode10.0.0/ch02.pdf`

[11] Unicode Consortium. The Unicode Standard 10.0.0; chapter 3.11 Normalization Forms. `unicode.org/versions/Unicode10.0.0/ch03.pdf`

◇ Bogusław Jackowski
Piotr Pianowski
Piotr Strzelczyk
Rzeczypospolitej 8
80-369 Gdańsk, Poland
`b_jackowski` ,
`p.pianowski` ,
`p.strzelczyk`
`(at) gust dot org dot pl`

HINT: Reflowing TEX output

Martin Ruckert

Introduction

Current implementations of TEX produce `.pdf` (portable document format) or `.dvi` (device independent) files. These formats are designed for printing output on physical paper where the paper size and perhaps even the output resolution is known in advance. If these conditions are met, TEX, in spite of its age, still produces results of unsurpassed quality.

Due to improvements in display size, resolution, and technology over the past decades, it has become common practice to read TEX output on screen not only before printing but also instead of printing. For viewing TEX output before printing, excellent programs [4, 5] for "pre-viewing" are available. The prefix "pre" indicates that these programs intend to provide the user with a view that matches, as close as possible, the "final" appearance on paper. If, however, there is no intention of printing, for example if we read during a train ride on a mobile device, then matching the appearance on paper is of no importance, and we would rather prefer that the TEX output instead adapt to the size and resolution of our mobile device. Anyone who has been forced to read a PDF file designed for output on letter paper on a 5″ smartphone screen knows the problem.

For this reason, web browsers or ebooks use a reflowable text format. The HTML format, however, was never designed as a format for book printing, and `epub`, the ebook file format based on it, has inherited its deficiencies. Microsoft's PDF reflow solution — converting PDF files to Word documents — is an indication of the need for reflowable file formats but is a proprietary surrogate at best.

Considering that the TEX engine is able to reflow whole documents just by assigning new values to `hsize` and `vsize`, it seems long overdue to put this engine to use for that purpose.

The HINT project does just that. It defines a file format and provides two utilities: HiTEX, a special version of TEX to produce such files, and HINT, a standalone viewer to display them.

What is HINT?

Adopting the usual free software naming convention, HINT is a recursive acronym for "HINT is not TEX". But then, what is it? One answer could be: It's 90% TEX and the rest is a mixture of good and bad luck. So let me start explaining the details.

A first overview can be obtained by looking at Figures 1–3. The first figure is a simplified depiction of TEX's structure: A complex input processing part translates TEX input files into lists of 16-bit integers, called tokens, which form the machine language of TEX. The main loop of TEX is an interpreter that executes these programs, which eventually produce lots of boxes — most of them character boxes — and glue (and a few other items) that end up on the so-called contribution list. Every now and then, the page builder will inspect the contribution list and moves items to the current page. As soon as it is satisfied with the current page, it will invoke the (user-defined) output routine, again a token list, which can inspect the proposed page, change it at will, add insertions like footnotes, floating images, page headers and footers, even store it for later use, and eventually "ship out" the page to a `.dvi` file.

HINT splits this whole machinery into two separate parts: frontend and backend. The backend is the HINT viewer. The design goal is to reduce the processing in the backend as much as possible because we expect the viewer to run on small mobile devices where reduced processing implies reduced energy consumption and thus longer battery life. The frontend is the HiTEX version of TEX which is prevented from doing the full job of TEX because it does not know the values of `hsize` and `vsize`. As a first approximation of this split, HiTEX can write the contribution list to a file and HINT can read this file and feed it to the page builder as shown in Figures 2 and 3.

As an overall design goal, the HiTEX and HINT combination should produce exactly the same rendering as TEX for a given `hsize` and `vsize`.

A closer look at Figure 3 reveals that the "Output" arrow, representing the user's output routine, has disappeared; instead a new arrow, labeled "Templates", has taken its place. Keeping the full power of TEX's output routines would imply keeping the full TEX interpreter, all the token lists generated from the TEX input file, and possibly even the files that such an output routine might read or write in the viewer. This seemed to be too high a price and therefore output routines have been replaced by the template mechanism described below. This was the single most important design decision guided by the desire to allow lightweight viewers to run efficiently with a minimum amount of resources.

Several iterations were necessary to arrive at a suitable file format that was compact, easy to digest, and sufficiently expressive to provide the necessary information to the viewer. Finally, many smaller

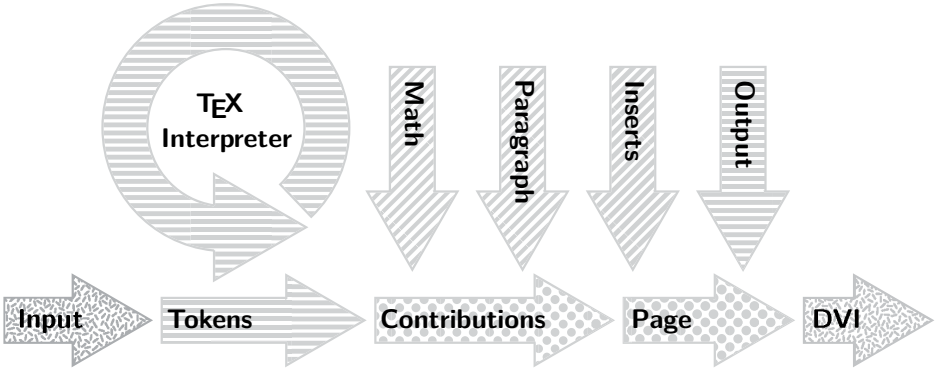


Figure 1: The structure of TEX

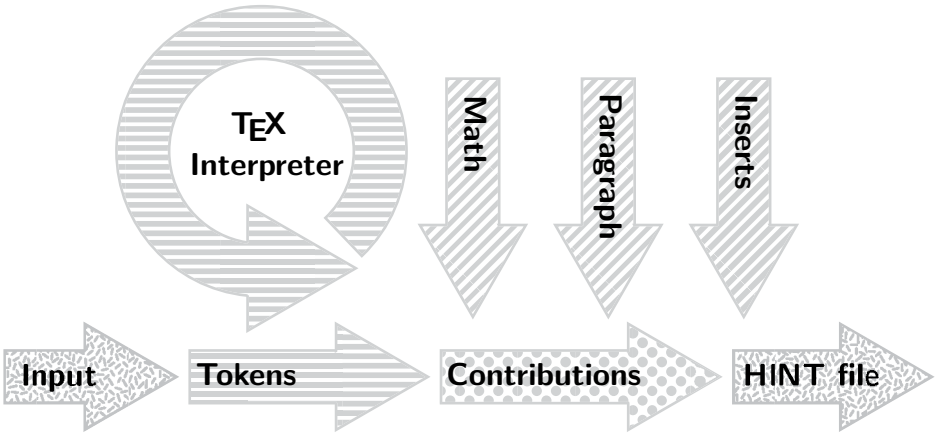


Figure 2: The structure of HiTEX

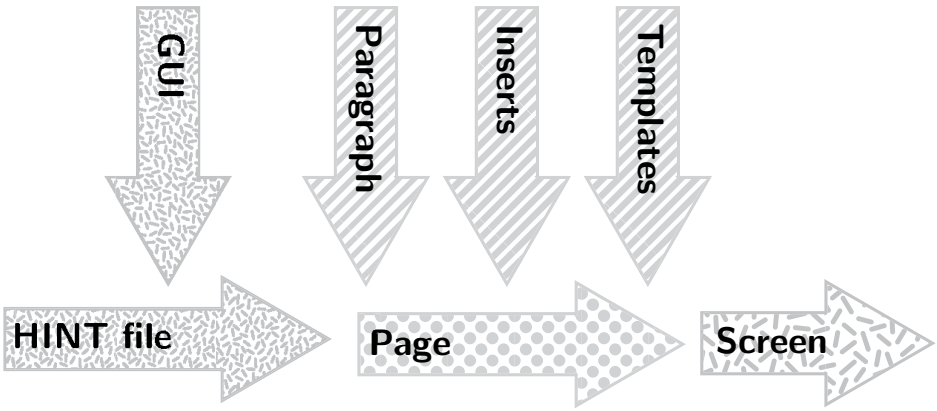


Figure 3: The structure of HINT

components of TEX needed to be moved back and forth between front- and backend before a satisfactory separation was accomplished.

Before I begin to describe these in more detail, I want to emphasize that the current state of the file format and the two utilities is not the end-point of development but a starting point. While I hope that the current specification provides enough functionality to attract a first small community of users, I see it more as a test-bed for experimentation with reflowable TEX output leading to better concepts, better formats, and better implementations.

Further, I consider the HINT viewer and its file format, while derived from TEX, as TEX independent. Why should not, for example, OpenOffice have a plug-in producing HINT output files?

```
HINT 1.0
<directory 4 (lists resources)
  <section 3 'TeXfonts/cmr10.tfm'>
  <section 4 'TeXfonts/cmr10.600pk'>
>
<definitions (lists definitions)
  <max <font 0>> (using just font 0)
  <font *0 'cmr10' 3 4
    <glue *13 (space skip)>
    <hyphen "-" 0 (default hyphen)>>
>
<content (a paragraph showing a kern)
  <par *0 "Hello w<kern -0x0.471D pt>orld!">
>
```

Figure 4: Example HINT file in long format

File formats

There are two file formats: a short form that represents HINT files as a compact byte stream for the viewer and a long form that represents HINT files in a readable form for editing and debugging. Figure 4 gives an example of the latter. Note the hexadecimal floating point notation in the kern node which is an exact representation of TEX's "scaled points".

After reading a HINT file, we have a byte stream in memory. This stream contains the directory, the definitions, the content stream, and finally resources. In the definition part, we define fonts and associate them with font-numbers for compact reference and do similar things for glues and other units that are used frequently. We supplement the definitions by setting suitable defaults. Then follows a content stream of at most 4GB. The latter restriction ensures that positions inside the content stream can be stored in 32 bits. The content stream consists of a list of nodes; each node representing a glue, a kern, a ligature, a discretionary hyphen, ..., or a box. Of course the content of boxes is again a list of nodes. After the content stream, we store file resources, for example image and font files.

If we want the viewer to support changing the page size while moving around in the stream — going to the next or previous page, following a link or using an index — practically any position in the stream can be the start of a page. This makes precomputing page starts impossible.

As a consequence, we need to be able to parse the content stream forward and backward. A node in the content stream therefore has a start byte, from which the parser can infer the structure and size of the node, and the same byte again as an end byte. Given an arbitrary position in the stream, it is possible to check if the current byte is a start byte or an end byte by computing the node-length from it and check the stream at the computed position for a matching byte. To be sure that the match is not a coincidence, the process can be repeated for a sequence of several nodes.

Start and end bytes contain a 5-bit "kind" and a 3-bit "info" field. This allows for 32 different kinds of nodes. The info bits can be used for small parameters or flags, or indicate the absence of certain fields in the node.

Lists. A special case is nodes describing lists of nodes. The method described above to distinguish start and end bytes is not feasible for a list of nodes because it is not possible to compute the size of the list from the start or end byte. Therefore, we store the size of the list content after the start byte and before the end byte. The three info bits are used to indicate whether the size is stored as 0, 1, 2, 3, or 4 bytes. This scheme enables a parser to find the corresponding start or end byte. Specifying 0 bytes for the size implies an empty list.

Texts. Because many lists consist mostly of characters, there is a special list format optimized for storing character nodes. We call such a list a "text". The start and end bytes of a text are like those of ordinary lists, but they are of kind "text". Only forward parsing is supported for a text node. Using the size information, we can skip easily to the beginning of a text.

A text can be thought of as a list of integers. Small integers in the range 0 to 127 are stored as single bytes; for larger integers the multi-byte encoding from UTF-8 is used. The integers from 0 to 32 are considered control codes, and all other integers are considered character codes — or rather glyph-numbers, to be more precise. The control codes are used for a variety of purposes. For example, a glyph-number in the range 0 to 32 can be specified by using the control code `0x1D` followed by the glyph-number; or arbitrary nodes can be inserted in the text after the control code `0x1E`.

A glyph-number references a specific glyph in the current font; the current font in a text is given implicitly. The control codes `0x00` to `0x07` can be used to select the 8 most common fonts; other fonts can be selected by using the control code `0x08` followed by the font number.

`hsize` and `vsize`

TEX treats `hsize` and `vsize` like any other dimension register; you can set them to any value and do all kinds of computations with them. HiTEX is more restrictive. At the global level, you cannot change `hsize` and `vsize` at all, because they denote the

dimensions given in the viewer. TEX, however, allows local modifications of dimension registers; for instance you can say `\vbox{\hsize = 0.5\hsize \advance \hsize by -8pt ...}` to obtain a vertical box, and inside this box, the value of `hsize` is just a bit smaller than half its global size. Hence, paragraphs inside this box are broken into lines that are almost half a page wide. The value of `hsize` will return to its old value once the box is completed. To make this possible, HiTEX treats dimensions as linear functions $\alpha+\beta\cdot$`hsize`$+\gamma\cdot$`vsize`, where α, β, and γ are constants. Computations are allowed as long as they stay inside the set of linear functions. For example `\multiply \hsize by \hsize` would not work. For lack of a better name, such a linear function is called an "extended dimension". The good news is that the viewer can convert an "extended dimension" immediately to a normal dimension since in the viewer `hsize` and `vsize` are always known.

Paragraphs

Breaking paragraphs into lines is TEX's most sophisticated and complex function. Fortunately the implementation is very efficient (it used to run fairly smoothly on my 8MHz 80286). It needs to be present in the frontend and in the backend. If `hsize` $= \alpha$ is a known constant (with $\beta = \gamma = 0$), the frontend can perform the line breaking; if $\beta \neq 0$ or $\gamma \neq 0$, line breaking must be performed in the backend.

On the other hand, we do not want the backend to perform hyphenation. Hyphenation is an expensive operation; it requires hyphenation tables to be present; and then it would be impossible for an author or editor to check the correctness of hyphenations. Therefore HiTEX will always insert all the discretionary hyphens that TEX would compute normally in the second pass of its line breaking algorithm. To reproduce the exact behavior of TEX's line breaking algorithm, the discretionary hyphens found in this way are marked and are used only during the second pass in the viewer. This gives preference to line breaks that do not use hyphenation (or only user specified discretionary hyphens) in the same way as TEX.

The paragraph shape is controlled by the variables `hangindent`, `hangafter`, and `parshape` which can be used to specify an individual indentation and length for any line in the paragraph. Obviously, these computations must be performed in the viewer. A complication arises if the viewer needs to start a page in the middle of a paragraph: the line number of the first line on the new page, and with it its indentation and length, then depends on how the previous page was formatted. This might not be known, for example if paging backward or if the page size has changed since the viewer had formatted the previous page. It remains an open question what gives the best user experience in such a situation.

Packing boxes and alignment

TEX knows two kinds of boxes: horizontal boxes, where the reference points of the content are aligned along the baseline; and vertical boxes, where the content is stacked vertically. Let's look at horizontal boxes; vertical boxes are handled similarly.

TEX produces horizontal boxes with the function `hpack`. The function traverses the content list and determines its total natural height, depth, and width. Furthermore, it computes the total stretchability and shrinkability. From these numbers it computes a glue ratio such that stretching or shrinking the glue inside the box by this ratio will make the box reach a given target width. HiTEX faces two problems: It might not be possible to determine the natural dimensions of the content, because, for example, the depth of a box can depend on how the line breaking algorithm forms the last line of a paragraph. In this case packing the box with `hpack` must be done in the viewer. But even if the natural dimensions of the content can be determined, a target width that depends on `hsize` will prevent HiTEX from computing a glue ratio. Therefore the HINT format knows three kinds of horizontal boxes: those that are completely packed, those that just need the computation of a glue ratio, and those that need a complete traversal of the box content.

Handling TEX's alignments introduces a little extra complexity. When TEX encounters a horizontal alignment, it packs the rows into `unset` boxes adding material from the alignment template and the appropriate `tabskip` glue. After all rows are processed, TEX packs the rows using the `hpack` function. At that point HiTEX can use the mechanisms just described for ordinary calls of `hpack`.

Baseline skips

When TEX builds vertical stacks of boxes, typically lines of text, it tries to keep the distances between the baselines constant, that is: independent of the actual depth of descenders or height of ascenders. Three parameters govern the insertion of glue between two boxes in vertical mode: TEX will insert glue to make the distance between baselines equal to `baselineskip` unless this would make the glue smaller than `lineskiplimit`; in the latter case, the

glue is set to `lineskip`. Additional white space between boxes, for instance a `\vskip 2pt`, does not interfere with this computation. Instead, TEX uses the variable `prev_depth`, containing the depth of the last box added to the list, for the computation. This offers a convenient lever for authors and macro designers to manipulate TEX's baseline calculations. For example setting `prev_depth` to the value `ignore_depth` will suppress the generation of a baselineskip for the next box on the list. This is of course a fact that the viewer should know about.

The HINT format is designed to be "stateless", that is: given the position of a page break in the stream, it is possible to read, understand, and format the page starting at that position or the page ending at that position. This turns the insertion of baseline skips into an interesting problem: In simple cases, when all relevant information is at hand, HiTEX can insert the correct glue directly. If some information is missing, a baseline node is generated. To process such a baseline node, the current values of the parameters mentioned before are required, and these parameters do change occasionally.

Storing the current values in every baseline node would require up to 54 bytes per node. HINT uses a more space-efficient approach: It defines default values that are constant for the entire stream. A baseline node using the defaults does not need to specify parameters. Further, the definition part of the stream can specify up to 256 baseline definitions, each defining the full set of parameters; such a parameter set can be used by specifying its number in a single byte. Only in the rare case that these two mechanisms are not sufficient must the baseline node contain the necessary values directly. The same approach is used for glues, extended dimensions, paragraphs, and displays. It can be generalized to arbitrary parameter sets.

Displayed equations

The positioning of displayed equations in TEX is no simple task. Usually the formula is centered on the line, but if `hsize` is so small that the formula would come too close to the equation number, it is centered in the remaining space between equation number and margin; if `hsize` is even smaller, the equation number will be moved to a separate line. Vertical spacing around the formula depends on the length of the last line preceding the display, which in turn depends on the outcome of the line breaking algorithm. If the line is short enough, TEX will use the `abovedisplayshortskip` glue, otherwise it uses `abovedisplayskip`. Of course there is also `belowdisplayshortskip` and `belowdisplayskip` to go with them. In addition, the variables controlling the paragraph shape influence the positioning of the displayed equation. The required computations must be done in the viewer; they are not very expensive but the code is complicated. HINT uses display nodes to describe displayed formulas. Fortunately, none of the math mode processing need be done in the viewer.

Images

Native TEX does not define a mechanism for including images, instead providing a generic extension mechanism. For the HINT viewer to be able to open and display any correct HINT file, we need to specify the image types that a viewer is required to support and the exact format of the image nodes. Image files are included in the resource part of the HINT file and are referenced by defining an image number, its position, and its size in the definition part.

For simplicity, the HINT viewer will not do any image manipulation except scaling. Scaling will be necessary to display the same HINT file on a wide variety of devices in a user friendly way. Various designs for the syntax and semantics of image nodes are possible and only the experience of real users will tell what is good or useless.

At present, images are treated like two dimensional glue: you can specify a width or a height, a stretchability, and a shrinkability. If neither width nor height are given, the natural width and height will be taken from the image file. When an image is part of the content of a box, it will stretch or shrink together with other glue to achieve the target size of the box. This mechanism works surprisingly well in practice; the image and the white space surrounding it scale in a consistent way to fill the space that is assigned to it by the enclosing box.

Page building

TEX's page builder starts at the top of a new page and collects vertical material, keeping track of its natural height, stretchability, and shrinkability until the page is so full that possible page breaks can only get worse. Then it uses the best page break found so far and moves remaining material back to the contribution list. Of course it also accounts for the size of inserts, and it uses the penalties found to estimate the goodness of a page break. HINT uses the same algorithm, complementing it with a reverse version that starts at the bottom of a new page. The reverse version is used when paging backward.

At the point where TEX calls the output routine, a new mechanism is needed, because (as mentioned above) we want the viewer to be simple, thus precluding the use of the TEX interpreter that would be necessary to execute a general output routine. HINT replaces output routines by page templates, but before we can describe this mechanism, it is necessary to see how HINT handles insertions.

Insertions. The TEX page builder identifies different insertions by their insertion number. It accounts for the contribution of inserted material to the total page height by weighting the insertion's natural height by the insertion scaling factor. There is also a constant overhead that needs to be added if the insertion is nonempty, for example the space occupied by a footnote rule and the space surrounding it.

HINT uses the concept of content streams for this. Stream number zero is used for the main page content; other stream numbers are defined in the definition part of the HINT file along with stream parameters such as the insertion scaling factor and the maximum vertical extent e that the stream content is allowed to occupy on the page. HiTEX maps insertion numbers to stream numbers and appends the insertion nodes to the content stream.

Streams have some more parameters: a list b of boxes that is used before and a list a that is used after the inserted material if it is not empty; the topskip glue g that is inserted between b and the first box of inserted material reduced by the height of this box; a stream number p, where the material from this stream should go if there is still space available for stream p; a stream number n, where the material from this stream should go if there is no more space available for the stream but still space available for stream n; a split ratio r that, if positive, specifies how to split the material of the stream between streams p and n.

The latter stream parameters are new and offer a mechanism to organize the flow of insertions on the page. For example, when plain TEX encounters a floating insertion, it decides whether there is still enough space on the current page and if so makes a mid-insert; otherwise a top-insert. HiTEX needs to postpone this decision. It will channel such an insertion to a stream with $e = 0$, $p = 0$, and n equal to the stream of top-inserts. When such an insertion arrives at the HINT page builder, it will check whether there is still space on stream 0, the main page, and if so moves the insertion there. Otherwise, setting the maximum extent e to zero forces the page builder to move the insertion to the stream n of top-inserts.

If the split ratio r is nonzero, the splitting of the stream will be postponed even further: The page builder will collect all contributions for the given stream and will split it in the given ratio between streams p and n just before assembling the final page. For example it is possible to put all the footnotes in one stream with an insertion scaling factor of 0.5 and split the collected footnotes into two columns using a split ratio of 0.5; with a cascade of splits, three or more columns are also possible.

Marks. TEX implements marks as token lists, and the output routine has access to the top, first, and bottom mark of the page. Sophisticated code can be written to execute these token lists producing very flexible headers or footers. In HINT we cannot use token lists but only boxes. Consequently, HINT uses the stream concept, developed for insertions, and extends it slightly. A flag can be added to a stream designating it as a "first" or "last" stream. Such a stream will retain at most one insertion per page. Now a package designer can open a stream for first marks and a stream for bottom marks, put TEX's marks into boxes, and add them into both streams. The implementation of top marks is difficult because it requires processing the preceding page. Top marks are not part of the present implementation.

Templates. Once the main page and all insertions are in place, HINT needs to compose the page. For this purpose it is possible in HiTEX to specify one or more page templates. A page template is just a `vbox` with arbitrary content: boxes, glue, rules, alignments, ..., and, most importantly, inserts. HiTEX will store the output template in the definition part together with its valid range of stream positions. When HINT needs to compose the page, it will search for an output template that includes the stream position of the current page in its range. It makes a copy of the template replacing each insert node by the material accumulated for it — insert node 0 will be replaced by the content of the main page. Material given as parameters a and b of an insert stream will be copied as necessary. After repacking the resulting `vbox` and all its subboxes, the `vbox` will be rendered on the display.

Implementation

For the work described above, I needed to make substantial changes to the TEX source code. The common tool chain from TEX Live uses `tangle` to convert `tex.web` into Pascal code (`tex.pas`) which is then translated by `web2c` [6] into C code. Already the translation to Pascal code expands all macros

and evaluates constant expressions, because neither is supported by Pascal. As a result, the generated Pascal code, let alone the further translation to C, becomes highly unreadable and cannot be used as a basis for any further work. So I wrote a translator converting the original WEB source code of TEX into cweb source code [2, 3]. This cweb source is the basis of the development of HiTEX and HINT.

For the implementation of HiTEX and HINT, I had only limited time at my disposal: my sabbatical during the fall semester of 2017/2018. As a consequence, I often moved on as soon as the current research problem had changed — in my view — into an engineering problem. This allowed me to make fast progress but left lots of "loose ends" in the code.

The current prototype has the functionality of Knuth's TEX with the adaptations described above, and without added features like search paths for input files or PDF specials. It is capable of generating format files for plain TEX or LATEX and it can handle even large files. The code for paging backwards is buggy because I occasionally implemented new features in the forward page builder and neglected to update the backwards page builder accordingly.

Open questions and future work

Conditionals. It seems reasonable to implement different output templates depending on screen size and aspect ratio. Also conditional content, for example a choice between a small and a wide table layout, might be useful. For a whole list of ideas, see [1].

Macros for LATEX support. Since the input part of HiTEX is taken directly from TEX, basic LATEX is supported. But since LATEX uses complex output procedures, many macros might need changes with variable page sizes now in mind. Templates are still an experimental feature of HINT that might need changes to better support LATEX.

Usage of control codes. Three control codes used in texts are indispensable: based on the bytes that follow, one control code switches to any of 256 possible fonts, one specifies an arbitrary character code, and one specifies an arbitrary node. The remaining 30 control codes provide plenty of room for experiments. Currently 8 of them are dedicated to font selection, 8 to reference globally predefined nodes, and 14 to reference font-specific predefined nodes. This should allow a convenient and compact encoding that can accomplish the most common operations with a single byte and use two or more bytes for less common operations. To decide whether the current dedication is optimal in this respect is an open question. A statistical analysis using a large collection of TEX documents should give an answer.

Images. The implementation of glue-like images is experimental. Another obvious ideas is the specification of background (and foreground) properties of boxes. The background could be a color (making rules a special case of boxes), a shading, or an image that can be stretched, or tiled, or positioned to fill the box. Certainly this would extend the capabilities of HINT beyond the necessities for TEX. Is this a direction worth considering?

Because it was easy to implement, currently only Windows bitmaps are supported. A full implementation should certainly support also JPEG and PNG files, and some form of vector graphic, probably SVG. I think it is better to have a small collection of formats that is well supported across all implementations than a long list of formats that enjoy only limited support. But how about sound and video? Should there be support? How could an extension mechanism look that keeps the HINT format open for future development?

Platforms. Currently the HINT viewer is written for the Windows platform just because this was convenient for me. Since HINT targets mobile devices, a HINT viewer for Android would be a next logical step. I also think that ebook readers deserve a better rendering engine and HINT would be a candidate.

References

[1] H. Hagen. Beyond the bounds of paper and within the bounds of screens; the perfect match of TEX and Acrobat. In *Proceedings of the Ninth European TEX Conference*, vol. 15a of *MAPS*, pp. 181–196. Elsevier Science, September 1995. ntg.nl/maps/15a/09.pdf

[2] M. Ruckert. Converting TEX from WEB to cweb. *TUGboat* 38(3):353–358, 2017. tug.org/TUGboat/tb38-3/tb120ruckert.pdf

[3] M. Ruckert. web2w*: Converting TEX from WEB to cweb*, 2017. ctan.org/pkg/web2w

[4] C. Schenk. Yap: Yet another previewer. miktex.org

[5] P. Vojta. Xdvi. math.berkeley.edu/~vojta/xdvi.html

[6] *Web2C: A TEX implementation.* tug.org/web2c

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München, Germany
ruckert (at) cs dot hm dot edu

Axessibility: Creating PDF documents with accessible formulae

D. Ahmetovic, T. Armano, C. Bernareggi, M. Berra, A. Capietto, S. Coriasco, N. Murru, A. Ruighi

Abstract

PDF documents containing formulae generated by LaTeX are usually not accessible by assistive technologies for visually impaired people (i.e., by screen readers and braille displays). The LaTeX package `axessibility.sty` that we have developed alleviates this issue, allowing one to create PDF documents where the formulae are read by these assistive technologies, since it automatically generates hidden comments in the PDF document (using the `/ActualText` attribute) in correspondence to each formula. This actual text is hidden in the PDF document, but the screen readers JAWS, NVDA and VoiceOver read it correctly. Moreover, we have created NVDA and Jaws dictionaries (in English and in Italian) that provide reading in the natural language in case the user does not know LaTeX commands. The package does not generate PDF/UA.

1 Introduction

In this paper, we describe `axessibility.sty`, a LaTeX package which allows automatic generation of a PDF document with formulae accessible by assistive technologies for visually impaired people. The package was first introduced in [3] and we took the material presented there as the starting point for the results presented here. Assistive technologies (screen readers and braille displays) perform satisfactorily with regard to digital documents containing text, but they still have a long way to go as far as formulae and graphs are concerned. A comprehensive overview of this problem can be found in [1, 2, 4].

Many studies have been conducted in order to improve the accessibility of digital documents with mathematical content. For instance, MathPlayer ensures accessibility of formulae inserted by using MathType in Word documents [13]. Another way to create accessible mathematical documents is given by the MathML language (see [7] for further information). However, accessibility of such documents is heavily affected by the versions of browsers, operating systems and screen readers, making this solution very unstable.

A system used by blind people for reading and writing mathematics is the LAMBDA system (Linear Access to Mathematics for Braille Device and Audiosynthesis). Mathematical language in LAMBDA is designed so that every symbol can be directly translated into words. For further details on LAMBDA we refer to [6]. Unfortunately, this system does not help to spread accessible digital documents, since it is used only by visually impaired people and is not a standard for the realization of documents by sighted people. Regarding LaTeX, assistive technologies can directly manage LaTeX documents. In this case, visually impaired people need to learn LaTeX in order to understand the commands. However, there is software which facilitates LaTeX comprehension and usability; one such is BlindMath [12]. Moreover, some converters from LaTeX to braille exist, see, e.g., [5] and [11].

In general, the most widespread digital documents are in PDF format. However, in the case of mathematical contents, they are not accessible at all, since formulae are usually unreadable by screen readers because they are bidimensional as images. None of the above systems directly support production of accessible formulae in PDF documents. This could be possible only performing specific tasks. For instance, using the Word editor, if each formula is manually tagged by the author (by using the alternative text), such a comment will be kept when the corresponding PDF file is generated and it will be read by the screen reader. However, this procedure does not help to improve the presence of accessible PDF documents, since it is a tedious and time-consuming method. It is difficult to imagine an author or editor performing these actions for the realization, e.g., of a book.

Currently, a standard and fast method for inserting accessible formulae into PDF documents is still lacking, despite it being a crucial issue for spreading accessible digital scientific documents. In [14], standard guidelines for accessibility of PDF documents are presented. Moreover, in [8], [9] and [10], an overview about accessibility of PDF documents is provided with a focus on mathematical contents.

In this paper, we describe the features of our package `axessibility.sty`, which provides the first method for an automated production of accessible PDF documents with mathematical contents. We would like to highlight that this package does not produce fully tagged PDF, such as the standard PDF/UA, but it does allow obtaining a PDF where formulae are described using the `/ActualText` attribute.

2 Problem statement

When a PDF document is generated starting from LaTeX, formulae are not accessible by screen readers and braille displays. They can be made accessible by inserting a hidden comment, i.e., actual text, similar to the case of web pages or Word documents.

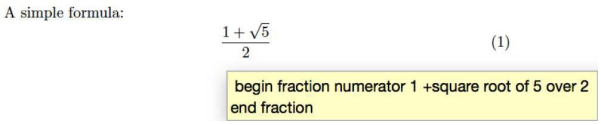


Figure 1: PDF document generated using the package `pdfcomment.sty`

This can be made, e.g., by using the LaTeX package `pdfcomment.sty` or using an editor for PDF files like Adobe Acrobat Pro. In any case, this task must be manually performed by the author and thus is surely inefficient, since the author must write the formulae and, in addition, insert a description for each formula. Note also that the package `pdfcomment.sty` does not allow insertion of special characters like *backslash*, *brace*, etc., in the comment. Moreover, with these solutions, the reading is bothersome, since the screen reader first incorrectly reads the formula and then the comment provided for the formula. In Figure 1, we show the PDF document generated from the following LaTeX code containing a simple formula with a manually inserted comment:

```
\documentclass{article}
\usepackage{pdfcomment}
\begin{document}
A simple formula:
\begin{equation}
\pdftooltip{
        \frac{1 + \sqrt{5}}{2}
        }{
        begin fraction numerator 1 +
    square root of 5 over 2 end fraction
        }
\end{equation}
\end{document}
```

When the screen reader accesses the PDF document, the formula will be read

> square root 1 plus 5 2 begin fraction numerator 1 plus square root of 5 over 2 end fraction

i.e., before reading the correct comment

> begin fraction numerator 1 plus square root of 5 over 2 end fraction

the screen reader reads incorrectly the formula

> square root 1 plus 5 2.

There are also some LaTeX packages that try to improve the accessibility of PDF documents produced by LaTeX. Specifically, the packages `accsupp.sty` (`ctan.org/pkg/accsupp`) and `accessibility.sty` (`github.com/AndyClifton/AccessibleMetaClass`) have been developed in order to obtain tagged PDF documents. However, neither package solves the problem of accessibility of formulae.

3 `axessibility.sty` LaTeX package

Our package, named `axessibility.sty`, solves the problem described in the previous section. It achieves this by inserting a hidden comment in the PDF file corresponding to any given formula. This comment, named `/ActualText`, contains the original LaTeX commands used to generate the formula. The hidden comment is read by screen readers and braille displays instead of the ASCII representation of the formula, which is often incorrect. We tested our package using Acrobat Reader together with the screen readers JAWS and NVDA for Windows and the native VoiceOver on macOS and iOS.

3.1 Usage

To create an accessible PDF document for visually impaired people, authors need only include the package `axessibility.sty` in the preamble of their LaTeX project. Mathematical environments automatically produce the `/ActualText` content and include it in the produced PDF file.

We handle the most common environments for inserting formulae, i.e., `equation`, `equation*`, `\[`, `\(`. Hence, any formula inserted using one of these environments is accessible in the corresponding PDF document. Additionally, the package enables copying the formula's LaTeX code from the PDF reader and pasting it elsewhere.

To preserve compatibility with Acrobat Reader, our package discourages the use of the underscore character `_`, which is not correctly read using screen readers in combination with this PDF reader. We suggest using the equivalent command `\sb`.

In-lined and display mathematical modes (`$`, `$$`) are not supported in this version of the package. However external scripts provided as companion software can also address these use cases.

If we use the package `axessibility.sty` applied to the previous example, we obtain the following LaTeX code:

```
\documentclass{article}
\usepackage{axessibility}
\begin{document}
    A simple formula:
    \begin{equation}
        \frac{1 + \sqrt{5}}{2}
    \end{equation}
\end{document}
```

We observe that, in this case, the author has to write the formula without adding anything else. Moreover,

inside the source code of the PDF file, we find an `/ActualText` tag with the LaTeX code, automatically generated by the `axessibility.sty` package.

```
/S/Span<</ActualText(\040\040\\frac
\040{1\040+\040\\sqrt
\040{5}}{2}\040)
>>
BDC
```

The screen reader reads correctly the LaTeX command `\frac{1+\sqrt{5}}{2}`. Moreover, we have created JAWS and NVDA dictionaries that provide the reading in the natural language in the case that the user does not know the LaTeX commands.

3.2 Technical overview

`axessibility.sty` first defines a pair of internal commands (`\BeginAxessible` and `\EndAxessible`) modelled on `\BeginAccSupp` and `\EndAccSupp` from the `accsup` package as follows:

```
\newcommand*{\BeginAxessible}[1]{%
  \begingroup
    \setkeys{ACCSUPP}{#1}%
    \edef\ACCSUPP@span{%
     /S/Formula<<%
        \ifx\ACCSUPP@Alt\relax
        \else
          /Alt\ACCSUPP@Alt
        \fi
        \ifx\ACCSUPP@ActualText\relax
        \else
          /ActualText\ACCSUPP@ActualText
        \fi
      >>%
    }%
    \ACCSUPP@bdc
    \ACCSUPP@space
  \endgroup
}
```

Specifically, as seen, `\BeginAxessible` adds a hidden comment that starts with `/S/Formula` instead of `/Span`. Then, to close:

```
\newcommand*{\EndAxessible}{%
  \begingroup
    \ACCSUPP@emc
  \endgroup
}
```

The second building block of this package is the wrapper. This routine takes the LaTeX code inside the formula, removes the tokens and passes it to `\BeginAxessible`:

```
\long\def\wrap#1{%
\BeginAxessible{method=escape,
ActualText=\detokenize\expandafter{#1},
Alt=\detokenize\expandafter{#1}}%
#1%
\EndAxessible
}
```

Finally, using the wrapper, we can redefine the mathematical environments using the command above. Here is an example using `equation`:

```
\renewenvironment{equation}{%
 \incr@eqnum
  \mathdisplay@push
  \st@rredfalse \global\@eqnswtrue
 \mathdisplay{equation}%
  \collect@body\wrap\auxiliaryspace}{%
  \endmathdisplay{equation}%
  \mathdisplay@pop
  \ignorespacesafterend}
```

4 Conclusions and future work

We have developed a LaTeX package that automatically generates comments for formulae when the PDF document is produced by LaTeX. The comments are hidden in the PDF document and they contain the LaTeX commands that generate the formulae. In this way, an accessible PDF document containing formulae is generated. Indeed, screen readers are able to access the comment when processing a formula and reading it. Moreover, we have created JAWS and NVDA dictionaries that provide for reading in natural languages in case the user does not know LaTeX commands.

There are a few issues that are yet to be solved with a pure LaTeX solution. Namely,

- Math environments delimited with `$`, `$$`.
- User-defined macros.
- Multi-line environments such as `\align` and `\eqnarray`.
- Semantic description of formulae.
- PDF/UA.

We address the first two problems using an external script — `axesscleaner.py`, from `github.com/integr-abile/axesscleaner` — coded in Perl and Python. We successfully "cleaned" two entire books using it. The script also replaces all underscore characters `_` with `\sb`. Using this solution we are now able to apply `axessibility.sty` to entire textbooks that were written without using the package in the first place. Multi-line environments are going to be

treated using a LaTeX solution that is currently in the test phase.

Concerning the last two problems, more in-depth research is in order. The authors are currently initiating the investigation to address these issues in future work.

The authors are also aware that the use of `/S/Formula` is in conflict with the internal structure of the PDF — the document is fully readable but it does not pass the so-called pre-flight test. This is not intended to be a final solution as the authors' goal is to create a PDF/UA with accessible formulae. However, we point out that this is an operative and reproducible solution, which automatically creates scientific material that is successfully used by people with visual impairment.

5 Acknowledgements

The authors wish to thank 'Fondazione Cassa di Risparmio di Torino', LeoClub (Biella, Italy) and the several volunteers with visual impairment who provided their fundamental contribution. We are grateful to U. Fisher for pointing us in the right direction concerning the use of the package `accsupp`. Specifically, she suggested not redefining `BeginAccSup`, but rather using a new environment. We also thank R. Moore for the fruitful discussion on the PDF structure and the suggestions to improve the manuscript.

References

[1] D. Ahmetovic, T. Armano, et al. Axessibility: A LaTeX package for mathematical formulae accessibility in PDF documents. In *Conference on Computers and Accessibility*. ACM, 2018.

[2] D. Archambault, B. Stöger, et al. Access to scientific content by visually impaired people. *Upgrade*, 2007.

[3] T. Armano, A. Capietto, et al. An automatized method based on LaTeX for the realization of accessible PDF documents containing formulae. In *Computers Helping People with Special Needs*, vol. 1089 of *Lecture Notes in Computer Science*. Springer, 2018.

[4] T. Armano, A. Capietto, et al. An overview on ICT for the accessibility of scientific texts by visually impaired students. In *SIREM-SIE-L Conference*, 2014.

[5] M. Batusic, K. Miesenberger, and B. Stöger. Labradoor, a contribution to making mathematics accessible for the blind. In *International Conference on Computers Helping People with Special Needs*. Springer, 1998.

[6] C. Bernareggi. Non-sequential mathematical notations in the LAMBDA system. In *Computers Helping People with Special Needs*. Springer, 2010.

[7] C. Bernareggi and D. Archambault. Mathematics on the Web: Emerging opportunities for visually impaired people. In *Conference on Web accessibility*. ACM, 2007.

[8] M. Borsero, N. Murru, and A. Ruighi. Il LaTeX come soluzione al problema dell'accesso a testi con formule da parte di disabili visivi. *ArsTEXnica* 22:12–18, Oct. 2016. `guitex.org/home/images/ArsTeXnica/AT022/murru-2016.pdf`

[9] R. Moore. Ongoing efforts to generate tagged PDF using pdfTeX. *TUGboat* 30(2):170–175, 2009. `tug.org/TUGboat/tb30-2/tb95moore.pdf`

[10] R. Moore. PDF/A-3u as an archival format for accessible mathematics. In S. Watt et al., eds., *CICM*. Springer, 2014.

[11] A. Papasalouros and A. Tsolomitis. Direct TeX-to-braille transcribing method. *Science Education for Students with Disabilities*, 2017.

[12] A. Pepino, C. Freda, et al. "BlindMath", a new scientific editor for blind students. In *Computers Helping People with Special Needs*. Springer, 2006.

[13] N. Soiffer. MathPlayer: Web-based math accessibility. In *Conference on Computers and Accessibility*. ACM, 2018.

[14] A. Uebelbacher, R. Bianchetti, and M. Riesch. PDF accessibility checker (PAC 2): The first tool to test PDF documents for PDF/UA compliance. In *Computers Helping People with Special Needs*. Springer, 2014.

◇ D. Ahmetovic
T. Armano
Dipartimento di Matematica
"G. Peano", Università di Torino
`dragan.ahmetovic` ,
`tiziana.armano`
`(at) unito.it`

◇ C. Bernareggi
Dipartimento di Informatica,
Università di Milano
`cristian.bernareggi (at)`
`unimi.it`

◇ M. Berra
A. Capietto
S. Coriasco
N. Murru
A. Ruighi
Dipartimento di Matematica
"G. Peano", Università di Torino
`michele.berra` ,
`anna.capietto` ,
`sandro.coriasco` ,
`nadir.murru` ,
`alice.ruighi`
`(at) unito.it`

Improving the representation and conversion of mathematical formulae by considering their textual context*

Moritz Schubotz, André Greiner-Petter, Philipp Scharpf, Norman Meuschke, Howard S. Cohl, Bela Gipp

Abstract

Mathematical formulae represent complex semantic information in a concise form. Especially in Science, Technology, Engineering, and Mathematics, mathematical formulae are crucial for communicating information, e.g., in scientific papers, and to perform computations using computer algebra systems. Enabling computers to access the information encoded in mathematical formulae requires machine-readable formats that can represent both the presentation and content, i.e., the semantics, of formulae. Exchanging such information between systems additionally requires conversion methods for mathematical representation formats.

We analyze how the semantic enrichment of formulae improves the format conversion process and show that considering the textual context of formulae reduces the error rate of such conversions. Our main contributions are: (1) providing an openly available benchmark dataset for the mathematical format conversion task consisting of a newly created test collection, an extensive, manually curated gold standard and task-specific evaluation metrics; (2) performing a quantitative evaluation of state-of-the-art tools for mathematical format conversions; (3) presenting a new approach that considers the textual context of formulae to reduce the error rate for mathematical format conversions.

Our benchmark dataset facilitates future research on mathematical format conversions as well as research on many problems in mathematical information retrieval. Because we annotated and linked all components of formulae, e.g., identifiers, operators and other entities, to Wikidata entries, the gold standard can, for instance, be used to train methods for formula concept discovery and recognition. Such methods can then be applied to improve mathematical information retrieval systems, e.g., for semantic formula search, recommendation of mathematical content, or detection of mathematical plagiarism.

* A version of this paper was published at JCDL 2018: M. Schubotz et al., "Improving the Representation and Conversion of Mathematical Formulae by Considering their Textual Context", in Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL), Fort Worth, USA, 2018.

1 Introduction

In STEM disciplines, i.e., Science, Technology, Engineering, and Mathematics, mathematical formulae are ubiquitous and crucial for communicating information in documents such as scientific papers, and to perform computations in computer algebra systems (CAS). Mathematical formulae represent complex semantic information in a concise form that is independent of natural language. These characteristics make mathematical formulae particularly interesting features to be considered by information retrieval systems.

In the context of digital libraries, major information retrieval applications for mathematical formulae include search and recommender systems as well as systems that support humans in understanding and applying mathematical formulae, e.g., by visualizing mathematical functions or providing autocompletion and error correction functionality in typesetting and CAS.

However, the extensive, context-dependent polysemy and polymorphism of mathematical notation is a major challenge to exposing the knowledge encoded in mathematical formulae to such systems. The number of mathematical concepts, e.g., mathematical structures, relations and principles, is much larger than the set of mathematical symbols available to represent these concepts. Therefore, the meaning of mathematical symbols varies in different contexts, e.g., in different documents, and potentially even in the same context. Identical mathematical formulae, even in the same document, do not necessarily represent the same mathematical concepts. Identifiers are prime examples of mathematical polysemy. For instance, while the identifier E commonly denotes energy in physics, E commonly refers to expected value in statistics.

Polymorphism of mathematical symbols is another ubiquitous phenomenon of mathematical notation. For example, whether the operator $\cdot$ denotes scalar multiplication or vector multiplication depends on the type of the elements to which the operator is applied. In contrast to programming languages, which handle polymorphism by explicitly providing type information about objects to the compiler, e.g., to check and call methods offered by the specific objects, mathematical symbols mostly denote such type information only implicitly, so that they need to be inferred from the context.

Humans account for the inherent polysemy and polymorphism of mathematical notation by defining context-dependent meanings of mathematical symbols in the text that surrounds formulae, e.g.,

for identifiers, subscripts and superscripts, brackets, and invisible operators. Without such explanations, determining the meaning of symbols is challenging, even for mathematical experts. For example, reliably determining whether $[a, b]$ represents an interval or the commutator $[a, b] = ab - ba$ in ring theory requires information on whether [ ] represent the Dirac brackets.

Enabling computers to access the full information encoded in mathematical formulae mandates machine-readable representation formats that capture both the presentation, i.e., the notational symbols and their spacial arrangement, and the content, i.e., the semantics, of mathematical formulae. Likewise, exchanging mathematical formulae between applications, e.g., CAS, requires methods to convert and semantically enrich different representation formats. The Mathematical Markup Language (MathML) allows one to encode both presentation and content information in a standardized and extensible way (see section 3).

Despite the availability of MathML, most Digital Mathematical Libraries (DML) currently exclusively use presentation languages, such as TEX and LATEX to represent mathematical content. On the other hand, CAS, such as Maple, Mathematica and SageMath,[1] typically use representation formats that include more content information about mathematical formulae to enable computations. Conversion between representation formats entails many conceptual and technical challenges, which we describe in more detail in section 2. Despite the availability of numerous conversion tools, the inherent challenges of the conversion process result in a high error rate and often lossy conversion of mathematical formulae in different representation formats.

To advance research on mathematical format conversion, we make the following contributions, which we describe in the subsequent sections:

1. We provide an openly available benchmark dataset to evaluate tools for mathematical format conversion (cf. section 3). The dataset includes:
   - a new test collection covering diverse research areas in multiple STEM disciplines;
   - an extensive, manually curated gold standard that includes annotations for both presentation and content information of mathematical formulae;
   - tools to facilitate the future extension of the gold standard by visually supporting human annotators; and
   - metrics to quantitatively evaluate the quality of mathematical format conversions.
2. We perform an extensive, quantitative evaluation of state-of-the-art tools for mathematical format conversion and provide an automated evaluation framework to support easily rerunning the evaluation in future research (cf. section 4).
3. We propose a novel approach to mathematical format conversion (cf. section 5). The approach imitates the human sense-making process for mathematical content by analyzing the textual context of formulae for information that helps link symbols in formulae to a knowledge base, in our case Wikidata, to determine the semantics of formulae.

2 Background and related work

In the following, we use the Riemann hypothesis (1) as an example to discuss typical challenges of converting different representation formats of mathematical formulae:

$$\zeta(s) = 0 \Rightarrow \Re s = \frac{1}{2} \vee \Im s = 0. \tag{1}$$

We will focus on the representation of the formula in LATEX and in the format of the CAS Mathematica. LATEX is a common language for encoding the presentation of mathematical formulae. In contrast to LATEX, Mathematica's representation focuses on making formulae computable. Hence the content must be encoded, i.e., both the structure and the semantics of mathematical formulae must be taken into consideration.

In LATEX, the Riemann hypothesis can be expressed using the following string:

```
\zeta(s) = 0 \Rightarrow \Re s
= \frac12 \lor \Im s=0
```

In Mathematica, the Riemann hypothesis can be represented as:

```
Implies[Equal[Zeta[s], 0], Or[Equal[Re[s],
Rational[1, 2]], Equal[Im[s], 0]]]
```

The conversion between these two formats is challenging due to a range of conceptual and technical differences.

First, the grammars underlying the two representation formats differ greatly. LATEX uses the unrestricted grammar of the TEX typesetting system. The entire set of commands can be re-defined and extended at runtime, which means that TEX effectively allows its users to change every character used

[1] The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose. All trademarks mentioned herein belong to their respective owners.

for the markup, including the \ character typically used to start commands. The high degree of freedom of the TEX grammar significantly complicates recognizing even the most basic tokens contained in mathematical formulae. In contrast to LATEX, CAS use a significantly more restrictive grammar consisting of a predefined set of keywords and set rules that govern the structure of expressions. For example, in Mathematica function arguments must always be enclosed in square brackets and separated by commas.

Second, the extensive differences in the grammars of the two languages are reflected in the resulting expression trees. Similar to parse trees in natural language, the syntactic rules of mathematical notation, such as operator precedence and function scope, determine a hierarchical structure for mathematical expressions that can be understood, represented, and processed as a tree. The mathematical expression trees of formulae consist of functions or operators and their arguments. We used nested square brackets to denote levels of the tree and Arabic numbers in a gray font to indicate individual tokens in the markup. For the LATEX representation of the Riemann hypothesis, the expression tree is:

$$\left[\quad \zeta_{\mathrm{l}}^{1}\,(_{\mathrm{l}}^{2}\,s_{\mathrm{l}}^{3}\,)_{\mathrm{l}}^{4} =_{\mathrm{l}}^{5} 0_{\mathrm{l}}^{6} \Rightarrow_{\mathrm{l}}^{7} \right.$$
$$\left. \Re_{\mathrm{l}}^{8}\, s_{\mathrm{l}}^{9} =_{\mathrm{l}}^{10} \left[\begin{smallmatrix}11\\ \vdots\end{smallmatrix}\right] 1_{\mathrm{l}}^{12}\, 2_{\mathrm{l}}^{13} \right]_{\mathrm{e}}^{14} \vee_{\mathrm{l}}^{15} \Im_{\mathrm{l}}^{16}\, s_{\mathrm{l}}^{17} =_{\mathrm{l}}^{18} 0_{\mathrm{l}}^{19} \quad \Big].$$

The tree consists of 18 nodes, i.e., tokens, with a maximum depth of two (for the fraction command `\frac12`). The expression tree of the Mathematica expression consists of 16 tokens with a maximum depth of five:

$$\left[\begin{smallmatrix}20\\ \Rightarrow\end{smallmatrix}\quad \left[\begin{smallmatrix}21\\ =\end{smallmatrix}\left[\begin{smallmatrix}22\\ \zeta\end{smallmatrix} s_{\mathrm{l}}^{23}\right]\ 0_{\mathrm{n}}^{24}\right]\right.$$
$$\left.\left[\begin{smallmatrix}25\\ \vee\end{smallmatrix}\ \left[\begin{smallmatrix}26\\ =\end{smallmatrix}\left[\begin{smallmatrix}27\\ \Re\end{smallmatrix} s_{\mathrm{l}}^{28}\right]\ \left[\begin{smallmatrix}29\\ \mathbb{Q}\end{smallmatrix} 1_{\mathrm{n}}^{30}\, 2_{\mathrm{n}}^{31}\right]\right]\ \left[\begin{smallmatrix}32\\ =\end{smallmatrix}\left[\begin{smallmatrix}33\\ \Im\end{smallmatrix} s_{\mathrm{l}}^{34}\right]\ 0_{\mathrm{n}}^{35}\right]\right]\right].$$

The higher complexity of the Mathematica expression reflects that a CAS represents the content structure of the formula, which is deeply nested. In contrast, LATEX exclusively represents the presentational layout of the Riemann hypothesis, which is nearly linear.

For the given example of the Riemann hypothesis, finding alignments between the tokens in both representations and converting one representation into the other is possible. In fact, Mathematica and other CAS offer a direct import of TEX expressions, which we evaluate in section 4.

However, aside from technical obstacles, such as reliably determining tokens in TEX expressions, conceptual differences also prevent a successful conversion between presentation languages, such as TEX, and content languages. Even if there was only one generally accepted presentation language, e.g., a standardized TEX dialect, and only one generally accepted content language, e.g., a standardized input language for CAS, an accurate conversion between the representation formats could not be guaranteed.

The reason is that neither the presentation language nor the content language always provide all the information required to convert an expression to the respective language. This can be illustrated by the simple expression: $F(a + b) = Fa + Fb$. The inherent content ambiguity of F prevents a deterministic conversion from the presentation language to a content language. F might, for example, represent a number, a matrix, a linear function or even a symbol. Without additional information, a correct conversion to a content language is not guaranteed. On the other hand, the transformation from content language to presentation language often depends on the preferences of the author and the context. For example, authors sometimes change the presentation of a formula to focus on specific parts of the formula or to improve its readability.

Another obstacle to conversions between typical presentation languages and typical content languages, such as the formats of CAS, are the restricted set of functions and the simpler grammars that CAS offer. While TEX allows users to express the presentation of virtually all mathematical symbols, thus denoting any mathematical concept, CAS do not support all available mathematical functions or structures. A significant problem related to the discrepancy in the space of concepts expressible using presentation markup and the implementation of such concepts in CAS are branch cuts. Branch cuts are restrictions of the set of output values that CAS impose for functions that yield ambiguous, i.e., multiple mathematically permissible outputs. One example is the complex logarithm [14, eq. 4.2.1], which has an infinite set of permissible outputs resulting from the periodicity of its inverse function. To account for this circumstance, CAS typically restrict the set of permissible outputs by cutting the complex plane of permissible outputs. However, since the method of restricting the set of permissible outputs varies between systems, identical inputs can lead to drastically different results [5]. For example, multiple scientific publications address the problem of accounting for branch cuts when entering expressions in CAS, such as [7] for Maple.

Our review of obstacles to the conversion of representation formats for mathematical formulae

Listing 1: MathML representation of the Riemann hypothesis (1) (excerpt).

```
<math><semantics><mrow>...
  <mo id="5" xref="20">=</mo>
  <mn id="5" xref="21">0</mn>
  <mo id="7" xref="19">⇒</ci>...</mrow>
<annotation-xml encoding="MathML-Content">
  <apply><implies id="19" xref="7" />
  <apply><eq id="20" xref="5" />...
  <apply><csymbol id="21" xref="1"
    cd="wikidata">Q187235...
</annotation-xml></semantics></math>
```

highlights the need to store *both* presentation and content information to allow for reversible transformations. Mathematical representation formats that include presentation and content information can enable the reliable exchange of information between typesetting systems and CAS.

MathML offers standardized markup functionality for both presentation and content information. Moreover, the declarative MathML XML format is relatively easy to parse and allows for cross references between presentation language (PL) and content language (CL) elements. Listing 1 represents excerpts of the MathML markup for our example of the Riemann hypothesis (1). In this excerpt, the PL token 7 corresponds to the CL token 19, PL token 5 corresponds to CL token 20, and so forth.

Combined presentation and content formats, such as MathML, significantly improve the access to mathematical knowledge for users of digital libraries. For example, including content information of formulae can advance search and recommendation systems for mathematical content. The quality of these *mathematical information retrieval systems* crucially depends on the accuracy of the computed document-query and document-document similarities. Considering the content information of mathematical formulae can improve these computations by:

1. Enabling the consideration of mathematical equivalence as a similarity feature. Instead of exclusively analyzing presentation information as indexed, e.g., by considering the overlap in presentational tokens, content information allows modifying the query and the indexed information. For example, it would become possible to recognize that the expressions $a(\frac{b}{c} + \frac{d}{c})$ and $\frac{a(b+d)}{c}$ have a distance of zero.
2. Allowing the association of mathematical tokens with mathematical concepts. For example, linking identifiers, such as E, m, and c, to energy, mass, and speed of light, could enable searching for all formulae that combine all or a subset of the concepts.
3. Enabling the analysis of structural similarity. The availability of content information would enable the application of measures, such as derivatives of the tree edit distance, to discover structural similarity, e.g., using λ-calculus. This functionality could increase the capabilities of *math-based plagiarism detection systems* when it comes to identifying obfuscated instances of reused mathematical formulae [10].

Content information could also enable interactive support functions for consumers and producers of mathematical content. For example, readers of mathematical documents could be offered interactive computations and visualizations of formulae to accelerate the understanding of STEM documents. Authors of mathematical documents could benefit from automated editing suggestions, such as auto-completion, reference suggestions, and sanity checks, e.g., type and definiteness checking, similar to the functionality of word processors for natural language texts.

Related work

A variety of tools exists to convert format representations of mathematical formulae. However, to our knowledge, Kohlhase et al. [26] presented the only study evaluating the conversion quality of tools. Many of the tools evaluated in that study are no longer available or out of date. Watt [27] presents a strategy to preserve formula semantics in TEX to MathML conversions. His approach relies on encoding the semantics in custom TEX macros rather than to expand the macros. Padovani [15] discusses the roles of MathML and TEX elements for managing large repositories of mathematical knowledge. Nghiem et al. [13] used statistical machine translation to convert presentation to content language. However, they do not consider the textual context of formulae. We will present detailed descriptions and evaluation results for specific conversion approaches in section 4.

Youssef [28] addressed the semantic enrichment of mathematical formulae in presentation language. He developed an automated tagger that parses LATEX formulae and annotates recognized tokens very similarly to part-of-speech (POS) taggers for natural language. Their tagger currently uses a predefined, context-independent dictionary to identify and annotate formula components. Schubotz et al. [19, 20]

proposed an approach to semantically enrich formulae by analyzing their textual context for the definitions of identifiers.

With their 'math in the middle' approach, Dehaye et al. [6] envision an entirely different approach to exchanging machine readable mathematical expressions. In their vision, independent and enclosed virtual research environments use a standardized format for mathematics to transfer mathematical expressions and numerical results between different systems.

For an extensive review of format conversion and retrieval approaches for mathematical formulae, refer to [18, Chapter 2].

3 Benchmarking MathML

This section presents MathMLben — a benchmark dataset for measuring the quality of MathML markup of mathematical formulae appearing in a textual context. MathMLben is an improvement of the gold standard provided by Schubotz et al. [24]. The dataset considers recent discussions of the International Mathematical Knowledge of Trust (`imkt.org`) working group, in particular the idea of a 'Semantic Capture Language' [9], which makes the gold standard more robust and easily accessible. MathMLben:

- allows comparisons to prior works;
- covers a wide range of research areas in STEM literature;
- provides references to manually annotated and corrected MathML items that are compliant with the MathML standard;
- is easy to modify and extend, i.e., by external collaborators;
- includes default distance measures; and
- facilitates the development of converters and tools.

In section 3.1, we present the test collection included in MathMLben. In section 3.2, we present the encoding guidelines for the human assessors and describe the tools we developed to support assessors in creating the gold standard dataset. In section 3.3, we describe the similarity measures used to assess markup quality.

3.1 Collection

Our test collection contains 305 formulae (more precisely, mathematical expressions ranging from individual symbols to complex multi-line formulae) and the documents in which they appear.

Expressions 1 to 100 correspond to the search targets used for the 'National Institute of Informatics Testbeds and Community for Information access Research Project' (NTCIR) 11 Math Wikipedia Task [24]. This list of formulae has been used for formula search and content enrichment tasks by at least 7 different research institutions. The formulae were randomly sampled from Wikipedia and include expressions with incorrect presentation markup.

Expressions 101 to 200 are random samples taken from the NIST Digital Library of Mathematical Functions (DLMF) [14]. The DLMF website contains 9,897 labeled formulae created from semantic LaTeX source files [3, 4]. In contrast to the examples from Wikipedia, all these formulae are from the mathematics research field and exhibit high quality presentation markup. The formulae were curated by renowned mathematicians and the editorial board keeps improving the quality of the markup of the formulae.[2] Sometimes, a labeled formula contains multiple equations. In such cases, we randomly chose one of the equations.

Expressions 201 to 305 were chosen from the queries of the NTCIR arXiv and NTCIR-12 Wikipedia datasets. 70% of these queries originate from the arXiv [1] and 30% from a Wikipedia dump.

All data are openly available for research purposes and can be obtained from `mathmlben.wmflabs.org`.[3]

3.2 Gold standard

We provide explicit markup with universal, context-independent symbols in content MathML. Since the symbols from the default content dictionary (CD) of MathML[4] alone were insufficient to cover the range of semantics in our collection, we added the Wikidata content dictionary [17]. As a result, we could refer to all Wikidata items as symbols in a content tree. This approach has several advantages. Descriptions and labels are available in many languages. Some symbols even have external identifiers, e.g., from the Wolfram Functions Site, or from StackExchange topics. All symbols are linked to Wikipedia articles, which offer extensive human-readable descriptions. Finally, symbols have relations to other Wikidata items, which opens a range of new research opportunities, e.g., for improving the taxonomic distance measure [25].

Our Wikidata-enhanced, yet standard-compliant, MathML markup facilitates the manual creation of content markup. To further support human assessors in creating content annotations, we extended the VMEXT visualization tool [21] to develop a visual support tool for creating and editing the MathMLben gold standard.

[2] `dlmf.nist.gov/about/staff`
[3] Visit `mathmlben.wmflabs.org/about` for a user guide.
[4] `www.openmath.org/cd`

Table 1: Special content symbols added to LaTeXML for the creation of the gold standard.

No	rendering	meaning	example ID
1	$[x, y]$	commutator	91
2	x^y_z	tensor	43, 208, 226
3	$x^\dagger$	adjoint	224, 277
4	x'	transformation	20
5	x°	degree	20
6	$x^{(dim)}$	contraction	225

For each formula, we saved the source document written in different dialects of LaTeX and converted it into content MathML with parallel markup using LaTeXML [11, 8]. LaTeXML is a Perl program that converts LaTeX documents to XML and HTML. We chose LaTeXML because it is the only tool that supports our semantic macro set. We manually annotated our dataset, generated the MathML representation, manually corrected errors in the MathML, and linked the identifiers to Wikidata concept entries whenever possible. Alternatively, one could initially generate MathML using a CAS and then manually improve the markup.

Since there is no generally accepted definition of expression trees, we made several design decisions to create semantic representations of the formulae in our dataset using MathML trees. In some cases, we created new macros to be able to create a MathML tree for our purposes using LaTeXML.[5] Table 1 lists the newly created macros. Hereafter, we explain our decisions and give examples of formulae in our dataset that were affected by the decisions.

- did not assign Wikidata items to basic mathematical identifiers and functions like `factorial`, `\log`, `\exp`, `\times`, `\pi`. Instead, we left these annotations to the DLMF LaTeX macros, because they represent the mathematical concept by linking to the definition in the DLMF and LaTeXML creates valid and accurate content MathML for these macros [GoldID 3, 11, 19, ...];
- split up indices and labels of elements as child nodes of the element. For example, we represent `i` as a child node of `p` in `p_i` [GoldID 29, 36, 43, ...];
- create a special macro to represent tensors, such as for $T_{\alpha\beta}$ [GoldID 43], to represent upper and lower indices as child nodes (see table 1);
- create a macro for dimensions of tensor contractions [GoldID 225], e.g., to distinguish the three dimensional contraction of the metric tensor in $g^{(3)}$ from a power function (see table 1);
- chose one subexpression randomly if the original expression contained lists of expressions [GoldID 278];
- remove equation labels, as they are not part of the formula itself. For example, in
$$E = mc^2, \qquad (\star)$$
the $(\star)$ is the ignored label;
- remove operations applied to entire equations, e.g., applying the modulus. In such cases, we interpreted the modulus as a constraint of the equation [GoldID 177];
- use additional macros (see table 1) to interpret complex conjugations, transformation signs, and degree-symbols as functional operations (identifier is a child node of the operation symbol), e.g., `*` or `\dagger` for complex conjugations [GoldID 224, 277], `S'` for transformations [GoldID 20], `30^\circ` for thirty degrees [GoldID 30];
- for formulae with multiple cases, render each case as a separate branch [GoldID 49];
- render variables that are part of separate branches in bracket notation. We implemented the Dirac Bracket commutator [ ] (we omitted the index `_\text{DB}`) and an anticommutator { } by defining new macros (see table 1). Thus, there is a distinction between a (ring) commutator
```
[a,b] = ab - ba
```
and an anticommutator
```
{a,b} = ab + ba,
```
without further annotation of Dirac or Poisson brackets [GoldID 91];
- use the command `\operatorname{}` for multi-character identifiers or operators [GoldID 22]. This markup is necessary because most of the LaTeX parsers, including LaTeXML, interpret multi-character expressions as multiplications of the characters. In general, this interpretation is correct, since it is inconvenient to use multi-character identifiers [2].

Some of these design decisions are debatable. For example, introducing a new macro, such as `\identifiername{}`, to distinguish between multi-character identifiers and operators might be advantageous to our approach. However, introducing many highly specialized macros is likely not a viable approach. A borderline example of this problem is Δx [GoldID 280]. Formulae of this form could be annotated as `\operatorname{}`, `\identifiername{}`

[5] `dlmf.nist.gov/LaTeXML/manual/customization/customization.latexml.html#SS1.SSS0.Px1`

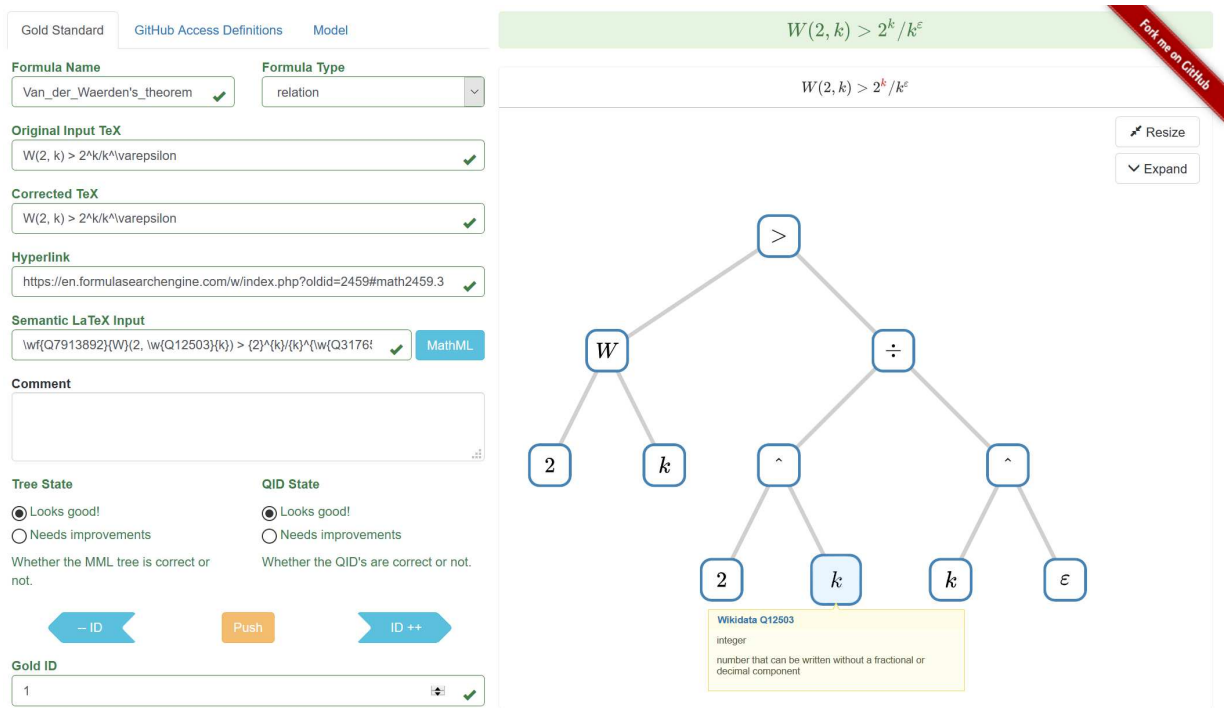


Figure 1: Graphical user interface to support the creation of our gold standard. The interface provides several TEX input fields (left) and a mathematical expression tree rendered by the VMEXT visualization tool (right).

or more generally as `\expressionname{}`. We interpret Δ as a difference applied to a variable, and render the expression as a function call.

Similar cases of overfeeding the dataset with highly specialized macros are bracket notations. For example, the bracket (Dirac) notation, e.g., [GoldID 209], is mainly used in quantum physics. The angle brackets for the Dirac notation, $\langle$ and $\rangle$, and a vertical bar | is already interpreted correctly as "latexml — quantum-operator-product". However, a more precise distinction between a twofold scalar product, e.g., $\langle a|b\rangle$, and a threefold expectation value, e.g., $\langle a|A|a\rangle$, might become necessary in some scenarios to distinguish between matrix elements and a scalar product.

We developed a Web application to create and cultivate the gold standard entries, which is available at `mathmlben.wmflabs.org`. The Graphical User Interface (GUI) provides the following information for each GoldID entry.

- **Formula Name:** name of the formula (optional)
- **Formula Type:** one of *definition*, *equation*, *relation* or *General Formula* (if none of the previous names fit)
- **Original Input TEX:** the LATEX expression as extracted from the source
- **Corrected TEX:** the manually corrected LATEX expression
- **Hyperlink:** hyperlink to the position of the formula in the source
- **Semantic LATEX Input:** manually created semantic version of the corrected LATEX field. This entry is used to generate our MathML with Wikidata annotations.
- **Preview of Corrected LATEX:** preview of the corrected LATEX input field rendered as SVG in real time using Mathoid [23], a service to generate SVG and MathML from LATEX input. It is shown in the top right corner of the GUI.
- **VMEXT Preview:** rendering of the expression tree based on the content MathML. The symbol in each node is associated with the symbol in the cross-referenced presentation markup.

Figure 1 shows the GUI for manual modification of the different formats of a formula. While the other fields are intended to provide additional information, the pipeline to create and cultivate a gold standard entry starts with the semantic LATEX input field. LATEXML will generate content MathML based on this input and VMEXT will render the generated content MathML afterwards. We control the output by using the DLMF LATEX macros [12] and

our developed extensions. The following list contains some examples of the DLMF LaTeX macros.

- `\EulerGamma@{z}`: $\Gamma(z)$: gamma function,
- `\BesselJ{\nu}@{z}`: $J_\nu(z)$: Bessel function of the first kind,
- `\LegendreQ[\mu]{\nu}@{z}`: $Q_\nu^\mu(z)$: associated Legendre function of the second kind,
- `\JacobiP{\alpha}{\beta}{n}@{x}`: $P_n^{(\alpha,\beta)}(x)$: Jacobi polynomial.

The DLMF web pages, which we use as one of the sources for our dataset, were generated from semantically enriched LaTeX sources using LaTeXML. Since LaTeXML is capable of interpreting semantic macros, generates content MathML that can be controlled with macros, and is easily extended by new macros, we also used LaTeXML to generate our gold standard. While the DLMF is a compendium for special functions, we need to annotate every identifier in the formula with semantic information. Therefore, we extended the set of semantic macros.

In addition to the special symbols listed in Table 1, we created macros to semantically enrich identifiers, operators, and other mathematical concepts by linking them to their Wikidata items. As shown in Figure 1, the annotations are visualized using yellow (grayscaled in print) info boxes appearing on mouseover. The boxes show the Wikidata QID, the name, and the description (if available) of the linked concept.

In addition to naming, classifying, and semantically annotating each formula, we performed three other tasks:

- correcting the LaTeX string extracted from the sources;
- checking and correcting the MathML generated by LaTeXML;
- visualizing the MathML using VMEXT.

Most of the extracted formulae contained concepts to improve human readability of the source code, such as commented line breaks (`%`⟨*newline*⟩), in long mathematical expressions, or special macros to improve the displayed version of the formula, e.g., spacing macros, delimiters, and scale settings, such as `\!`, `\,` or `\>`. Since they are part of the expression, all of the tested tools (including LaTeXML) try to include these formatting improvements into the MathML markup. For our gold standard, we focus on the pure semantic information and forgo formatting improvements related to displaying the formula. The corrected TeX field shows the cleaned mathematical LaTeX expression.

Using the corrected TeX field and the semantic macros, we were able to adjust the MathML output using LaTeXML and verify it by checking the visualization from VMEXT.

3.3 Evaluation metrics

To quantify the conversion quality of individual tools, we computed the similarity of each tool's output and the manually created gold standard. To define the similarity measures for this comparison, we built upon our previous work [25], in which we defined and evaluated four similarity measures: taxonomic distance, data type hierarchy level, match depth, and query coverage.

The measures taxonomic distance and data type hierarchy level require the availability of a hierarchical ordering of mathematical functions and objects. For our use case, we derived this hierarchical ordering from the MathML content dictionary. The measures assign a higher similarity score if matching formula elements belong to the same taxonomic class. The match depth measure operates under the assumption that matching elements, which are more deeply nested in a formula's content tree, i.e., farther away from the root node, are less significant for the overall similarity of the formula, hence are assigned a lower weight. The query coverage measure performs a simple 'bag of tokens' comparison between two formulae and assigns a higher score the more tokens the two formulae share.

In addition to these similarity measures, we also included the tree edit distance. For this purpose, we adapted the robust tree edit distance (RTED) implementation for Java [16]. We modified RTED to accept any valid XML input and added math-specific 'shortcuts', i.e., rewrite rules that generate lower distance scores than arbitrary rewrites. For example, rewriting $\frac{a}{b}$ to ab^{-1} causes a significant difference in the expression tree: Three nodes $(\wedge, -, 1)$ are inserted and one node is renamed $\div \to \cdot$. The 'cost' for performing these edits using the stock implementation of RTED is $c = 3i + r$. However, the actual difference is an equivalence, which we think should be assigned a cost of $e < 3i + r$. We set $e < r < i$.

4 Evaluation of context-agnostic conversion tools

This section presents the results of evaluating existing, context-agnostic conversion tools for mathematical formulae using our benchmark dataset MathMLben (see section 3). We compare the distances between the presentation MathML and the content MathML tree of a formula yielded by each tool to the respective trees of formulae in the gold standard. We use the tree edit distance with customized weights and math-specific shortcuts. The goal of shortcuts is

eliminating notational-inherent degrees of freedom, e.g., additional PL elements or layout blocks, such as `mrow` or `mfenced`.

4.1 Tool selection

We compiled a list of available conversion tools from the W3C[6] wiki, from *GitHub*, and from questions about automated conversion of mathematical LaTeX to MathML on *Stack Overflow*. We selected the following converters:

- LaTeXML: supports converting generic and semantically annotated LaTeX expressions to XML/HTML/MathML. The tool is written in Perl [11] and is actively maintained. LaTeXML was specifically developed to generate the DLMF web page and can therefore parse entire TeX documents. Notably, LaTeXML supports conversions to content MathML.
- LaTeX2MathML(a.k.a. LaTeX2MML): a small Python project to generate presentation or content MathML from generic LaTeX expressions.[7]
- Mathoid: a service using Node.js, PhantomJS and MathJax (a JavaScript display engine for mathematics) to generate SVG and MathML from LaTeX input. Mathoid is currently used to render mathematical formulae on Wikipedia [23].
- SnuggleTeX: an open-source Java library developed at the University of Edinburgh.[8] The tool can convert simple LaTeX expressions to XHTML and presentation MathML.
- MathToWeb: an open-source Java-based web application that generates presentation MathML from LaTeX expressions.[9]
- TeXZilla: a JavaScript web application for LaTeX to MathML conversion capable of handling Unicode characters.[10]
- Mathematical: an application written in C and wrapped in Ruby to provide a fast translation from LaTeX expressions to the image formats SVG and PNG. The tool also provides translations to presentation MathML.[11]
- CAS: we included a prominent CAS capable of parsing LaTeX expressions.
- Part-of-Math (POM) Tagger: a grammar-based LaTeX parser that tags recognized tokens with information from a dictionary [28]. The POM tagger is currently under development. In this paper, we use the first version. In [5], this version was used to provide translations LaTeX to the CAS Maple. In its current state, this program offers no export to MathML. We developed an XML exporter to be able to compare the tree provided by the POM tagger with the MathML trees in the gold standard.

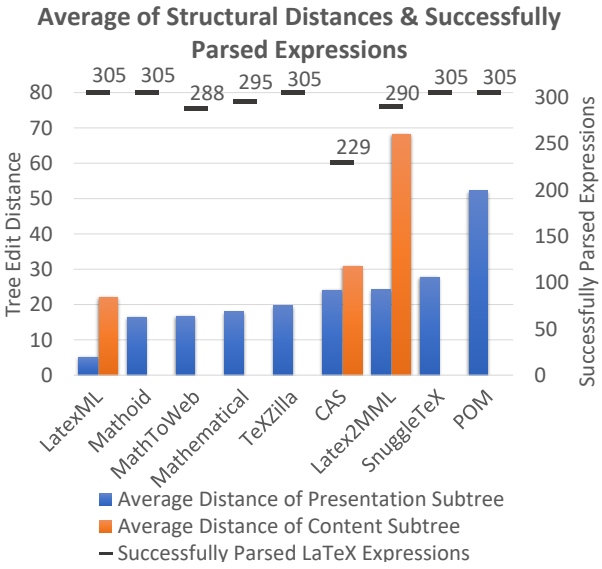


Figure 2: Overview of the structural tree edit distances (using $r = 0$, $i = d = 1$) between the MathML trees generated by the conversion tools and the gold standard MathML trees.

4.2 Testing framework

We developed a Java-based framework that calls the programs to parse the corrected TeX input data from the gold standard to presentation MathML, and, if applicable, to content MathML. In case of the POM tagger, we parsed the input string to a general XML document. We used the corrected TeX input instead of the originally extracted string expressio (see section 3.2).

Executing the testing framework requires the manual installation of the tested tools. The POM tagger is not yet publicly available.

4.3 Results

Figure 2 shows the averaged structural tree edit distances between the presentation trees (blue) and content trees (orange) of the generated MathML files and the gold standard. To calculate the structural tree edit distances, we used the RTED [16] algorithm with costs of $i = 1$ for inserting, $d = 1$ for deleting and $r = 0$ for renaming nodes. Furthermore, the

[6] `www.w3.org/wiki/Math_Tools`
[7] `github.com/Code-ReaQtor/latex2mathml`
[8] `www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html`
[9] `www.mathtowebonline.com`
[10] `fred-wang.github.io/TeXZilla`
[11] `github.com/gjtorikian/mathematical`

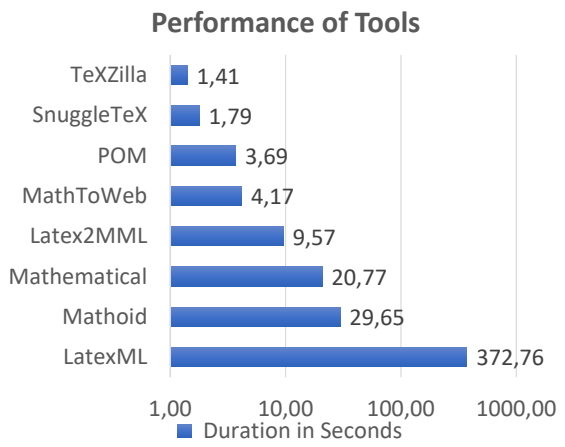


Figure 3: Time in seconds required by each tool to parse the 305 gold standard LaTeX expressions in logarithmic scale.

figure shows the total number of successful transformations for the 305 expressions (black ticks).

We consider differences of the presentation tree to the gold standard as deficits, because the mapping from LaTeX expressions to rendered expressions is unique (as long as the same preambles are used). A larger number indicates that more elements of an expression were misinterpreted by the parser. However, certain differences between presentation trees might be tolerable, e.g., reordering commutative expressions, while differences between content trees are more critical.

Also note that improving content trees may not necessarily improve presentation trees and vice versa. In case of $f(x + y)$, the content tree will change depending whether f represents a variable or a function, while the presentation tree will be identical in both cases. In contrast, $\frac{a}{b}$, a/b, and a/b have different presentation trees but identical content trees.

Figure 3 illustrates the runtime performance of the tools. We excluded the CAS from the runtime performance tests, because the system is not primarily intended for parsing LaTeX expressions, but for performing complex computations. Therefore, runtime comparisons between a CAS and conversion tools would not be representative. We measured the times required to transform all 305 expressions in the gold standard and write the transformed MathML to the storage cache. Note that the native code of LaTeX2MML, Mathematical and LaTeXML were called from the Java Virtual Machine (JVM) and Mathoid was called through local web-requests, which increased the runtime of these tools. The figure is scaled logarithmically. We would like to emphasize that LaTeXML is designed to translate sets of LaTeX documents instead of single mathematical expressions. Most of the other tools are lightweight engines.

In this benchmark, we focused on the structural tree distances rather than on distances in semantics. While our gold standard provides the information necessary to compare the extracted semantic information, we will focus on this problem in future work (see section 6).

5 Towards a context-sensitive approach

In this section, we present our new approach that combines textual features, i.e., semantic information from the surrounding text, with the converters to improve the outcome. Figure 4 illustrates the process of creating the gold standard, evaluating conversions, and how we plan to improve the converters with tree refinements (outside the MathMLben box). Our improvement approach includes three phases.

1. In the first phase, the Mathematical Language Processing (MLP) approach [19] extracts semantic information from the textual context by providing identifier-definiens[12] pairs.
2. The MLP annotations self-assess their reliability by annotating each identifier-definiens pair with its probabilities. Often, the methods do not find highly ranked semantic information. In such cases, we combine the results from the MLP with a dictionary-based method. In particular, we use the dictionaries from the POM tagger [28] that associate context-free semantics with the presentation tree. Since the dictionary entries are not ranked, we use them to drop unmentioned identifier-definiens pairs and choose the highest rank of the remaining pairs.
3. Based on the chosen semantic information, we redefine the content tree by reordering the nodes and subtrees.

Currently, the implementation is too immature to release it as a semantic annotation package. Instead, we discuss the method using the following selected examples that represent typical classes of disambiguation problems:

- Invisible operator disambiguation for the times vs. apply special case.
- Parameter vs. label disambiguation for subscripts.
- Einstein notation discovery.
- Multi-character operator discovery.

[12] In a definition, the *definiendum* is the expression to be defined and *definiens* is the phrase that defines the definiendum. Identifier-definiens pairs are candidates for an Identifier-definition. See [19] for a more detailed explanation.

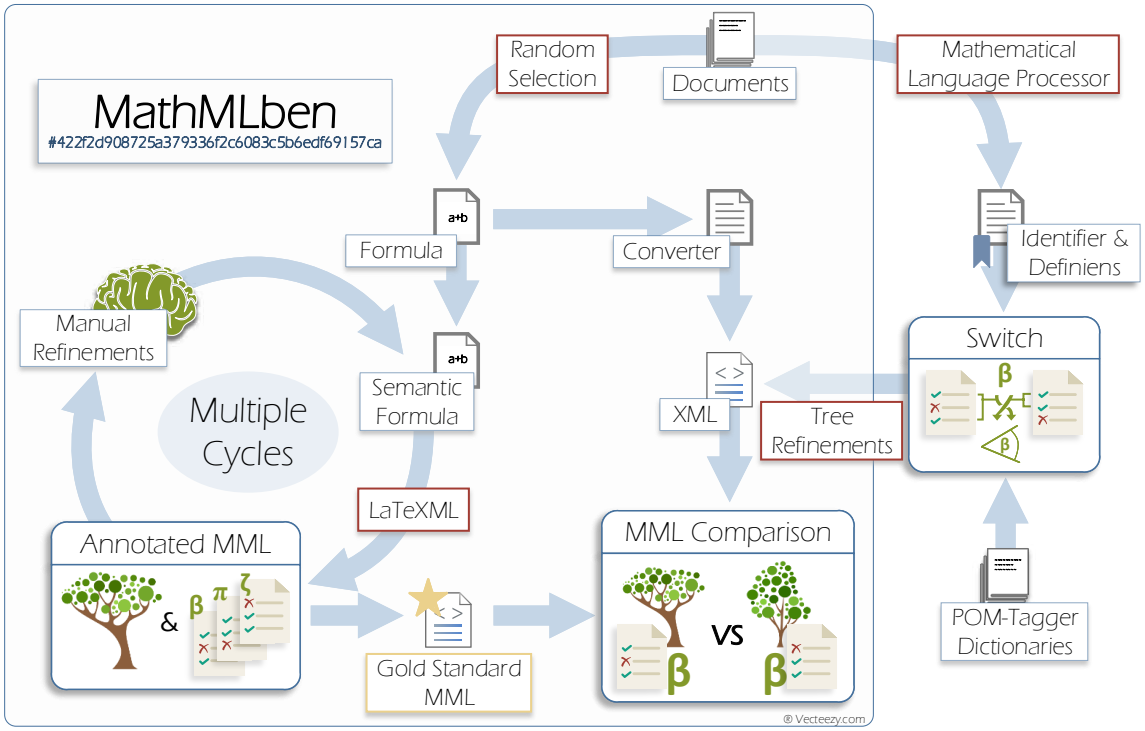


Figure 4: Mathematical language processing is the task of mapping textual descriptions to components of mathematical formulae (Part-of-Math tagging).

Learning special notations like the examples above is subject to future work. However, we deem it reasonable to start with these examples, since our manual investigation of the tree edit distances showed that such cases represented major reasons for errors in the content MathML tree.

Previously, the MLP software was limited to extracting information about identifiers, not general mathematical symbols. Moreover, the software was optimized for the Wikipedia dataset. We thus expanded the software for this study to enable parsing pure XHTML input as provided by the NTCIR tasks and the DLMF website. Achieving this goal required realizing a component for symbol identification. We chose the strategy of considering every simple expression that is not an identifier as a candidate for a symbol.

For our first experiments we tried to improve the output by LaTeXML, since LaTeXML performs best in our tests and it was able to generate content MathML. Moreover, with the newly developed semantic macros, we are able to optimize MathML in a pre-processing step by enhancing the input LaTeX expression. Consequently, we do not need to develop complex post-processing algorithms to manipulate content MathML.

As part of this study, we created a custom style sheet that fixes the following problems: (1) use of the power symbols for superscript characters unless Einstein notation was discovered, (2) interpretation of subscript indices as parameters, unless they are in text mode; for text mode, the ensemble of main symbol and subscript will be regarded as an identifier, (3) symbols that are considered as a 'function' are applied to the following identifier, rather than being multiplied with the identifier.

First experiments using these refinement techniques have proven to be very effective. We have chosen a small set of tem functions for performing the refinements and to show the potential of the techniques. Of those 10 cases, with simple regular expression matching, our MLP approach found four cases, where the highest ranked identifier-definiens pair was 'function' for at least one identifier in the formula. In these four cases, the distances of the content trees decreased to zero with all previously explained refinements enabled.

While this is just a first indication for the suitability of our approach, it shows that the long chain of processing steps shows promise. Therefore, we are actively working on the presented improvements and plan to focus on the task of learning how to generate

mappings from the input PL encoding to CL encoding without general rules for branch selection as we applied them so far.

6 Conclusion and future work

We make available the first benchmark dataset to evaluate the conversion of mathematical formulae between presentation and content formats. During the encoding process for our MathML-based gold standard, we presented the conceptual and technical issues that conversion tools for this task must address. Using the newly created benchmark dataset, we evaluated popular context-agnostic LaTeX-to-MathML converters. We found that many converters simply do not support the conversion from presentation to content format, and those that did often yielded mathematically incorrect content representations even for basic input data. These results underscore the need for future research on mathematical format conversions.

Of the tools we tested, LaTeXML yielded the best conversion results, was easy to configure, and highly extensible. However, these benefits come at the price of a slow conversion speed. Due to its comparatively low error rate, we chose to extend the LaTeXML output with semantic enhancements.

Unfortunately, we failed to develop an automated method to learn special notation. However, we could show that the application of special selection rules improves the quality of the content tree, i.e., allows choosing the most suitable tree from a selection of candidates. While the implementation of a few selection rules fixes nearly all issues we encountered in our test documents, the long tail of rules shows the limitations of a rule-based approach.

Future work

We will focus our future research on methods for automated notation detection, because we consider this approach as better suited and better scalable than implementing complex systems of selection rules. We will extract the considered notational features from the textual context of formulae and use them to extend our previously proposed approach of constructing identifier name spaces [19] towards constructing notational name spaces. We will check the integrity of such formed notational name spaces with methods comparable to those proposed in our previous publication [22] where we used physical units as a sanity check, if semantic annotation in the domain of physics are correct.

Acknowledgments. We would like to thank Abdou Youssef for sharing his Part-of-Math tagger with us and for offering valuable advice. We are also indebted to Akiko Aizawa for her advice and for hosting us as visiting researchers in her lab at the National Institute of Informatics (NII) in Tokyo. Furthermore, we thank Wikimedia Labs for providing cloud computing facilities and hosting our gold standard dataset. This work was supported by the FITWeltweit program of the German Academic Exchange Service (DAAD) as well as the German Research Foundation (DFG, grant GI-1259-1).

References

[1] A. Aizawa, M. Kohlhase, et al. NTCIR-11 math-2 task overview. In *Proc. 11th NTCIR Conf. on Evaluation of Information Access Technologies, Tokyo, Japan*, 2014.

[2] F. Cajori. *A History of Mathematical Notations*, vol. 1. Courier Corporation, 1928.

[3] H. S. Cohl, M. A. McClain, et al. Digital repository of mathematical formulae. In *Conference on Intelligent Computer Mathematics (CICM), Coimbra, Portugal*, pp. 419–422, 2014. doi:10.1007/978-3-319-08434-3_30

[4] H. S. Cohl, M. Schubotz, et al. Growing the digital repository of mathematical formulae with generic sources. In M. Kerber, J. Carette, et al., eds., *CICM, Washington, DC, USA*, vol. 9150, pp. 280–287, 2015. doi:10.1007/978-3-319-20615-8_18

[5] H. S. Cohl, M. Schubotz, et al. Semantic preserving bijective mappings of mathematical formulae between document preparation systems and computer algebra systems. In *CICM, Edinburgh, UK*, 2017. doi:10.1007/978-3-319-62075-6_9

[6] P. Dehaye, M. Iancu, et al. Interoperability in the OpenDreamKit project: The math-in-the-middle approach. In M. Kohlhase, M. Johansson, et al., eds., *CICM, Bialystok, Poland*, vol. 9791, pp. 117–131, 2016. doi:10.1007/978-3-319-42547-4_9

[7] M. England, E. S. Cheb-Terrab, et al. Branch cuts in Maple 17. *ACM Comm. Comp. Algebra* 48(1/2):24–27, 2014. doi:10.1145/2644288.2644293

[8] D. Ginev, H. Stamerjohanns, and M. Kohlhase. The LaTeXml daemon: Editable math on the collaborative web. In *LWA 2011, Magdeburg, Germany*, pp. 255–256, 2011.

[9] P. D. F. Ion and S. M. Watt. The global digital mathematical library and the international mathematical knowledge trust. In *CICM, Edinburgh, UK*, vol. 10383, pp. 56–69, 2017.

[10] N. Meuschke, M. Schubotz, et al. Analyzing mathematical content to detect academic plagiarism. In *Proc. CIKM*, 2017.

[11] B. Miller. LaTeXML: A LaTeX to XML converter. http://dlmf.nist.gov/LaTeXML/

[12] B. R. Miller and A. Youssef. Technical aspects of the digital library of mathematical functions. *Ann. Math. Artif. Intell.* 38(1-3):121–136, 2003. doi:10.1023/A:1022967814992

[13] M.-Q. Nghiem, G. Yoko, et al. Automatic approach to understanding mathematical expressions using mathml parallel markup corpora. In *26th Annu. Conf. Jap. Society for Artificial Intell.*, 2012.

[14] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.17 of 2017-12-22. F. W. J. Olver et al., eds.

[15] L. Padovani. On the roles of LaTeX and MathML in encoding and processing mathematical expressions. In A. Asperti, B. Buchberger, and J. H. Davenport, eds., *Mathematical Knowledge Management (MKM), Bertinoro, Italy*, vol. 2594, pp. 66–79, 2003. doi:10.1007/3-540-36469-2_6

[16] M. Pawlik and N. Augsten. RTED: A robust algorithm for the tree edit distance. *CoRR* abs/1201.0230, 2012. http://arxiv.org/abs/1201.0230

[17] M. Schubotz. Implicit content dictionaries in the NIST digital repository of mathematical formulae. Talk presented at the OpenMath workshop CICM, 2016. http://cicm-conference.org/2016/cicm.php?event=&menu=talks#03

[18] M. Schubotz. *Augmenting Mathematical Formulae for More Effective Querying & Efficient Presentation*. PhD thesis, TU Berlin, Germany, 2017. http://d-nb.info/1135201722

[19] M. Schubotz, A. Grigorev, et al. Semantification of identifiers in mathematics for better math information retrieval. In R. Perego, F. Sebastiani, et al., eds., *SIGIR, Pisa, Italy*, pp. 135–144, 2016. doi:10.1145/2911451.2911503

[20] M. Schubotz, L. Krämer, et al. Evaluating and improving the extraction of mathematical identifier definitions. In G. J. F. Jones, S. Lawless, et al., eds., *Conference and Labs of the Evaluation Forum (CLEF), Dublin, Ireland*, vol. 10456, pp. 82–94, 2017. doi:10.1007/978-3-319-65813-1_7

[21] M. Schubotz, N. Meuschke, et al. VMEXT: A visualization tool for mathematical expression trees. In *CICM, Edinburgh, UK*, pp. 340–355, 2017. doi:10.1007/978-3-319-62075-6_24

[22] M. Schubotz, D. Veenhuis, and H. S. Cohl. Getting the units right. In A. Kohlhase, P. Libbrecht, et al., eds., *Workshop and Work in Progress Papers at CICM 2016, Bialystok, Poland*, vol. 1785, pp. 146–156, 2016. http://ceur-ws.org/Vol-1785/W45.pdf

[23] M. Schubotz and G. Wicke. Mathoid: Robust, scalable, fast and accessible math rendering for Wikipedia. In *CICM, Coimbra, Portugal*, pp. 224–235, 2014. doi:10.1007/978-3-319-08434-3_17

[24] M. Schubotz, A. Youssef, et al. Challenges of mathematical information retrieval in the NTCIR-11 math Wikipedia task. In R. A. Baeza-Yates, M. Lalmas, et al., eds., *Special Interest Group on Information Retrieval (SIGIR), Santiago, Chile*, pp. 951–954, 2015. doi:10.1145/2766462.2767787

[25] M. Schubotz, A. Youssef, et al. Evaluation of similarity-measure factors for formulae based on the NTCIR-11 math task. In N. Kando, H. Joho, and K. Kishida, eds., *11th NTCIR, Tokyo, Japan*, 2014. http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings11/pdf/NTCIR/Math-2/04-NTCIR11-MATH-SchubotzM.pdf

[26] H. Stamerjohanns, D. Ginev, et al. MathML-aware article conversion from LaTeX. In *Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada*, pp. 109–120, 2009. http://eudml.org/doc/220017

[27] S. M. Watt. Exploiting implicit mathematical semantics in conversion between TeX and MathML. *Proc. Internet Accessible Math. Commun.*, 2002.

[28] A. Youssef. Part-of-math tagging and applications. In *CICM, Edinburgh, UK*, pp. 356–374, 2017. doi:10.1007/978-3-319-62075-6_25

◇ Moritz Schubotz
André Greiner-Petter
Philipp Scharpf
Norman Meuschke
Universitätsstraße 10
78464 Konstanz, Germany
moritz.schubotz , andre.greiner-petter , philipp.scharpf , norman.meuschke (at) uni-konstanz.de

◇ Howard S. Cohl
National Institute of Standards and Technology
Mission Viejo, CA 92694, U.S.A
howard.cohl (at) nist dot gov

◇ Bela Gipp
Universitätsstraße 10
78464 Konstanz, Germany
bela.gipp (at) uni-konstanz dot de

Dednat6: An extensible (semi-)preprocessor for LuaLaTeX that understands diagrams in ASCII art

Eduardo Ochs

1 Prehistory

Many, many years ago, when I was writing my master's thesis, I realized that I was typesetting too many natural deduction trees, and that this was driving me mad. The code (in `proof.sty`) for a small tree like this one

$$\dfrac{\dfrac{[a]^1 \quad a\to b}{b} \qquad b\to c}{\dfrac{c}{a\to c}\,1}$$

was this:

```
\infer[{1}]{ a\to c }{
 \infer[{}]{ c }{
  \infer[{}]{ b }{
   [a]^1 &
   a\to b } &
  b\to c } } }
```

This was somewhat manageable, but the code for bigger trees was very hard to understand and to debug. I started to add 2D representations of the typeset trees above the code, and I defined a macro `\defded` to let me define the code for several trees at once, and a macro `\ded` to invoke that code later:

```
%  [a]^1  a->b
%  -----------
%       b       b->c
%       ------------
%            c
%          ----1
%          a->c
%
%          ^a->c
%
\defded{a->c}{
 \infer[{1}]{ a\to c }{
  \infer[{}]{ c }{
   \infer[{}]{ b }{
    [a]^1 &
    a\to b } &
   b\to c } } }
%
$$\ded{a->c}$$
```

Then I realized that if I made the syntax of my 2D representations a bit more rigid, I could write a preprocessor that would understand them directly, and write all the '`\defded`'s itself to an auxiliary file. If a file `foo.tex` had this (note: I will omit all header and footer code, like `\begin{document}` and `\end{document}`, from the examples),

```
\input foo.dnt

%:  [a]^1  a->b
%:  -----------
%:       b       b->c
%:       ------------
%:            c
%:          ----1
%:          a->c
%:
%:          ^a->c

$$\ded{a->c}$$
```

then I just had to run "`dednat.icn foo.tex;latex foo.tex`" instead of "`latex foo.tex`".

2 `dednat.lua`

A few years after that, I learned Lua, fell in love with it, and ported dednat.icn from Icon — which was a *compiled* language — to Lua.

The first novel feature in `dednat.lua` was a way to run arbitrary Lua code from the `.tex` file being preprocessed, and so extend the preprocessor dynamically. `dednat.lua` treated blocks of lines starting with '`%:`' as specifications of trees, and blocks of lines starting with '`%L`' as Lua code. More precisely, the initial set of *heads* was `{"%:", "%L", "%D"}`, and `dednat.lua` processed each block of contiguous lines starting with the same head in a way that depended on the head.

The second novel feature in `dednat.lua` was a way to generate code for categorical diagrams, or "2D diagrams" for short, automatically, analogous to what we did for trees. I wanted to make the preprocessor write the '`\defdiag`'s seen here itself:

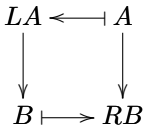
```
%   LA <-| A
%   |      |
%   v      v
%   B |-> RB
%
\defdiag{adj_L-|R}{
  \morphism(0,0)/<-|/<400,0>[LA`A;]
  \morphism(0,0)/->/<0,-400>[LA`B;]
  \morphism(400,0)/->/<0,-400>[A`RB;]
  \morphism(0,-400)/|->/<400,0>[B`RB;]
}
$$\diag{adj_L-|R}$$
```

where '`\morphism`' is the main macro in `diagxy`, Michael Barr's front-end for XY-pic.

After months of experimentation I arrived at a good syntax for 2D diagrams. This code:

```
%D diagram adj_L-|R
%D 2Dx     100    +25
%D 2D  100 LA <-| A
%D 2D      |      |
%D 2D      |      |
%D 2D      v      v
%D 2D  +25 B |-> RB
%D 2D
%D (( LA A <-|
%D    LA B -> A RB ->
%D    B RB |->
%D ))
%D enddiagram
%D
$$\diag{adj_L-|R}$$
```

generates this:



The lines with '`%D 2Dx`' and '`%D 2D`' define a grid with coordinates and nodes, and the lines between '`%D ((`' and '`%D ))`' connect these nodes with arrows.

2.1 A Forth-based language for 2D diagrams — low-level ideas

The article "Bootstrapping a Forth in 40 lines of Lua code" [1] describes how a Forth-like language can be reduced to a minimal extensible core, and bootstrapped from it. The most basic feature in [1] is "words that eat text"; the fact that Forth is a stack-based language is secondary — stacks are added later. The code for '`%D`'-lines is based on [1].

A "Forth" — actually the "outer interpreter" of a Forth, but let's call it simply a "Forth" — works on one line of input at a time, reads each "word" in it and executes it as soon as it is read. A "word" is any sequence of one of more non-whitespace characters, and an input line is made of words separated by whitespace. The "outer interpreter" of Forth does essentially this on each line, in pseudocode:

```
while true do
  word = getword()
  if not word then break end
  execute(word)
end
```

Note that `word` is a global variable. The current input line is stored in `subj` and the current position of the parser is stored in `pos`; `subj` and `pos` are also global variables — which means the `execute(word)` can change them!

The function `getword()` parses whitespace in `subj` starting at `pos`, then parses a word and returns it, and advances `pos` to the position after that word. There is a similar function called `getrestofline()` that returns all the rest of the line from `pos` onwards, and advances `pos` to the end of the line.

One of the simplest Forth words is '`#`' ("comment"). It is defined as:

```
forths["#"] = function ()
    getrestofline()
  end
```

It simply runs `getrestofline()`, discards its return value, and returns. We say that `#` "eats the rest of the line".

In a "real" Forth we can define words using '`:`' and '`;`', like this:

```
: SQUARE DUP * ;
```

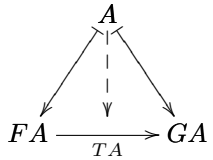
but the Forth-based language in `dednat.lua` is so minimalistic that we don't have '`:`' and '`;`' — we define words by storing their Lua code in the table `forths`.

2.2 A Forth-based language for 2D diagrams — code for diagrams

Let's look at an example. This code

```
%D diagram T:F->G
%D 2Dx     100 +20 +20
%D 2D  100     A
%D 2D         /|\
%D 2D        v v v
%D 2D  +30 FA --> GA
%D 2D
%D (( A FA |-> A GA |->
%D    FA GA -> .plabel= b TA
%D    A FA GA midpoint |->
%D ))
%D enddiagram
%D
$$\diag{T:F->G}$$
```

yields this:



The word `diagram` eats a word — the name of the diagram — and sets `diagramname` to it. The word `2Dx` eats the rest of the line, and uses it to attribute x-coordinates to some columns. The word `2D` also eats the rest of the line; when it is followed by *nnn* or *+nnn* that number gives the y-coordinate of that line, and the words that intersect a point that has both an x-coordinate and a y-coordinate become *nodes*. When a `2D` is not followed by an *nnn*

or $+nnn$ then this is a line without a y-coordinate, and it is ignored.

In a sequence like "`A FA |->`", both `A` and `FA` put nodes on the stack, and `|->` creates an arrow joining the two nodes on the top of the stack, without dropping the nodes from the stack. In a sequence like "`FA GA midpoint`" the `midpoint` creates a phantom node halfway between the two nodes on the top of the stack, drops (pops) them and pushes the phantom node in their place. The word `.plabel=` eats two words, a *placement* and a *label*, and modifies the arrow at the top of the stack by setting the arrow's label and placement attributes with them. The word '`((`' remembers the depth of the stack — 42, say — and the word '`))`' pops elements from the top of the stack; if the depth at '`))`' is 200 then '`))`' pops 200 − 42 elements to make the depth become 42 again.

The word `enddiagram` defines a diagram with the name stored in `diagramname`; each arrow that was created, even the ones that were dropped from the stack, becomes a call to `\morphism` — the main macro in `diagxy` — in the body of the diagram.

A good way to understand in detail how everything works is to inspect the data structures. Let's modify the code of the example to add some '`print`'s in '`%L`'-lines in the middle of the '`%D`'-code:

```
%D diagram T:F->G
%D 2Dx     100 +20 +20
%L print("xs:"); print(xs)
%D 2D  100     A
%D 2D         /|\
%D 2D        v v v
%D 2D  +30 FA --> GA
%L print("nodes:"); print(nodes)
%D 2D
%D (( A FA |-> A GA |->
%D    FA GA -> .plabel= b TA
%D    A FA GA midpoint -->
%L print("ds:"); print(ds)
%D ))
%L print("arrows:"); print(arrows)
%D enddiagram
```

The preprocessor outputs this on stdout:

```
xs:
{12=100, 16=120, 20=140}
nodes:
{ 1={"noden"=1, "tag"="A", "x"=120, "y"=100},
  2={"noden"=2, "tag"="FA", "x"=100, "y"=130},
  3={"noden"=3, "tag"="-->", "x"=120, "y"=130},
  4={"noden"=4, "tag"="GA", "x"=140, "y"=130},
  "-->"={"noden"=3, "tag"="-->", "x"=120, "y"=130},
  "A"={"noden"=1, "tag"="A", "x"=120, "y"=100},
  "FA"={"noden"=2, "tag"="FA", "x"=100, "y"=130},
  "GA"={"noden"=4, "tag"="GA", "x"=140, "y"=130}
}
ds:
12={"arrown"=4, "from"=1, "shape"="-->", "to"=5}
11={"TeX"="\\phantom{0}", "noden"=5, "x"=120,
    "y"=130}
10={"noden"=1, "tag"="A", "x"=120, "y"=100}
9={"arrown"=3, "from"=2, "label"="TA",
   "placement"="b", "shape"="->", "to"=4}
8={"noden"=4, "tag"="GA", "x"=140, "y"=130}
7={"noden"=2, "tag"="FA", "x"=100, "y"=130}
6={"arrown"=2, "from"=1, "shape"="|->", "to"=4}
5={"noden"=4, "tag"="GA", "x"=140, "y"=130}
4={"noden"=1, "tag"="A", "x"=120, "y"=100}
3={"arrown"=1, "from"=1, "shape"="|->", "to"=2}
2={"noden"=2, "tag"="FA", "x"=100, "y"=130}
1={"noden"=1, "tag"="A", "x"=120, "y"=100}
arrows:
{ 1={"arrown"=1, "from"=1, "shape"="|->", "to"=2},
  2={"arrown"=2, "from"=1, "shape"="|->", "to"=4},
  3={"arrown"=3, "from"=2, "label"="TA",
     "placement"="b", "shape"="->", "to"=4},
  4={"arrown"=4, "from"=1, "shape"="-->", "to"=5}
}
```

3 Semi-preprocessors

`dednat.icn`, `dednat.lua` and all its successors until `dednat5.lua` were preprocessors in the usual sense — they had to be run *outside* `latex` and *before* `latex`. With dednat6 this changed; dednat6 can still be run as a preprocessor, but the recommended way to run it on, say, `foo.tex`, is to put a line like

```
\directlua{dofile "dednat6load.lua"}
```

somewhere near the beginning of `foo.tex`, add some calls to `\pu` at some points — as we will explain soon — and compile `foo.tex` with `lualatex` instead of `latex`, to make `foo.tex` be processed "in parallel" by TEX and by Lua. That "in parallel" is a simplification, though; consider this example:

```
%:
%:  a  b
%:  ----
%:   c
%:
%:   ^my-tree
%:
$$\pu\ded{my-tree}$$
%:
%:  d  e  f
%:  -------
%:     g
%:
%:     ^my-tree
%:
$$\pu\ded{my-tree}$$
```

Suppose that this fragment starts at line 20. (As mentioned above, we are omitting the header and footer — e.g., `\begin{document}` and `\directlua{dofile "dednat6load.lua"}`.)

We have a `%:`-block from lines 20–26, a call to `\pu` at line 27, another `%:`-block from lines 28-34, and another call to `\pu` at line 35.

The output of the first `%:`-block above is a `\defded{my-tree}`, and the output of the second `%:`-block above is a *different* `\defded{my-tree}`.

'`\pu`' means "process until" — or, more precisely, *make dednat6 process everything until this point that it hasn't processed yet*. The first `\pu` processes the lines 1–26 of `foo.tex`, and "outputs" — i.e., sends to TEX — the first `\defded{my-tree}`; the second `\pu` processes the lines 28–34 of `foo.tex`, and "outputs" the second `\defded{my-tree}`. Thus, it is not technically true that TEX and dednat6 process `foo.tex` in parallel; dednat6 goes later, and each `\pu` is a synchronization point.

3.1 Heads and blocks

In order to understand how this idea — "semi-preprocessors" — is implemented in dednat6 we need some terminology.

The initial *set of heads* is `{"%:", "%L", "%D"}`. It may be extended with other heads, but we may only add heads that start with '`%`'.

A *block* is a set of contiguous lines in the current `.tex` file. This code

```
Block {i=42, j=99}
```

creates and returns a block that starts on line 42 and ends on line 99. The Lua function `Block` receives a table, changes its metatable to make it a "block object", and returns the modified table.

A *head block* is a (maximal) set of contiguous lines all with same head. Head blocks are implemented as blocks with an extra field `head`. For example:

```
Block {i=20, j=26, head="%:"}
```

A block is *bad* when it contains a part of a head block but not the whole of it. We avoid dealing with bad blocks — dednat6 never creates a block object that is "bad".

Each head has a *processor*. *Executing* a head block means running it through the processor associated with its head. Executing an arbitrary (non-bad) block means executing each head block in it, one at a time, in order. Note: the code for executing non-bad arbitrary blocks was a bit tricky to implement, as executing a '`%L`'-block may change the set of heads and the processors associated to heads.

A *texfile block* is a block that refers to the whole of the current `.tex` file, and that has an extra field `nline` that points to the first line that dednat6 hasn't processed yet. If `foo.tex` has 234 lines then the texfile block for `foo.tex` starts as:

```
Block {i=1, j=234, nline=1}
```

We saw in sections 1 and 2.2 that the "output" of a `%:`-block is a series of '`\defded`'s and the "output" of a `%D`-block is a series of '`\defdiags`'s. We can generalize this. For example, the "output" of

```
%L output [[\def\Foo{FOO}]]
%L output [[\def\Bar{BAR}]]
```

is

```
\def\Foo{FOO}
\def\Bar{BAR}
```

The *output* of a head block is the concatenation of the strings sent to `output()` when that block is executed. The output of an arbitrary (non-bad) block is the concatenation of the strings sent to `output()` by its head blocks when the arbitrary block is executed.

A `\pu`-*block* is created by dednat6 when a `\pu` is executed, pointing to the lines between this `\pu` and the previous `\pu`. If `foo.tex` has a `\pu` at line 27 and another at line 35 then the first `\pu` creates this block,

```
Block {i=1, j=26}
```

and the second `\pu` creates this:

```
Block {i=28, j=34}
```

As '`\pu`'s only happen in non-comment lines, `\pu`-blocks are never bad.

3.2 The implementation of `\pu`

The macro `\pu` is defined as

```
\def\pu{\directlua{
    processuntil(tex.inputlineno)
  }}
```

in LATEX, and `processuntil()` is this (in Lua):

```
processuntil = function (puline)
    local publock =
      Block {i=tf.nline, j=puline-1}
    publock:process()
    tf.nline = puline + 1
  end
```

Here's a high-level explanation. When dednat6 is loaded and initialized it creates a texfile block for the current `.tex` file — with `nline=1` — and stores it in the global variable `tf`. The macro `\pu` creates a `\pu`-block that starts at line `tf.nline` and ends at line `tex.inputlineno - 1`, executes it, and advances `tf.nline` — i.e., sets it to `tex.inputlineno + 1`.

The code above looks simple because the line `publock:process()` does all the hard work.

4 Creating new heads

New heads can be created with `registerhead`, and they are recognized immediately. For example, this

```
%L eval = function (str)
%L     return assert(loadstring(str))()
%L   end
%L expr = function (str)
%L     return eval("return "..str)
%L   end
%L
%L registerhead "%A" {
%L   name   = "eval-angle-brackets",
%L   action = function ()
%L       local i,j,str = tf:getblockstr()
%L       str = str:gsub("<(.-)>", expr)
%L       output(str)
%L     end,
%L }
%A $2+3 = <2+3>$
\pu
```

produces "$2 + 3 = 5$"; that looks trivial, but it is easy to write bigger examples of '`%A`'-blocks with `pict2e` code in them, in which the Lua expressions in '`<...>`'s generate '`\polyline`'s and '`\puts`'s whose coordinates are all calculated by Lua.

5 A read-eval-print-loop (REPL)

Dednat6 uses only one function from the LuaTEX libraries — `tex.print` — and two variables, `status.filename` and `tex.inputlineno`, but it includes a nice way to play with the other functions and variables in the libraries.

Dednat6 includes a copy of `lua-repl` (by Rob Hoelz, `github.com/hoelzro/lua-repl`), and we can invoke it by running `luarepl()`. If we put this in our `foo.tex`,

```
\setbox0=\hbox{abc}
\directlua{luarepl()}
```

then running `lualatex foo.tex` will print lots of stuff, and then the prompt '`>>>`' of the `lua-repl` inside dednat6; if we send these commands to the REPL,

```
print(tex.box[0])
print(tex.box[0].id,        node.id("hlist"))
print(tex.box[0].list)
print(tex.box[0].list.id,   node.id("glyph"))
print(tex.box[0].list.char, string.byte("a"))
print(tex.box[0].list.next)
print(tex.box[0].list.next.char,
                            string.byte("b"))
```

we get this in the terminal:

```
>>> print(tex.box[0])
<node    nil <  35981 >    nil : hlist 2>
>>> print(tex.box[0].id,        node.id("hlist"))
0 0
>>> print(tex.box[0].list)
<node    nil <   6107 >   6114 : glyph 256>
>>> print(tex.box[0].list.id,   node.id("glyph"))
29 29
>>> print(tex.box[0].list.char, string.byte("a"))
97 97
>>> print(tex.box[0].list.next)
<node   6107 <   6114 >  32849 : glyph 256>
>>> print(tex.box[0].list.next.char,
>>>>                             string.byte("b"))
98 98
>>>
```

The best way to use `luarepl()` — in my not so humble opinion — is from Emacs, with the `eev` library. The tutorial of eev at

`http://angg.twu.net/eev-intros/find-eev-quick-intro.html`

explains, in the section "Controlling shell-like programs", how we can edit the commands to be sent to `lualatex` in a buffer, called the "notes buffer", and send them line by line to another buffer that runs `lualatex foo.tex` in a shell — the "target buffer"; each time that we type the F8 key Emacs sends the current line to the program running in the target buffer, *as if the user had typed it*.

6 Availability

Dednat6 is not in CTAN yet (as of October, 2018). Until it gets there you can download it from:

`http://angg.twu.net/dednat6.html`

References

[1] E. Ochs: *Bootstrapping a Forth in 40 Lines of Lua Code*. Chapter 6 (pp. 57–70) of *Lua Programming Gems*, L.H. de Figueiredo, W. Celes, and R. Ierusalimschy, eds. `lua.org/gems`, 2008. Available from `http://angg.twu.net/miniforth-article.html`.

◇ Eduardo Ochs
`eduardoochs (at) gmail dot com`
`http://angg.twu.net/dednat6.html`

Managing forlorn paragraph lines (a.k.a. widows and orphans) in LaTeX

Frank Mittelbach

Contents

1 The name of the game

Splitting off the first or last line of a paragraph at a page or column break is considered bad practice in typesetting circles. It is thus not surprising that the craftspeople have come up with fairly descriptive names for such lines when they appear in typeset documents.

Commonly used are the terms "widow" for the last and "orphan" for the first line. These are, for example used in English, French ("veuve" and "orpheline"), Italian ("Vedova" and "Orfano"), Spanish ("línea huérfana" and "línea viuda"), or to a lesser extent in German ("Witwe" and "Waise").

One way to remember them is to think of orphaned lines appearing at the start (birth) and widows near the end (death) of a paragraph or by using Bringhurst's mnemonic, "An orphan has no past; a widow has no future" [6].

German typesetters coined some more profane descriptions by calling the widow line a "Hurenkind" (child of a whore) and the orphan line a "Schusterjunge" (son of a shoemaker) allegedly because these boys have been notoriously meddlesome. For German practitioners these are still the predominantly used terms, though "Witwen" and "Waisen" are also well understood. Dutch uses "hoerenjong" and "weeskind" which translates to son of a whore and orphan, i.e., somewhere in between the German usage and the other languages.

Don Knuth catered for this typographic detail in the TeX program by providing parameters whose values are used as penalties if the pagination algorithm considers breaking in such a place. Widow lines are penalized via `\widowpenalty`; however, orphans are not controlled by `\orphanpenalty` as one might expect, but by a parameter named `\clubpenalty`.

There have been some queries about this choice of names on Stack Exchange and after a little Internet searching I found a listing for "club line" in the Collins English Dictionary Digital Edition [1], listing it as a British (!) term used in printing for an orphan line. Further checks through the first dozen or so pages of google hits for "club line" by its own, and the same with additional restrictions such as "printing" or "typography" revealed a handful of additional references (two of which mentioned that Knuth used the term — thus circular references). So on the whole it was a meager result and all except one indicated a use of the term in British not American English.

Contrast this with a search for "orphan line" in google: Now we will find that nearly all the results in the first five pages are relevant, with only two or three near the end being unrelated.

However, what we also see from following them up is that about a third of them give the terms different meanings, either by swapping the definition of orphan and widow lines or by giving them a slightly different meaning altogether: The last line of a paragraph being nearly empty, i.e., containing only a single word or even part of a single word.[1]

Most of the time, though, the forlorn lines we want to deal with are called widows and orphans and this is what we will call them in the remainder of the article, even if we have to set a `\clubpenalty` to deal with one of them.

2 The problem

Essentially everyone in typography circles agrees that widows and orphans are very distracting to the reader as well as a sign of bad craftsmanship, and should therefore be avoided. In fact, most writing guides and other books on typography generally suggest that a document should have no such lines whatsoever, e.g., in older editions of the Chicago Manual of Style [2] we find "A page should not begin with the last line of a paragraph unless it is full measure and should not end with the first line of a new paragraph."

However, that is easier said than done, so in a newer edition of that guide [4] we now find "A page should not begin with the last line of a paragraph unless it is full measure. (A page can, however, end with the first line of a new paragraph.)" instead.

As a result of this sort of guidance many journal classes for LaTeX completely forbid widows and orphans by setting `\widowpenalty` and `\clubpenalty` to `10000` which prohibits a break at such points — *TUGboat* being no exception.

[1] ... as demonstrated here. In TeX this kind of typographical issue can also be dealt with, although by different means and somewhat more manually: The parameter `\finalhyphenpenalty` can make hyphenation in the last line unattractive, and using unbreakable spaces will ensure that there is more than one word in the last line; `\parfillskip` can also help.

But doing this introduces severe problems: As LaTeX (and in fact all major typesetting systems to date) use a greedy algorithm to determine the pagination of a document, it will recognize problems with orphan or widow lines late in the game and will have only the current page to work with. This means the best it can do to avoid the situation is to push an orphan to the next page if there is not enough room to squeeze in another line. The same happens with widows; here LaTeX is forced to move the second-last line to the next page even though it would still nicely fit.

As a result the current page will have an additional line-height worth of white space that needs to be distributed somewhere on the page. If there are headings, displays, lists or other objects for which the design allows some flexibility in the surrounding white space, then this extra space may not create much of an issue. If, however, the page consists only of text or objects without any flexibility, then all LaTeX can do is run the page or column short, generating a fairly ugly hole at the bottom.[2]

Related problems

Besides widows and orphans there are a number of similar issues that typography manuals mandate eliminating if at all possible. One is a paragraph split across pages at a hyphenation point so that only a part of the word is visible at any time; another is a widow line with a following display formula. For both, TeX offers parameters to control the undesirability of the scenario. By default Don Knuth considered them of lesser importance and provided default values of `100` and `50` for `\brokenpenalty` and `\displaywidowpenalty`, respectively while he specified `150` for orphans and widows. However, if your style guide (or your class file) wants to avoid them at all cost then you are in precisely the same situation as with widows and orphans discussed above.

A special variation of the last issue is a display formula starting a page, that is, with the introductory material completely on the previous page or column. That is considered a no-go by nearly everybody so in TeX the controlling `\predisplaypenalty` parameter has by default a value of `10000`. But again, there may be valid reasons to ignore this advice in a special situation, e.g., when the space constraints are high.

3 Fixing the problem

The alternative to preventing widows and orphans (or hyphens across page boundaries, etc.) automatically and at all costs is to manually resolve the issues when they arise. For this one finds a number of suggestions in the typography literature; a good collection is given in the guidelines section of the Wikipedia page on "Widows and Orphans" [5]. We will look at them one by one below and discuss their applicability and possible implementation in a LaTeX document.

- ▶ *Forcing a page break early, producing a shorter page*

This is what LaTeX and most other typesetting tools automatically do if you completely forbid widows and orphans and if often leads to badly filled pages as discussed above.

However, if you force the page break manually, you can lessen the impact by also explicitly forcing earlier breaks and thereby shifting the extra whitespace to a page or column where it can be absorbed by the available flexibility on that page.

- ▶ *Adjusting the leading, the space between lines of text (although such carding or feathering is usually frowned upon)*

That is indeed frowned upon and for good reason. The human eye is tuned to notice even small differences in the vertical spacing of lines and across columns or pages such changes, even if they are small, are very noticeable and distracting. Besides, managing such a change in a LaTeX document would be, while possible, quite cumbersome, so this is not particularly useful advice for us.

- ▶ *Adjusting the spacing between words to produce 'tighter' or 'looser' paragraphs*

This is certainly a practical option if you choose the right paragraph or paragraphs, e.g., those that are somewhat longer and that have a last line that is either nearly full (for lengthening) or nearly empty (for shortening). In that case squeezing the word spaces might result in one line less and extending it might get you an additional line (with just a word or two). In many cases the resulting gray value is still of acceptable quality so this is a typical trick of the trade. In LaTeX this is achieved by using the `\looseness` parameter that is discussed in Section 4.

Note that you do not necessarily need to manipulate one of the paragraphs of the problem page; there might be a better candidate on an earlier page.

- ▶ *Adjusting hyphenation within the paragraph*

This is a variation of the "change the number of paragraph lines" type of approach. However, given that LaTeX is usually good in considering most of the possible hyphenation points when breaking paragraphs, one is unlikely to gain much if anything. Thus, this suggestion might have some merits in a system that

[2] This can be observed on the first page of this article, where an orphan line was pushed onto the current page. Fortunately, the resulting hole is partly masked by the footnote.

does not hyphenate well (or at all) but with LaTeX one is better off applying `\looseness`.

If you have a very badly broken paragraph because of a missed hyphenation point you should fix that anyway (by adding `\-` or a `\hyphenation` exception) regardless of whether or not there is a widow or orphan nearby.

► *Adjusting the page's margins*

Now this is an interesting approach. In contrast to changes in `\baselineskip` small changes in the line width are virtually undetectable without a ruler — unless you make it a huge change. Unfortunately, it is not really an option when using LaTeX as the underlying TeX engine essentially assumes a fixed line width throughout, so it is fairly difficult to change that at arbitrary places.

In other words, this advice is really geared towards interactive systems where you can change the width of a text region and get an immediate reflow as a visual feedback.

► *Subtle scaling of the page, though too much non-uniform scaling can visibly distort the letters*

In principle, that would be possible with LaTeX though so far nobody has implemented the necessary changes to the output routine. That is, when a column or page runs short it will be scaled to the right height before attaching headers and footers. Instead of a non-uniform scaling, one could do uniform scaling and adjust the horizontal widths of headers and footers accordingly.

"An orphan has no past; a widow has no future"

However, that approach has issues already discussed: Scaling means we get a different leading (though this time the characters will also grow). And if we do non-linear scaling, i.e., only vertically, then we will distort characters and from a certain point onwards this will also be noticeable. So it is questionable whether this will actually improve the situation.

► *Rewriting a portion of the paragraph*

This is obviously something you can do only if you are the author and not typesetting some text written by others. But if so, it is a valid strategy since it enables you to easily shorten or enlarge a paragraph so that your orphan or widow is reunited with other lines.

Again, there is no requirement to do this rewrite with the paragraph causing the issue (as implied by the advice); you can choose any[3] earlier paragraph to achieve the desired effect.

[3] Well, "any" is an exaggeration: If you change a paragraph on an earlier page the gained (or extra) space might get swallowed up by available flexibility on some intermediate page and your widow or orphan thus stays put.

► *Reduce the tracking of the words*

Tracking in this context means adjusting the spacing between characters in a uniform way (in contrast to kerning, which means adjusting the spacing between individual glyph pairs, e.g., "AV" cf. "AV").

Figure 1 shows a line of text with different amounts of tracking (negative and positive) applied. Clearly by applying tracking one can shorten or lengthen a text. However, when comparing the lines side by side it is also obvious that the gray value of words changes fairly rapidly too. Thus even with small tracking values, changes may become noticeable and thus distracting. To illustrate the point this article contains one manipulated paragraph; see if you can spot it — perhaps it was on an earlier page and you thought: hmm that doesn't look quite right.[4]

On the whole, common typographical advice is to *not use* tracking for such purposes or, if there is no better alternative, then only with very small tracking values in which case there may not be any noticeable effects on the paragraph length unless you are lucky. It is possible to experiment in LaTeX if you load the microtype package and use, for example, `\textls`.

► *Adding a pull quote to the text (more common for magazines)*

Pull quotes are catch phrases from the text that are "pulled out" and typeset prominently again in a different place, typically in a larger and often different font. They serve as eye catchers and if carefully chosen will give the reader a preview of the content or main points of an article.

The design needs to clearly distinguish them from other display material, e.g., there should be no way to confuse them with headings, etc. In two-column texts this is often done by placing them in a window with both columns flowing around them (as shown on this page), but placing them into the content of one column is also often done.

Placing them within a single column is fairly easy in TeX, all you need to do is to define an environment that places the material between paragraph lines (using `\vadjust` if used inside paragraph text).

Producing pull quotes with the column texts flowing around it, is more manual work and fairly cumbersome, but doable. On the present page, we used the wrapfig package. The approach, as well as a few others are discussed in answers to a question on Stack Exchange [3].

[4] The answer is given at the end of the article.

Tracking is the uniform increase or decrease of spacing between glyphs.	−0.05 em
Tracking is the uniform increase or decrease of spacing between glyphs.	−0.03 em
Tracking is the uniform increase or decrease of spacing between glyphs.	−0.02 em
Tracking is the uniform increase or decrease of spacing between glyphs.	−0.01 em
Tracking is the uniform increase or decrease of spacing between glyphs.	— LaTeX's default setting —
Tracking is the uniform increase or decrease of spacing between glyphs.	+0.01 em
Tracking is the uniform increase or decrease of spacing between glyphs.	+0.02 em
Tracking is the uniform increase or decrease of spacing between glyphs.	+0.05 em
Tracking is the uniform increase or decrease of spacing between glyphs.	+0.07 em

Figure 1: Tracking in action

As the above advice already mentions, these are more commonly found in magazine type documents, so this approach may or may not be applicable.

► *Adding a figure to the text, or resizing an existing figure*

Just like adding a pull quote, resizing a figure will obviously change the amount of material a column can hold and thus will enable us to move an orphan or widow out of harm's way. Whether or not it is a valid option depends on the figure in question; often enough graphics do offer some freedom and can be adjusted either by scaling (up or down) or by cropping, etc.

Summary

To resolve issues with widows and orphans one has to somehow adjust the amount of material typeset in the respective column or page. For LaTeX users the most promising approaches are

- forcing material from one column to the next through explicit page breaks;
- generating more or less material by lengthening or shortening some paragraphs;
- rewriting paragraphs (if you are the author);
- or resizing a float.

Adjusting the line width for a single column is rather difficult to achieve in LaTeX and therefore not recommended, even though it can lead to good results. Applying tracking seldom works well, it usually either makes no difference or results in noticeable grey-level differences.

Using pull quotes is similar to changing or resizing floats or modifying paragraphs. However, the quotes carry meaning and so you can't simply add one arbitrarily for the sake of better pagination. Thus, adding or moving them around is a bit like changing the document structure and you therefore have to be careful not to sacrifice semantics for form. This makes them a less desirable approach.

Scaling the page or changing the leading is typographically rather questionable, so these can't be recommended (besides their being rather complicated to achieve with LaTeX).

In any case, it should be noted though that all approaches are manual and thus the adjustments will become invalid the moment there is a document change that modifies the amount of material typeset. It is therefore of paramount importance to manually fix widows and orphans only at the very last stage of producing the final document. Otherwise all the effort might be in vain and will need to be undone or changed over and over again.

The situation would be somewhat different if TeX was extended to globally optimize pagination rather than applying a greedy algorithm as it currently does. Some theoretical work in that direction has been carried out in recent years by the current author and it may eventually lead to a production-ready system using LuaTeX [7, 8]. However, at present it is available only in a private prototype implementation and can't be used with vanilla LaTeX.

4 Using TeX's `\looseness` approach

TeX (and therefore LaTeX) uses a globally optimizing line-breaking algorithm to find the best breaks for a given paragraph based on a given set of parameters. One can ask TeX to try to find a solution (within given quality boundaries) that is a number of lines longer or shorter than the optimal result. If such a solution exists it will be used; if not, then TeX will try to match the request as closely as possible.

The paragraph will still be optimized (under the new conditions), i.e., its overall gray level will be fairly uniform, etc., but, inevitably, the inter-word spacing will get looser or tighter in the process.

To activate this feature you need to set the parameter `\looseness` to the desired value. This has to be done directly in front of (or within) the paragraph text via low-level TeX syntax

```
\looseness=1    % to lengthen by one line
%               % <- no blank line here!
The text that gets manipulated ...
```

as there is no LaTeX interface available.

TEX automatically resets the value to zero whenever a `\par` command or blank line is encountered; thus it will affect at most one paragraph.

A value of `-1` has the best chance to work if the last line is already nearly empty (and the paragraph is of reasonable length). Lengthening is somewhat easier as inter-word spaces can stretch arbitrarily (as long as they do not exceed the `\tolerance`), whereas they can shrink by only a fixed amount. But again there is a better chance for success if the last line is already (nearly) filled.

So far so easy, but there are a few pitfalls that need to be avoided: First of all, with a positive value of `\looseness` TEX will usually move only a single word or even part of a single word into the last line, as this way there is more material in the others and thus less stretching of the inter-word spaces necessary. As this usually looks rather ugly, it is best to tie the last words together by using `~` and if necessary prevent hyphenation of the last word by placing an `\mbox` around it.

Secondly, lengthening of a paragraph may go horribly wrong (as shown here) if the document is set with a high `\tolerance` value, e.g., most definitely when a `\sloppy` declaration is in force.

The `\tolerance` defines how bad a line can get while still being a candidate for the line-breaking algorithm and `\sloppy` (or `sloppypar`) simply sets this tolerance nearly[5] to infinity, i.e., arbitrarily bad lines are acceptable and thus you might end up with a paragraph like this one (where we asked for two extra lines and got them). With a lower `\tolerance` value that would never have happened: TEX would have refused to produce a result like this.

Seeing the previous paragraph you might ask yourself why one would want to use a high, let alone infinite, `\tolerance` at all. The reason is that this caters for situations where line breaking is very difficult. If there is a better solution with a lower tolerance value TEX would use it, but if not it could still proceed. So normally we wouldn't see such bad paragraphs even with a high tolerance in force (it just means that TEX evaluates more candidate solutions). But if we apply `\looseness` we explicitly ask TEX to deviate from the optimal number of lines and to fulfill this request TEX may resort to a solution with bad lines that nobody would want.

[5] In the early days of LATEX `\sloppy` used to set the tolerance to `10000` (i.e., TEX's infinity) but that tended to produce even more bizarre looking paragraphs: TEX then made one line really, really bad and all others perfect, as that looked to the optimizer to be the best solution.

5 Identifying pages with widows or orphans

If the document class you use sets `\widowpenalty` and `\clubpenalty` to `10000`, then LATEX will automatically prevent widows and orphans, i.e., an orphan is forced to the top of the next page or column; and the same with the line preceding a widow. The downside, as discussed previously, is partly empty pages and if space is a premium (for example, if your conference paper is not allowed to be more than X pages in total) then this is a possible problem. Thus you are better off allowing widows and orphans (by changing the parameter values) and manually correcting them in one way or another.

The question then becomes, how do you identify the problematic page breaks without manually going through the printout of your document and searching for them? While that is certainly an option it is error prone and it would be much nicer if LATEX (even if it can't automatically resolve the issues for you) at least identifies them so that you only have to check the problem pages.

This is possible by simply loading the package widows-and-orphans.[6] This package adjusts the parameter values slightly so that so that all possible combinations lead to distinctive numbers. For example, instead of the LATEX default values it would choose

```
\widowpenalty        = 150
\clubpenalty         = 152
\displaywidowpenalty =  50
\brokenpenalty       = 101
```

so that it can distinguish between a widow and an orphan (`\widowpenalty` or `\clubpenalty`) or a display widow that comes together with a hyphen at the break (`\displaywidowpenalty` + `\brokenpenalty`). In case you wonder why `151` wasn't used: that value is already used by LATEX for `\@medpenalty` which you get if you issue `\nopagebreak[2]`. By making sure that all technically possible combinations lead to unique numbers it is only necessary to look at the penalty of the page break to determine whether or not that break exhibits one or more of the problems. So at any page or column break the `\outputpenalty` is inspected and depending on the findings a warning or error is generated that can then be checked and corrected manually.

To ease this process further the package has a number of key/value options. The `check` option determines how findings are handled: the default

[6] The implementation of the package (which is written in the expl3 programming language) is documented in a separate article in this *TUGboat* issue [9].

is `warning` in which case warnings are written to the terminal and the `.log` file. In the last phase of document development you may want to change that to `error` in which case the package will stop at each problem with an error message rather than just a warning. In the opposite direction, `info` will not clutter the terminal with messages and only writes to the `.log`. And if you know for sure that all the remaining issues have to stay, you can also use the value `none` in which case no checks are done at all.

Why would one want the last option instead of not loading the package? The reason is this: as the package has to change the parameter settings slightly, not loading it would mean running the document with different values and even though those changes are minimal, it is possible to construct examples where the difference matters and leads to changed results. So once you have fixed what is possible to fix it's safest to still load the package, even if you no longer want the remaining warnings. Another reason is that the package offers the command `\WaOsetup` that allows you to change options mid-document, e.g., turn the warnings off for chapters already handled, but turn them on again for others.

Instead of suppressing all checking for a part of the document via `\WaOsetup` you can issue the command `\WaOignorenext` somewhere in the document, after which the next check — for the current page or column — will be silenced. The check is still performed and if no problems are found you will receive an error message, because either you have added it to the wrong page or your text has changed and it is no longer needed.

The package also offers options to set individual parameters to "reasonable" values. These are `widows`, `orphans`, `hyphens` that all accept `default` (LaTeX default), `avoid` (higher value but still possible) or `prevent`. And there are also the valueless options `default-all`, `avoid-all` and `prevent-all` to set all parameters in one go. Of course, as an alternative one can always change the parameters individually in the preamble or even in the middle of the document by assigning explicit numerical values.

If you want to see the resulting parameter settings (and the combinations that need to be unique in order to allow the package to work) you can issue the command `\WaOparameters` at any point after the preamble, which will give you a somewhat terse listing.

Answer to the riddle

The paragraph with negative tracking (−0.01 em) is the first one after "▶ Rewriting a portion . . . " on page 248, toward the bottom of the first column.

Due to the tracking it needs one line less compared to the default line breaks. But as a result of the tracking, the characters are noticeably closer to each other, for example, in the word "obviously". Depending on your aesthetic judgment, a value of ±0.02 em is roughly the borderline of what can be considered acceptable, so if that or a lower value works, it might be an option.

References

[1] Anonymous. Collins English dictionary — complete & unabridged 2012 digital edition. `https://www.collinsdictionary.com/dictionary/english/club-line`.

[2] Anonymous. *The Chicago Manual of Style*. University of Chicago Press, Chicago, IL, USA, 14th edition, 1993.

[3] Anonymous. How can you create pullquotes?, 2012. `https://tex.stackexchange.com/questions/45709/how-do-you-create-pull-quotes`.

[4] Anonymous. *The Chicago Manual of Style*. University of Chicago Press, Chicago, IL, USA, 17th edition, 2017.

[5] Anonymous. Widows and orphans, 2017. `https://en.wikipedia.org/wiki/Widows_and_orphans`.

[6] Robert Bringhurst. *The Elements of Typographic Style*. Hartley & Marks Publishers, Point Roberts, WA, USA and Vancouver, BC, Canada, 1992.

[7] Frank Mittelbach. A general framework for globally optimized pagination. In *Proceedings of the 2016 ACM Symposium on Document Engineering*, DocEng '16, pages 11–20, New York, NY, USA, 2016. ACM. Download from `https://www.latex-project.org/publications`.

[8] Frank Mittelbach. Effective floating strategies. In *Proceedings of the 2017 ACM Symposium on Document Engineering*, DocEng '17, pages 29–38, New York, NY, USA, 2017. ACM. Download from `https://www.latex-project.org/publications`.

[9] Frank Mittelbach. The widows-and-orphans package. *TUGboat* 39:3, 2018, 252–262. `https://ctan.org/pkg/widows-and-orphans`

◇ Frank Mittelbach
Mainz, Germany
`frank.mittelbach (at) latex-project dot org`
`https://www.latex-project.org`

The widows-and-orphans package

Frank Mittelbach

Abstract

The widows-and-orphans package checks page or column breaks for issues with widow or orphan lines and issues warnings if such problems are detected. In addition, it checks and complains about breaks involving hyphenated words and warns about display formulas directly after a page break — if they are allowed by the document parameter settings, which by default isn't the case.

A general discussion of the problem of widows and orphans and suggestions for resolution is given in [1].

Contents

1 Overview

To determine if a widow or orphan has occurred at a column or page break, the package analyzes the `\outputpenalty` that triggered the break. As TEX adds special penalties to widow and orphan lines (`\widowpenalty` and `\clubpenalty`), we can hope to identify them, provided the penalties have unique values so that we don't end up with false positives.

The package therefore analyzes the different parameter values and if necessary adjusts the settings slightly so that all possible combinations that can appear in documents have unique values and can thus be identified.

All that remains is to hook into the output routine check; if `\outputpenalty` has a value that matches one of the problematic cases, issue a warning.

Besides widows and orphans it is possible to detect other cases controlled through penalty parameters, e.g., `\brokenpenalty` that is added if a line ends in a hyphen. So by including this parameter into the checks, we can identify when that happens at the end of a column and issue a warning there too.

We also do this for `\predisplaypenalty`, which controls a break just in front of a math display. This is normally set to `10000` so such breaks don't happen in standard LATEX, but if the value is lowered it becomes possible, and thus a possible issue.

1.1 Options

The package has a number of key/value options to adjust its behavior. The option `check` defines what happens when an issue is found: default is `warning`, other possibilities are `error`, `info` and `none`.

The options `orphans` and `widows` set reasonable parameter values; the default is to use whatever the class defines. Possible values are `prevent`, `avoid` or `default`, the latter meaning use standard LATEX defaults.

To set all parameters in one go you can use `prevent-all`, `avoid-all` or `default-all`. These options also assign values to `\brokenpenalty` and `\predisplaypenalty`.

1.2 User commands

The package provides three user-level commands.

\WaOsetup `\WaOsetup` ⟨*comma list*⟩

This command accepts any of the package options and allows adjusting the package behavior in mid-document if necessary.

\WaOparameters `\WaOparameters`

This command produces a listing of the parameter combinations and their values.

\WaOignorenext `\WaOignorenext`

This command directs the package to not generate warnings for the current page or columns (if we know that they can't be corrected).

1.3 Related packages

Package nowidow: This package offers some commands to help pushing lines from one page to another by locally requesting no widows or orphans — possibly for several lines. In that respect it implements one of the possibilities discussed in the *TUGboat* article [1]. This is, however, in many cases not the best solution to get rid of a widow or orphan when the interest is to achieve high typographical quality.

2 The implementation

The package is implemented in expl3, so the first thing we do is to define a prefix for our internal commands, so that we can write `@@` in the source when we mean the prefix `__fmwao`, indicating that something is internal.[1]

```
1 ⟨@@=fmwao⟩
```

Then we check that we are running on top of LaTeX 2_ε and load the two packages we want to use: xparse for the user interface and l3keys2e for key/value option syntax. They load the needed expl3 code so we are ready to roll afterwards.

```
2 \NeedsTeXFormat{LaTeX2e}    \RequirePackage{xparse,l3keys2e}
```

Then we announce the package to the world at large. This declaration will also tell LaTeX that this is an expl3 package and that from now on the expl3 conventions are to be obeyed, e.g., `_` and `:` can appear in command names and whitespace is automatically ignored (so no need for `%` all over the place).[2]

```
3 \ProvidesExplPackage{widows-and-orphans}{2018/11/18}{v1.0b}
4                     {Detecting widows and orphans (FMi)}
```

\@makecol As mentioned in the introduction we want to check the value of `\outputpenalty` inside the output routine, and so we need to add some code to the output routine.

We add it to the front of `\@makecol`, the macro that assembles the actual column. This way it only gets executed if we make a column or a page but not when the output routine is triggered because of a float or `\marginpar`.

```
5 \tl_put_left:Nn \@makecol { \__fmwao_test_for_widows_etc: }
```

(*End definition for* `\@makecol`.)

[1] `l3docstrip` expands that for us, so in the `.sty` file we get the longer names with double underscores even though the real source just contains `@@` all over the place. The same is done by the `l3doc` class for the printed documentation you are currently reading.

[2] Figure 1 gives a short introduction to the naming conventions of expl3 for those readers who haven't seen any code written in expl3. For more details refer to [2].

Commands in expl3 use the following naming convention:

```
\⟨module⟩_⟨action⟩:⟨arg-spec⟩    % externally available command
\__⟨module⟩_⟨action⟩:⟨arg-spec⟩  % internal command local to the package
```

⟨*module*⟩ describes the (main) area to which the command belongs,

⟨*action*⟩ describes the (main) action of the command, and

⟨*arg-spec*⟩ shows the expected command arguments and their preprocessing:
`N` means expect a single token;
`n` means expect a (normal) braced argument;
`T` and `F` also represent braced arguments, indicating "true" and "false" branches in a conditional;
`V` means expect a single token, interpret it as a variable and pass its value on to the command;
Finally, `p` stands for a (possibly empty) parameter spec, e.g., `#1#2...` in a definition.
There are a number of other argument types, but they aren't used in the code described here.

Examples:

`\cs_if_eq:NNTF` is a conditional from the module `cs` (command names) implementing the action `if_eq` (if equal) and expecting two single tokens (commands to compare) and a true and a false branch (one of which is executed).

`\tl_put_left:Nn` is a function from the module `tl` (token lists) implementing `put_left` and expecting a single token (a token list variable) and a braced argument (the data to insert at the front/left in the variable).

Variables start with `\l_` (for local) or `\g_` (for global) and have the data type as the last part of the name. Variables internal to the package use two underscores, e.g., `\l__fmwao_gen_warn_bool`.

Figure 1: Crash course in expl3 command name conventions

2.1 Checking `\outputpenalty`

`\g__fmwao_gen_warn_bool`

To be able to suppress checking we define a global boolean variable which by default is set to `true` (warnings enabled).

```
\bool_new:N \g__fmwao_gen_warn_bool
\bool_gset_true:N \g__fmwao_gen_warn_bool
```

(*End definition for* `\g__fmwao_gen_warn_bool`.)

`\__fmwao_test_for_widows_etc:`

What are the different values related to orphans and widows and the like that can appear in `\outputpenalty`? Here is the basic list:

`\widowpenalty` + `\interlinepenalty` → if the break happens on the second-last line of a paragraph and the paragraph is not followed by a math display.

`\displaywidowpenalty` + `\interlinepenalty` → if the break happens on the second-last line of a paragraph and the paragraph is followed by a math display.

`\clubpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph and the paragraph has more than two lines.

`\clubpenalty` + `\widowpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph in a two-line paragraph (thus this line is also the second-last line).

`\clubpenalty` + `\displaywidowpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph in a two-line paragraph and a math display follows.

That's it for widows and orphans. If we also consider hyphenated breaks then we get a further set of cases, namely all of the above with `\brokenpenalty` added in and the case of `\brokenpenalty` on its own (with just `\interlinepenalty` added).

```
\cs_new:Npn \__fmwao_test_for_widows_etc: {
```

So here is the main test. We compare `\outputpenalty` with each case until we have a hit or run out of cases. Instead of adding `\interlinepenalty` to each case we subtract it once from `\outputpenalty` to make the comparison a little more efficient.

If we get a hit, we execute the corresponding code: either `\__fmwao_problem_identified:n` or `\__fmwao_problem_identified:nn` (i.e., one or two arguments) to issue the warning. The arguments are to select the correct warning text and customize it further if necessary. For example, in the first case it is a "widow" problem and the text in the message should start with "`Widow`" while in the second case it is also a "widow" problem but the text will say "`Display~ widow`".[3] This just saves a bit of space when different cases share more or less the same message text.

```
  \int_case:nnF { \outputpenalty - \interlinepenalty }
    {
      { \widowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Widow} }
      { \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Display~ widow} }
      { \clubpenalty }
        { \__fmwao_problem_identified:n{orphan} }
      { \clubpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{} }
      { \clubpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{display} }
```

A similar issue comes from a hyphen at the end of a column or page in which case TEX adds `\brokenpenalty` so we can test against that:

```
      { \brokenpenalty }
        { \__fmwao_problem_identified:n{hyphen} }
```

However, I said "TEX adds", which means if a widow line also ends in a hyphen then the penalty will be the sum of both individual penalties. So all the cases above need to be repeated with `\brokenpenalty` added to the value. We generate the same warnings, though — e.g., we will say "Widow detected" and not "Hyphenated widow line detected" as the latter seems to be overkill.

```
      { \brokenpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Widow} }
      { \brokenpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Display~ widow} }
      { \brokenpenalty + \clubpenalty }
        { \__fmwao_problem_identified:n{orphan} }
      { \brokenpenalty + \clubpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{} }
      { \brokenpenalty + \clubpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{display} }
```

Finally there is `\predisplaypenalty` that we may as well check also (in case it was set to a value lower than `10000`). If it appears it means we have a display at the very top of the page. We reuse the "widow" warning but this time say "`Lonely~ display`" in the second argument. This case does not have `\interlinepenalty` added by TEX, so we have to undo the optimization above.

```
      { \predisplaypenalty - \interlinepenalty }
        { \__fmwao_problem_identified:nn{widow}{Lonely~ display} }
    }
```

[3] If you haven't seen much expl3 code you may wonder about the `~`. As the code ignores spaces we have to mark up real spaces and for this the tilde is used. In expl3 code this does not act as a tie, but simply as a catcode 10 space character (while normal spaces are ignored).

The last argument of `\int_case:nnF` is executed in the "false" case, i.e., when no match has been found. In that case we check the status of `\g__fmwao_gen_warn_bool` and if that is also "false", i.e., we have been asked not to generate warnings, we issue an error message. Why? Because the user asked us explicitly to ignore problems on the current page, but we found nothing wrong. This either means a problem got corrected or the request was intended for a different page. Either way it is probably worth checking.

```
    { \bool_if:NF \g__fmwao_gen_warn_bool
        { \msg_error:nn{widows-and-orphans}{no-problem} } }
```

Finally, we make sure that the next page or column is again checked.

```
  \bool_gset_true:N \g__fmwao_gen_warn_bool
}
```

(*End definition for* `\__fmwao_test_for_widows_etc:`.)

`\__fmwao_problem_identified:n` `\__fmwao_problem_identified:nn` These commands prepare for generating a warning, but only if we are supposed to, i.e., if `\g__fmwao_gen_warn_bool` is true.

```
\cs_new:Npn \__fmwao_problem_identified:n #1 {
   \bool_if:NT \g__fmwao_gen_warn_bool
               { \msg_warning:nn{widows-and-orphans}{#1} }
}
\cs_new:Npn \__fmwao_problem_identified:nn #1 #2 {
   \bool_if:NT \g__fmwao_gen_warn_bool
               { \msg_warning:nnn{widows-and-orphans}{#1}{#2} }
}
```

(*End definition for* `\__fmwao_problem_identified:n` *and* `\__fmwao_problem_identified:nn`.)

2.2 Messages to the user

`\__fmwao_this_page:` `\__fmwao_next_page:` For displaying nice messages to the user we need a few helper commands. The two here show the page number of the current or next page. They are semi-smart, that is they will recognize if the document uses roman numerals and if so display the number as a roman numeral (but in all other cases it uses arabic numerals).

```
\cs_new:Npn \__fmwao_this_page: { \__fmwao_some_page:n   \c@page       }
\cs_new:Npn \__fmwao_next_page: { \__fmwao_some_page:n { \c@page + 1 } }
```

(*End definition for* `\__fmwao_this_page:` *and* `\__fmwao_next_page:`.)

`\__fmwao_some_page:n` `\__fmwao_roman_thepage:` This macro first compares `\thepage` against the code that would be used in the case of a roman numeral representation, and then displays its argument using either arabic numbers or roman numerals.

```
\cs_new:Npn \__fmwao_some_page:n #1 {
  \cs_if_eq:NNTF \thepage \__fmwao_roman_thepage:
    { \int_to_roman:n } { \int_to_arabic:n }
  { #1 }
}
```

`\__fmwao_roman_thepage:` just stores the default definition of `\thepage` if page numbers are represented by roman numerals for use in the comparison above.

```
\cs_new_nopar:Npn \__fmwao_roman_thepage: {\csname @roman\endcsname \c@page}
```

(*End definition for* `\__fmwao_some_page:n` *and* `\__fmwao_roman_thepage:`.)

`\legacy_switch_if:nTF` To evaluate LaTeX 2_ε boolean switches in a nice way, we need a conditional. Eventually this will probably make it into the expl3 code in this or a similar form, but right now it is missing.

```
\prg_new_conditional:Npnn \legacy_switch_if:n #1 {p, T , F , TF }
  { \exp_args:Nc\if_meaning:w { if#1 } \iftrue \prg_return_true:
                                      \else: \prg_return_false: \fi: }
```

(*End definition for* `\legacy_switch_if:nTF`.)

The first message is issued if we have been directed to ignore a problem and there wasn't one:

```
\msg_new:nnnn {widows-and-orphans} {no-problem}
  { No~ problem~ to~ suppress~ on~ this~ page! }
  { Suppression~ of~ a~ widow~ or~ orphan~ problem~ was~ requested~
    but~ on~ the~ current~ page~ there~ doesn't~ seem~ to~ be~ any.~
    Maybe~ the~ text~ was~ changed~ and~ the~ request~ should~ get~
    (re)moved?}
```

The next message is about orphans. They can appear at the bottom of the first or the second column of the current page, if we are in two-column mode. So we check for this and adjust the message accordingly.

```
\msg_new:nnnn {widows-and-orphans} {orphan}
  { Orphan~ on~ page~ \__fmwao_this_page:
    \legacy_switch_if:nT {@twocolumn}
      { \space ( \legacy_switch_if:nTF {@firstcolumn}
                                      { first~ } { second~ } column) }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ the~ orphan.}
```

A hyphen at the end of a page or column requires more or less the same message, so this could have been combined with the previous one.

```
\msg_new:nnnn {widows-and-orphans} {hyphen}
  { Hyphen~ in~ last~ line~ of~ page~ \__fmwao_this_page:
    \legacy_switch_if:nT {@twocolumn}
      { \space ( \legacy_switch_if:nTF {@firstcolumn}
                                      { first~ } { second~ } column) }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ get~
    a~ better~ line~ break. }
```

Widows need a different logic since we detect them when cutting a previous page or column but the widow is on the following one. This message works for "widows", "display widows" as well as math displays by just changing the first word (or words), so here we use an additional argument:

```
\msg_new:nnnn {widows-and-orphans} {widow}
  { #1~ on~ page~
    \legacy_switch_if:nTF {@twocolumn}
      { \legacy_switch_if:nTF {@firstcolumn}
          { \__fmwao_this_page: \space (second~ }
          { \__fmwao_next_page: \space (first~  }
        column)
      }
      { \__fmwao_next_page: }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ the~ widow.}
```

The case of both widow and orphan is similar, but we obviously need different text so we made it its own message.

```
\msg_new:nnnn {widows-and-orphans} {orphan-widow}
  { Orphan~
    \legacy_switch_if:nTF {@twocolumn}
       { \legacy_switch_if:nTF {@firstcolumn}
           { and~ #1 widow~ on~ page~ \__fmwao_this_page: \space
             (first~ and~ second~ }
           { on~ page~ \__fmwao_this_page: \space (second~ column)~
             and~ #1 widow~ on~ page~ \__fmwao_next_page: \space (first~ }
       }
       { on~ page~ \__fmwao_this_page: \space (second~ column)~
         and~ #1 widow~ on~ page~ \__fmwao_next_page: \space (first~ }
     column)
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ both~
    orphan~ and~ widow.}
```

2.3 Adjusting parameter values

To avoid (a lot of) false positives during checking it is important that the parameter values are chosen in a way that all possible combinations lead to unique `\outputpenalty` values. At the same time, we want them to be as close as possible to the values that have been initially requested by the user (or in the document class) and if we deviate too much then this will likely alter the page breaks TEX finds. So here is an outline of how we handle the parameters:

- We set up a property list to hold penalty values that can appear in `\outputpenalty` inside the output routine. The penalties are the "keys" and the corresponding property list value is the source of how they got produced. For example, the key might be `150` and the value `\widowpenalty`.
- Initially the property list is empty. So adding the first item simply means taking the value of one parameter, say `150` from `\widowpenalty` + `\interlinepenalty`, as the key and this formula as the property list value.
- For the next parameter, say `\clubpenalty`, we check if its value (or more precisely its value plus `\interlinepenalty`) is already a key in the property list. If that is the case, then we have failed and must modify the parameter value somehow.
- If not, we also have to check any combination of the current parameter with any parameter processed earlier. If that combination is possible, e.g., `\clubpenalty` (new) and `\widowpenalty` (already processed) then we also have to check the sum. If that sum is already a key in the property list then we have failed as well.
- If we have failed, we iterate by incrementing the current parameter value and try again. Eventually we will get to a value where all combinations we test work, that is, are not yet in the property list.
- We then change the parameter to this value and add all the combinations we tried before to the property list (that is `\clubpenalty` + `\interlinepenalty` both alone and together with `\widowpenalty` in our example). Thus from now on those are also forbidden values.
- We do all this with a helper command that takes the new parameter as the first argument and the list of different cases to try as a comma-separated list as a second argument, e.g.,

```
\__fmwao_decide_penalty:Nn \clubpenalty
         { \clubpenalty + \interlinepenalty ,
           \clubpenalty + \widowpenalty + \interlinepenalty }
```

- This way we are adding all relevant parameters to the property list and at the same time adjusting their values if needed.

- Once all parameters are handled the property list is no longer needed as the parameters got changed along the way, but we keep it around as it allows for a simple display of all settings in one go.

\l__fmwao_penalties_prop Here is the property list for our process.

```
\prop_new:N \l__fmwao_penalties_prop
```

(*End definition for* `\l__fmwao_penalties_prop`.)

__fmwao_initialize: Now we are ready to go. The first action is to clear the property list as the initialization may happen several times.

```
\cs_new:Npn \__fmwao_initialize: {
  \prop_clear:N  \l__fmwao_penalties_prop
```

When TEX breaks a page at a glue item with no explicit penalty involved it sets `\outputpenalty` to `10000` in the output routine to distinguish it from a case where an explicit penalty of `0` was in the document. That means none of our parameters or parameter combinations can be allowed to have that particular value, because otherwise we would get a false match for each break at glue and report an issue. So we enter that value first (by hand) so that it will not be used by a parameter or parameter combination.

```
  \prop_put:Nnn \l__fmwao_penalties_prop {10000} {break~ at~ glue}
```

The next thing is to add the values for `\@lowpenalty`, `\@medpenalty`, `\@highpenalty` to the property list as they also may show up in `\outputpenalty` if a user says, for example, `\nopagebreak[2]`.
Such a penalty from an explicit page break request does not get `\interlinepenalty` added in.

```
  \__fmwao_decide_penalty:Nn \@lowpenalty  { \@lowpenalty}
  \__fmwao_decide_penalty:Nn \@medpenalty  { \@medpenalty}
  \__fmwao_decide_penalty:Nn \@highpenalty { \@highpenalty}
```

Then comes the first real parameter for the orphans:

```
  \__fmwao_decide_penalty:Nn \clubpenalty
    { \clubpenalty + \interlinepenalty }
```

followed by the one for the widows and the one for the display widows:

```
  \__fmwao_decide_penalty:Nn \widowpenalty
    { \widowpenalty + \interlinepenalty ,
      \widowpenalty + \clubpenalty  + \interlinepenalty }

  \__fmwao_decide_penalty:Nn \displaywidowpenalty
    { \displaywidowpenalty + \interlinepenalty ,
      \displaywidowpenalty + \clubpenalty + \interlinepenalty }
```

`\brokenpenalty` can appear on its own, and also with each and every combination we have seen so far:

```
  \__fmwao_decide_penalty:Nn \brokenpenalty
    { \brokenpenalty + \interlinepenalty ,
      \brokenpenalty + \clubpenalty + \interlinepenalty ,
      \brokenpenalty + \widowpenalty + \interlinepenalty ,
      \brokenpenalty + \widowpenalty + \clubpenalty + \interlinepenalty ,
      \brokenpenalty + \displaywidowpenalty + \clubpenalty
                                            + \interlinepenalty }
```

Finally we have the parameter for lonely displays (again without `\interlinepenalty` being added):

```
  \__fmwao_decide_penalty:Nn \predisplaypenalty { \predisplaypenalty }
}
```

If we run the above code with LaTeX's default parameter settings in force it will make a few adjustments and the property list will afterwards contain the following entries:

```
The property list \l__fmwao_penalties_prop contains the pairs (without
outer braces):
>  {10000}  =>  {break at glue}
>  {51}  =>  {\@lowpenalty }
>  {151}  =>  {\@medpenalty }
>  {301}  =>  {\@highpenalty }
>  {150}  =>  {\clubpenalty +\interlinepenalty }
>  {152}  =>  {\widowpenalty +\interlinepenalty }
>  {302}  =>  {\widowpenalty +\clubpenalty +\interlinepenalty }
>  {50}  =>  {\displaywidowpenalty +\interlinepenalty }
>  {200}  =>  {\displaywidowpenalty +\clubpenalty +\interlinepenalty }
>  {100}  =>  {\brokenpenalty +\interlinepenalty }
>  {250}  =>  {\brokenpenalty +\clubpenalty +\interlinepenalty }
>  {252}  =>  {\brokenpenalty +\widowpenalty +\interlinepenalty }
>  {402}  =>  {\brokenpenalty +\widowpenalty +\clubpenalty +\interlinepenalty }
>  {300}  =>  {\brokenpenalty +\displaywidowpenalty +\clubpenalty
                                                    +\interlinepenalty }
>  {10001}  =>  {\predisplaypenalty }.
```

(*End definition for* `\__fmwao_initialize:`.)

`\l__fmwao_tmp_int` `\l__fmwao_tmp_tl` `\l__fmwao_success_bool`

For doing the calculations and insertions into the property list, we will also need an integer register, a token list variable and another boolean variable.

```
\int_new:N  \l__fmwao_tmp_int
\tl_new:N   \l__fmwao_tmp_tl
\bool_new:N \l__fmwao_success_bool
```

(*End definition for* `\l__fmwao_tmp_int`, `\l__fmwao_tmp_tl`, *and* `\l__fmwao_success_bool`.)

`\__fmwao_decide_penalty:Nn`

This is the core command that does the real work of choosing values. Let's recall that its first argument is the parameter we are currently handling and the second argument is a comma-separated list of cases for which we need to ensure that their results are not yet in the property list.

```
\cs_new:Npn \__fmwao_decide_penalty:Nn #1 #2 {
```

We start by setting the boolean to `false` and then run a loop until we have found a suitable parameter value that meets our criteria.

```
  \bool_set_false:N \l__fmwao_success_bool
  \bool_do_until:Nn \l__fmwao_success_bool
```

Inside the loop we start with the assumption that the current value of the parameter is fine and then check if that assumption is true. If yes, we can exit the loop, otherwise we will have to try with a different value.

```
   { \bool_set_true:N \l__fmwao_success_bool
```

For the verification we try each item in the second parameter to see if that is already in the property list. This means evaluating the expression to get the penalty value and then looking it up in the property list. If it is there, we have failed. In this case we set the boolean back to false and break out of the loop over the second argument since there is no point in testing further.

```
     \clist_map_inline:nn { #2 }
       { \int_set:Nn \l__fmwao_tmp_int {##1}
         \prop_get:NVNT
            \l__fmwao_penalties_prop \l__fmwao_tmp_int \l__fmwao_tmp_tl
            { \clist_map_break:n {\bool_set_false:N\l__fmwao_success_bool} }
       }
```

Once we have finished, the boolean will tell us if we are successful so far. If yes, it means there was no conflict. We therefore add all combinations with this parameter to the property list, as from now on they are forbidden as well.

So we map once more over the second argument and enter them:

```
    \bool_if:NTF \l__fmwao_success_bool
      { \clist_map_inline:nn { #2 }
          { \int_set:Nn \l__fmwao_tmp_int {##1}
            \prop_put:NVn \l__fmwao_penalties_prop \l__fmwao_tmp_int {##1}
          } }
```

If we failed we increment the parameter value and retry:

```
      { \int_incr:N #1 }
    }
}
```

One place where we will run this code is at the beginning of the document (so that changes to the parameters in the document class or the preamble are picked up). The other place is when the user changes any of the parameters in the middle of the document via `\WaOsetup`.

```
\AtBeginDocument { \__fmwao_initialize: }
```

(*End definition for* `\__fmwao_decide_penalty:Nn`.)

2.4 The option setup

The options are fairly straightforward:

```
\keys_define:nn {fmwao} {
```

By default messages are given as warnings above. If anything else is wanted the option `check` can be used which simply changes the message class used internally:

```
  ,check .choice:
  ,check / error
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{error}
  ,check / info
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{info}
  ,check / none
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{none}
  ,check / warning
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{ }
```

The other options set parameters to some hopefully "reasonable" values — no real surprises here. LaTeX internally uses `\@clubpenalty` so we need to set this too, if we change `\clubpenalty`.

```
  ,orphans .choice:
  ,orphans / prevent .code:n = \int_set:Nn \clubpenalty  { 10000        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }
  ,orphans / avoid   .code:n = \int_set:Nn \clubpenalty  {  5000        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }
  ,orphans / default .code:n = \int_set:Nn \clubpenalty  {   150        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }

  ,widows .choice:
  ,widows / prevent  .code:n = \int_set:Nn \widowpenalty  { 10000   }
  ,widows / avoid    .code:n = \int_set:Nn \widowpenalty  {  5000   }
  ,widows / default  .code:n = \int_set:Nn \widowpenalty  {   150   }

  ,hyphens .choice:
  ,hyphens / prevent .code:n = \int_set:Nn \brokenpenalty { 10000   }
  ,hyphens / avoid   .code:n = \int_set:Nn \brokenpenalty {  2000   }
```

```
  ,hyphens / default .code:n = \int_set:Nn \brokenpenalty {    50   }
  ,prevent-all  .code:n = \int_set:Nn \clubpenalty        { 10000   }
                          \int_set:Nn \widowpenalty       { 10000   }
                          \int_set:Nn \displaywidowpenalty{ 10000   }
                          \int_set:Nn \brokenpenalty      { 10000   }
                          \int_set:Nn \predisplaypenalty  { 10000   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
```

As an exception, `avoid-all` doesn't set `\predisplaypenalty`; maybe it should.

```
  ,avoid-all    .code:n = \int_set:Nn \clubpenalty         { 5000   }
                          \int_set:Nn \widowpenalty        { 5000   }
                          \int_set:Nn \displaywidowpenalty { 2000   }
                          \int_set:Nn \brokenpenalty       { 2000   }
%                         \int_set:Nn \predisplaypenalty   { 9999   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
```

`default-all` reverts back to the standard LaTeX default values:

```
  ,default-all  .code:n = \int_set:Nn \clubpenalty          { 150   }
                          \int_set:Nn \widowpenalty         { 150   }
                          \int_set:Nn \displaywidowpenalty  {  50   }
                          \int_set:Nn \brokenpenalty        { 100   }
                          \int_set:Nn \predisplaypenalty  { 10000   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
}
```

Once declared we evaluate the options given to the package:

```
\ProcessKeysPackageOptions{fmwao}
```

2.5 Document-level commands

Finally we declare the user-level commands:

`\WaOsetup` This runs the key setup on the first argument and then reinitializes the parameter setup:

```
\NewDocumentCommand\WaOsetup{m}
  { \keys_set:nn{fmwao}{#1}  \__fmwao_initialize: \ignorespaces }
```

(*End definition for* `\WaOsetup`*. This function is documented on page 253.*)

`\WaOparameters` This parameterless command outputs a display of the current parameter settings.

```
\NewDocumentCommand\WaOparameters{}{\prop_show:N \l__fmwao_penalties_prop}
```

(*End definition for* `\WaOparameters`*. This function is documented on page 253.*)

`\WaOignorenext` And here is the command that suppresses any warning on the current page or column:

```
\NewDocumentCommand\WaOignorenext{}
  { \bool_gset_false:N \g__fmwao_gen_warn_bool }
```

(*End definition for* `\WaOignorenext`*. This function is documented on page 253.*)

References

[1] Frank Mittelbach. Managing forlorn paragraph lines (a.k.a. widows and orphans) in LaTeX. *TUGboat* 39:3, 246–251, 2018.

[2] LaTeX3 Project Team. A collection of articles on expl3. https://latex-project.org/publications/indexbytopic/l3-expl3/

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/widows-and-orphans

The dashundergaps package*

Frank Mittelbach

Abstract

The dashundergaps package offers the possibility to replace material in running text with white space in order to build up forms that can be filled in at a later time.

By default the gaps are underlined and followed by a gap number in parentheses, but many other designs are possible, e.g., dashes or dots instead of the underline, no gap numbers or a different format for them, gap widening for easier fill-in, etc.

There is also a teacher's mode which shows the normally hidden text in a special (customizable) format.

This is another article in a series of *TUGboat* articles describing small packages to introduce coding practices using the expl3 programming language. See [1] for the first article in the series. For more details on expl3 refer to [2].

Contents

1 Introduction

The dashundergaps package provides a single command \gap which takes one argument and produces a gap of the width of that argument. To better mark this gap it is underlined in some form (could be a solid line, a dashed or dotted line or even a wriggling line). Furthermore, gaps can be numbered to be able to easily refer to them. Figure 1 shows an example in the form of a fill-in puzzle.

As you see there, some gaps are numbered with a superscript number (not the default setting) while others aren't. How this is done and how to change the result is explained in the next section.

There also exists a "teacher mode" in which the gaps are filled with the text given in the argument. This can be used to show the correct answers of a test (as we do in Section 4) or to give a sample fill-in for a form, to help people fill it out correctly. The

* This is a reimplementation (using expl3, the LaTeX3 programming language) of a package originally written by Luca Merciadri in 2010.

The initial 'E.' in Donald E. Knuth's name stands for _____(1). The well-known answer to the Ultimate Question _____ is 42 according to _____ _____(2). The first edition of _____(3) celebrates its silver anniversary in 2019. Historically speaking, `expl3` stands for _____ ____(4) even though it is a production language these days.

And here are some hints for the puzzle if you want to fill it out:

1. If only everything would be that easy to answer.
2. The author of the book "Last Chance To See" and of a famous radio show.
3. Back then known as the doggie book.
4. Old names die hard.

The answers are given in Section 4, showing the gaps filled in using the so-called teacher mode, which can be activated or deactivated at any point in the document.

Figure 1: A fill-in puzzle using dashundergaps

"teacher mode" produces the same line breaks because it ensures that the fill-ins take the same amount of space as the gaps.

Another important feature is the possibility to artificially widen the gaps, compared to the textual material in the argument. After all, when a form is filled by hand people typically need more space to write some text compared to the same text being typeset. So making the gaps simply as wide as the material likely results in too little space.

2 The user interface

The dashundergaps package is built as a small application on top of the ulem package, a package that defines several commands for underlining ⟨*simple-text*⟩ in various ways.

`\uline` `\uuline` `\uwave` `\dashuline` `\dotuline`

`\uline{`⟨*simple-text*⟩`}` `\uwave{`⟨*simple-text*⟩`}` ...

This means that by loading dashundergaps the ulem commands such as `\uline`, `\uwave` and so forth are automatically made available. These commands are used to do most of the work and the current package only makes sure that, instead of the words, empty boxes of the same width are used by ulem. This way we get underlined gaps of the right size.

By default, ulem changes `\emph` to underline text, so for this application, it is loaded with the option `normalem` to prevent that from happening.

`\gap`

`\gap*[`⟨*style*⟩`]{`⟨*text*⟩`}`

The main command provided by the package is `\gap` which expects a mandatory ⟨*text*⟩ argument containing the material that is used to produce the gap (and is normally invisible). By default the gap is underlined, though that can be changed.

Possible ⟨*style*⟩s:
`u` = `\uline`
`d` = `\uuline`
`w` = `\uwave`
`b` = ⟨*blank*⟩
`-` = `\dashuline`
`.` = `\dotuline`

The optional ⟨*style*⟩ argument explicitly defines a certain type of underlining: `u` stands for normal underlining (via `\uline`), `d` for double underlining (via `\uuline`), `w` for a wavy line (via `\uwave`), `b` for blank (i.e., no underlining whatsoever), "`-`" for a dash-line (via `\dashuline`) and finally "`.`" for underlining with dots (via `\dotuline`).

In the default configuration gaps are numbered using the counter `gapnumber` and this number is shown in parentheses after the gap. With the star form the generation of the number is toggled, i.e., if it would be produced because of the current option settings

it will be suppressed; if it is suppressed through an option it will be typeset. This way one can select the most convenient setting via an option for the whole document and use `*` to toggle it as needed.

Since `\gap` uses ulem's commands it inherits the limitations of these commands; notably, only simple text can be used in the ⟨*text*⟩ argument. For example, a `\footnote` couldn't be used in the argument (but then that wouldn't make much sense in a gap, would it?).

`\TeacherModeOn` `\TeacherModeOn % show gap material`
`\TeacherModeOff` `\TeacherModeOff % do not show gap material`

Also supported is a teacher mode in which the material for the gaps is visible. This can be used to show the expected answers in case `\gap` is used for preparing tests, or to show a sample fill-in of a form. The teacher mode can be turned on or off anywhere in the document using `\TeacherModeOn` or `\TeacherModeOff`, respectively. Alternatively, it can also be set via an option, as we will see below.

`\dashundergapssetup` `\dashundergapssetup{`⟨*comma-separated key-value list*⟩`}`

The package can be loaded with a number of options (discussed in Section 2.1). A likely better approach is to set any options with the declaration `\dashundergapssetup` which is normally used in the preamble, but can be used throughout the document to change settings on the fly. It only changes explicitly given options so it can be used to overwrite some defaults but leave everything else unchanged.

2.1 Options to customize the gap display

All of the package options are implemented as key/value options. For boolean options one can give just the option name as a short form for setting the option to `true`. Most options can be specified during package loading in the optional argument of `\usepackage`. However if the value requires some LaTeX code (e.g., `gap-font`, which expects a font declaration command) then this will not work due to some limitations in the current LaTeX package loader. For such options use `\dashundergapssetup` instead, which will always work.

2.1.1 Gap modes

The general processing mode is defined through the following options:

`teacher-mode` Boolean that turns on teacher mode (i.e., the gap material will be visible if set to `true`). Its default is `false`.

`gap-mode` Boolean that is the inverse of `teacher-mode` and just provided for convenience, i.e., an abbreviation for `teacher-mode=false`.

`teachermode` Alternative name for `teacher-mode` because that is what it was called in the first package release.

2.1.2 Gap formatting

Formatting of the gaps is handled by the following six options:

`gap-format` A choice option defining how the gap is marked. It accepts the following values: `underline` (default), `double-underline`, `dash`, `dot`, `wave`, `blank`.

`gap-format-adjust` A boolean (default `true`). If set, the "line" below the gap is raised to be roughly at the baseline, which normally looks better when there is no text above the line.

`teacher-gap-format` Another choice option, with the same values as `gap-format`, used when we are in "teacher mode", but this time the default is `blank` as normally the gap text is typeset in the bold font and is therefore already identifiable, with

no need for additional underlining. However, depending on the circumstances it might be helpful to keep the underlining (or use a different kind of underlining) while in "teacher mode".

gap-font This option expects a font directive as its value, e.g., `\bfseries` (which is also the default). Using this option without supplying a value is equivalent to supplying an empty value. It will be used to determine the font for the gap material regardless of the mode. This is important to ensure that the gaps always have the same width regardless of whether or not the material is shown.

For the example puzzle above it was set to `\itshape`, which you can see in the puzzle answer.

dash Short name for `gap-format=dash`.

dot Short name for `gap-format=dot`.

2.1.3 Gap numbers

Producing the gap numbers is handled by the following options:

gap-numbers Boolean that determines whether or not gap numbers are displayed. Default is `true`.

gap-number-format Code that is executed when a gap number is produced. Default is `\textnormal{␣(\thegapnumber)}`.

numbers Short name for `gap-numbers`.

There is also a way to control displaying the total number of gaps:

display-total-gaps Boolean to determine if the total number of gaps should be shown at the very end of the document. Default is `false`.

displaynbgaps This is just another name for the same boolean; it was used in the first version of the package.

2.1.4 Gap widening

Finally, for extending the gap width we have these options:

gap-widen Boolean that decides if the gaps should be made wider or not (default is `false` but mainly for historical reasons).

gap-extend-minimum Minimum of extra space that should be added to each gap if gap widening is active. Default is `20pt`, i.e., `10pt` on either side.

gap-extend-percent Percentage (as a number) by which the gap should be made wider if widening is active. The result is compared to `gap-extend-minimum` and the larger of the two is used. Default is `20`.

widen Short name for `gap-widen`.

3 Differences from the original package

The main user interface of the two versions is identical, so it is possible to use the new version as a drop-in replacement for the old. However, the feature set in form of key/value options has been greatly extended, offering functionality previously unavailable. Furthermore, a number of bugs have been corrected (and possibly new ones introduced).

- Stray spaces in the definition of `\gap` (that showed up in the output) have been eliminated.
- Various combinations of options that didn't work are now possible.
- Explicit hyphenations `\-` showed up in gap mode, now they can be used.
- Nesting isn't possible for obvious reasons, but the fact is now detected and catered to by ignoring the inner gap requests after generating an error.

- Option names have been normalized (though the original names are still available).
- The option `phantomtext` is no longer necessary, though still supported (with a warning) as a no-op.
- The names of the LaTeX counters used have changed, so if you directly addressed them that would need changing.
- The font used in teacher mode (by default boldface) is now also used if gap mode is chosen, to ensure that the output in all modes produces identical line breaks; for the same reason, the ulem machinery is always used, even if not underlining (or dashing, etc.).
- The gaps can be extended by a percentage or by a minimum amount to ensure that there is enough space to fill in the text (given that hand-written text is typically wider than typeset material); the values are adjustable.
- `\gap` now has an optional argument through which you can explicitly request the type of underlining you want to use.
- `\gap` also supports a star form which toggles the setting of gap numbers.
- The use of `\label` within the `\gap` command argument allows for later reference to that gap by its number (provided a gap number is typeset).
- The implementation is done with `expl3`, the programming language for LaTeX3. Although invisible to the user, in some sense that was the main purpose of the exercise: to see how easy it is to convert a package and use the extended features of `expl3`.

4 Solution to the puzzle

Here we repeat the puzzle from above with `\TeacherModeOn`.

> The initial 'E.' in Donald E. Knuth's name stands for *Ervin* (5). The well-known answer to the Ultimate Question *of Life, the Universe, and Everything* is 42 according to *Douglas Adams* (6). The first edition of *The LaTeX Companion* (7) celebrates its silver anniversary in 2019. Historically speaking, `expl3` stands for ***EX**perimental **P**rogramming **L**anguage **3*** (8) even though it is a production language these days.

This was produced using the following changes to the defaults:

```
\dashundergapssetup{
  ,gap-number-format  = \,\textsuperscript{\normalfont
                                          (\thegapnumber)}
  ,gap-font           = \itshape
  ,teacher-gap-format = underline
  ,gap-widen
}
```

As you can see we use `\itshape` for the font (to be able to show the bold face in one of the answers) and also force underlining in teacher mode to better show the gap widening. The gap number is raised and we separate it a tiny bit from the gap material. We also use `\normalfont` in the formatting to ensure that the gap number is set upright and not in italic shape.

5 The implementation

5.1 Loading and fixing/changing ulem

The first thing to do is to load ulem without changing \emph or \em:

```
⟨*package⟩
\RequirePackage[normalem]{ulem}
```

The code in this section follows LaTeX 2ε conventions, i.e., models the commands as they look in the ulem package.

\dotuline The dots produced by \dotuline depend on the current font, which is a somewhat questionable design — if you underline a text part with a single bold word somewhere inside it will change the shape of the dot line. So we always use the \normalfont dot (this is not done in the original definition).

```
\def\dotuline{\bgroup
  \UL@setULdepth
  \markoverwith{\begingroup
     \advance\ULdepth0.08ex
     \lower\ULdepth\hbox{\normalfont \kern.1em .\kern.04em}%
     \endgroup}%
  \ULon}
\MakeRobust\dotuline
```

(*End definition for* \dotuline*. This function is documented on page 264.*)

\uwave The original \uwave used a hard-wired value of 3.5pt for the lowering. We change that to be based on the current value of \ULdepth so that the user (or this package here) can change the placement.

```
\def\uwave{\bgroup
  \UL@setULdepth
  \advance\ULdepth 0.6\p@
  \markoverwith{\lower\ULdepth\hbox{\sixly \char58}}\ULon}
\MakeRobust\uwave
```

(*End definition for* \uwave*. This function is documented on page 264.*)

\fmdug@ublank \fmdug@ublank underlines with blanks. Normally not especially useful (which is why we make it internal), but if we want to have ulem acting, but without actually visibly underlining, this is the command to use.

```
\def\fmdug@ublank{\bgroup\let\UL@leadtype\@empty\ULon}
```

(*End definition for* \fmdug@ublank*.*)

\UL@dischyp We need to do a little patching to ensure that nothing is output by the ulem commands
\UL@putbox if we don't want it to. So the next two commands are from ulem with \box replaced by \fmdug@box so that we can change the behavior.

```
\def\UL@dischyp{\global\setbox\UL@hyphenbox\hbox
   {\ifnum \hyphenchar\font<\z@ \string-\else \char\hyphenchar\font \fi}%
   \kern\wd\UL@hyphenbox \LA@penalty\@M
   \UL@stop \kern-\wd\UL@hyphenbox
   \discretionary{\fmdug@box\UL@hyphenbox}{}{}\UL@start}

\def\UL@putbox{\ifx\UL@start\@empty \else % not inner
  \vrule\@width\z@ \LA@penalty\@M
  {\UL@skip\wd\UL@box \UL@leaders \kern-\UL@skip}%
  \fmdug@box\UL@box  \fi}
```

(*End definition for* \UL@dischyp *and* \UL@putbox*.*)

\fmdug@box By default we output the box in the commands above, but when we don't want to output anything visible we change the definition to generate a box with empty content but the right size.

```
\let\fmdug@box\box
```

(*End definition for* `\fmdug@box`.)

5.2 The main implementation part

The rest of the package is written in `expl3`. We use `fmdug` as our internal prefix.

```
⟨@@=fmdug⟩
```

We need the package `xparse` for specifying the document-level interface commands and `l3keys2e` to use the `expl3` key value methods within LaTeX 2_ε. These packages automatically require `expl3` so there is no need to load that explicitly.

```
\RequirePackage{xparse,l3keys2e}
```

Here we introduce the package and specify its version number:

```
\ProvidesExplPackage{dashundergaps}
                    {2018/11/09}
                    {v2.0d}
                    {Dashing and underlining phantom text}
```

5.2.1 User interface commands

\gap The `\gap` command parses for a star, optional and mandatory argument and then calls `\__fmdug_gap:nnn` to do the work.

```
\DeclareDocumentCommand \gap { som } { \__fmdug_gap:nnn {#1}{#2}{#3} }
```

(*End definition for* `\gap`. *This function is documented on page 264.*)

\dashundergapssetup Change options anywhere.

```
\NewDocumentCommand \dashundergapssetup { m }
  { \keys_set:nn {fmdug} {#1} \ignorespaces }
```

(*End definition for* `\dashundergapssetup`. *This function is documented on page 265.*)

\TeacherModeOn We provide shortcuts for turning teacher mode on or off.

\TeacherModeOff

```
\DeclareDocumentCommand \TeacherModeOn  {}
                        { \bool_set_true:N \l__fmdug_teacher_bool }
\DeclareDocumentCommand \TeacherModeOff {}
                        { \bool_set_false:N \l__fmdug_teacher_bool }
```

(*End definition for* `\TeacherModeOn` *and* `\TeacherModeOff`. *These functions are documented on page 265.*)

5.2.2 Counters

\c@gapnumber We have one user-level counter which is referenceable and holds the gap number of the current gap. It can be reset to 0 to restart counting.

```
\newcounter{gapnumber}
```

(*End definition for* `\c@gapnumber`.)

\c@totalgapnumber We also keep track of all gaps ever made using another user-level counter. Since this one is supposed to keep track of the total number of gaps, it makes little sense to modify it at the document level. However, there may be use cases even for that and more importantly, by making it a user-level counter it is possible to refer to the total

number of gaps easily, e.g., via \thetotalgapnumber.

```
\newcounter{totalgapnumber}
```

(*End definition for* \c@totalgapnumber.)

\l__fmdug_extend_dim A help register to calculate the gap width later on.

```
\dim_new:N \l__fmdug_extend_dim
```

(*End definition for* \l__fmdug_extend_dim.)

\l__fmdug_extra_left_gap_tl
\l__fmdug_extra_right_gap_tl Two scratch token lists to enlarge the gap on the left or right side.

```
\tl_new:N \l__fmdug_extra_left_gap_tl
\tl_new:N \l__fmdug_extra_right_gap_tl
```

(*End definition for* \l__fmdug_extra_left_gap_tl *and* \l__fmdug_extra_right_gap_tl.)

\l__fmdug_gap_format_tl
\l__fmdug_teacher_gap_format_tl The gap formatting is normally handled by a ulem command; which one depends on the options used. To record the choice we store it in a token list (one for normal and one for teacher mode).

```
\tl_new:N \l__fmdug_gap_format_tl
\tl_new:N \l__fmdug_teacher_gap_format_tl
```

(*End definition for* \l__fmdug_gap_format_tl *and* \l__fmdug_teacher_gap_format_tl.)

5.2.3 Messages

```
\msg_new:nnn {dashundergaps} {deprecated}
  { The~ #1~ `#2'~ you~ used~ \msg_line_context: \ is~ deprecated~ and~
    there~ is~ no~ replacement.~ Since~ I~ will~ not~ guarantee~ that~
    #1~ `#2'~ will~ be~ kept~ forever~ I~ strongly~ encourage~ you~
    to~ remove~ it~ from~ your~ document. }
\msg_new:nnnn {dashundergaps} {nested}
  { The~ \gap command~ can't~ be~ nested! }
  { Nesting~ doesn't~ make~ much~ sense~ as~ the~ inner~ one~
    wouldn't~ be~ visible.~ ~ To~ allow~ further~ processing~ it~ is~
    handled~ as~ if~ it~ hasn't~ been~ asked~ for. }
\msg_new:nnnn {dashundergaps} {gap-format-value}
  { Unknown~ value~ for~ key~ '#1 gap-format'! }
  { Supported~ values~ are~ 'underline',~ 'double-underline',\\
    'dash',~ 'dot',~ 'wave'~ or~ 'blank'. }
```

5.2.4 Option handling

Here we define all the possible option keys for use either as package options or inside \dashundergapssetup. These are all straightforward assignments to variables. These internal variables are declared by the key declarations if unknown, so they are not separately declared beforehand.

```
\keys_define:nn {fmdug}
    {
      % ====================================
     ,teacher-mode      .bool_set:N = \l__fmdug_teacher_bool
     ,teacher-mode      .default:n  = true
     ,teacher-mode      .initial:n  = false
      % ------------------
     ,gap-mode  .bool_set_inverse:N = \l__fmdug_teacher_bool
      % ====================================
     ,gap-format
        .choice:
```

In the case of dashes and even more so in the case of dots, it looks fairly ugly if they are below the baseline as if there were text above. We therefore raise them up a bit if the option `gap-format-adjust` is given (which is the default).

In the case of dots we undo exactly the amount by which they are lowered in ulem so that they end up precisely at the baseline, in case they are followed by a real dot. In other cases we stay a bit below the baseline.

The same is done below when the optional argument is evaluated. But we don't do this in teacher mode since there we *will* have text above and we don't want to bump into that.

```
      ,gap-format / underline
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl
                              { \__fmdug_gap_format_adjust:n{.4pt} \uline }
      ,gap-format / double-underline
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl
                              { \__fmdug_gap_format_adjust:n{2pt} \uuline }
      ,gap-format / dash
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl
                           { \__fmdug_gap_format_adjust:n{0pt} \dashuline }
      ,gap-format / dot
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl
                         { \__fmdug_gap_format_adjust:n{-.08ex} \dotuline }
      ,gap-format / wave
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl
                              { \__fmdug_gap_format_adjust:n{1pt} \uwave }
      ,gap-format / blank
         .code:n = \tl_set:Nn \l__fmdug_gap_format_tl { \fmdug@ublank  }
      ,gap-format / unknown
         .code:n = \msg_error:nnn{dashundergaps}{gap-format-value}{}
      ,gap-format
         .initial:n = underline
       % ====================================
```

This controls the raising of the gap underline by some amount. We implement it as a `.choice` even though it looks like a boolean.

```
       ,gap-format-adjust
         .choice:
       ,gap-format-adjust / true
         .code:n = \cs_set:Npn \__fmdug_gap_format_adjust:n ##1
                               { \setlength\ULdepth {##1} }
       ,gap-format-adjust / false
         .code:n = \cs_set_eq:NN \__fmdug_gap_format_adjust:n \use_none:n
      ,gap-format-adjust
         .default:n = true
      ,gap-format-adjust
         .initial:n = true
      ,adjust .meta:n = { gap-format-adjust }
       % ====================================
      ,teacher-gap-format
         .choice:
     ,teacher-gap-format / underline
         .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \uline }
      ,teacher-gap-format / double-underline
         .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \uuline }
      ,teacher-gap-format / dash
         .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \dashuline }
      ,teacher-gap-format / dot
         .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \dotuline }
```

```
    ,teacher-gap-format / wave
       .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \uwave }
    ,teacher-gap-format / blank
       .code:n = \tl_set:Nn \l__fmdug_teacher_gap_format_tl { \fmdug@ublank }
    ,teacher-gap-format / unknown
       .code:n = \msg_error:nnn{dashundergaps}{gap-format-value}{teacher-}
    ,teacher-gap-format
       .initial:n = blank
     % ====================================
    ,gap-widen         .bool_set:N = \l__fmdug_gap_widen_bool
    ,gap-widen         .default:n  = true
    ,gap-widen         .initial:n  = false
     % ------------------
    ,widen                  .meta:n = { gap-widen }
     % ------------------
    ,gap-extend-minimum .dim_set:N = \l__fmdug_gap_min_dim
    ,gap-extend-minimum .initial:n = 20pt
     % ------------------
    ,gap-extend-percent .tl_set:N  = \l__fmdug_gap_percent_tl
    ,gap-extend-percent .initial:n = 20
     % ====================================
    ,gap-numbers        .bool_set:N = \l__fmdug_number_bool
    ,gap-numbers        .default:n  = true
    ,gap-numbers        .initial:n  = true
     % ------------------
    ,numbers                .meta:n = { gap-numbers }
     % ------------------
    ,gap-number-format .tl_set:N    = \l__fmdug_gapnum_format_tl
    ,gap-number-format .initial:n   = \textnormal{\space (\thegapnumber)}
     % ====================================
    ,display-total-gaps .bool_gset:N = \g__fmdug_display_total_gaps_bool
    ,display-total-gaps   .default:n = true
    ,display-total-gaps   .initial:n = false
     % ====================================
    ,gap-font    .tl_set:N = \l__fmdug_font_tl
    ,gap-font   .default:n =
    ,gap-font   .initial:n = \bfseries
```

And finally the original options, now as aliases:

```
     % ====================================
    ,teachermode   .meta:n =  { teacher-mode }
    ,dash          .meta:n =  { gap-format = dash }
    ,dot           .meta:n =  { gap-format = dot }
    ,displaynbgaps .meta:n =  { display-total-gaps }
     % ------------------
    ,phantomtext
       .code:n = \msg_warning:nnnn{dashundergaps}{deprecated}
                                  {option}{phantomtext}
     % ====================================
  }
```

`\__fmdug_gap:nnn` At last, here comes the action. `\__fmdug_gap:nn` expects two arguments: `#1` indicates what kind of "underlining" is wanted (anything not recognized is ignored, in particular "–NoValue–" if `\gap` was used without an optional argument) and `#2` is the material to produce a gap for.

```
\cs_new:Npn\__fmdug_gap:nnn #1#2#3 {
  \group_begin:
```

Define the font used inside the gap. We need to do this up front since we want to measure the text (and that needs the correct font already).

```
\l__fmdug_font_tl
```

Nesting is not supported so inside the gap we redefine `\__fmdug_gap:nnn` to raise an error and just return the third argument if it is encountered again.

```
\cs_set:Npn \__fmdug_gap:nnn ##1##2##3
  {
    \msg_error:nn{dashundergaps}{nested}
    ##3
  }
```

We always increment the counter for the total number of gaps, but increment the `gapnumber` only if we are displaying it. For the latter one we use `\refstepcounter` to make it referenceable.

```
\stepcounter{totalgapnumber}
\bool_xor:nnT { #1 } { \l__fmdug_number_bool }
  { \refstepcounter{gapnumber} }
```

Next we prepare for widening if that is being asked for: Measure the width of the text and then set `\l__fmdug_extend_dim` to be the requested percentage divided by two of that width (since we add it later on both sides).

```
\bool_if:NTF \l__fmdug_gap_widen_bool
  {
    \settowidth \l__fmdug_extend_dim {#3}
    \dim_set:Nn \l__fmdug_extend_dim
              { \l__fmdug_gap_percent_tl \l__fmdug_extend_dim / 200 }
```

Then compare it to the minimum / 2 and choose whatever is larger.

```
     \dim_compare:nNnT \l__fmdug_extend_dim < { .5\l__fmdug_gap_min_dim }
        { \dim_set:Nn \l__fmdug_extend_dim { .5\l__fmdug_gap_min_dim } }
```

Now we prepare what needs to go to the left and the right of the gap.

```
     \tl_set:Nn \l__fmdug_extra_left_gap_tl
        { \hbox_to_wd:nn\l__fmdug_extend_dim{} \allowbreak }
     \tl_set:Nn \l__fmdug_extra_right_gap_tl
        { \allowbreak \hbox_to_wd:nn\l__fmdug_extend_dim{} }
  }
```

And if no widening is asked for we clear these two token lists so they don't do anything.

```
  {
    \tl_clear:N \l__fmdug_extra_left_gap_tl
    \tl_clear:N \l__fmdug_extra_right_gap_tl
  }
```

Next comes deciding the gap format. If in teacher mode it will be whatever is in `\l__fmdug_teacher_gap_tl`. Otherwise, either it is based on the content of the optional argument or, if that is not given or unknown, it will be `\l__fmdug_gap_format_tl`.

```
\bool_if:NTF \l__fmdug_teacher_bool
  { \l__fmdug_teacher_gap_format_tl }
  {
```

But before we execute any of the ulem commands we make sure that they do not output text.

```
    \cs_set:Npn   \fmdug@box ##1 {\hbox_to_wd:nn{\box_wd:N ##1}{}}
    \str_case:nnF {#2}
        {
          {u} { \__fmdug_gap_format_adjust:n{.4pt}   \uline  }
```

```
              {d} { \__fmdug_gap_format_adjust:n{2pt}    \uuline }
              {w} { \__fmdug_gap_format_adjust:n{1pt}    \uwave  }
              {b} { \fmdug@ublank }
              {.} { \__fmdug_gap_format_adjust:n{-.08ex} \dotuline }
              {-} { \__fmdug_gap_format_adjust:n{0pt}   \dashuline }
            }
            { \l__fmdug_gap_format_tl }
      }
```

Whatever was decided as the gap format, it needs one argument, i.e., the material (with possible gap extension on both sides).

```
    {\l__fmdug_extra_left_gap_tl #3 \l__fmdug_extra_right_gap_tl }
```

Finally we typeset the gap number if that was requested.

```
    \bool_xor:nnT { #1 } { \l__fmdug_number_bool }
                  { \l__fmdug_gapnum_format_tl }
```

Close the group from above to keep any of the redefinitions confined.

```
  \group_end:
}
```

(*End definition for* `\__fmdug_gap:nnn`.)

`\__fmdug_display_total_gaps:` This command will display the total number of gaps if requested. The hard-wired formatting comes from the first version of the package.

```
\cs_new:Npn \__fmdug_display_total_gaps: {
  \vfill \centering
  \bfseries Total~ Gaps:~ \thetotalgapnumber
}
```

(*End definition for* `\__fmdug_display_total_gaps:`.)

5.2.5 Closing shop

At the end of the document we typeset the total number of gaps if requested.

```
\AtEndDocument{
  \bool_if:NT \g__fmdug_display_total_gaps_bool
              \__fmdug_display_total_gaps:
}
```

So what remains to be done is executing all options passed to the package via `\usepackage`.

```
\ProcessKeysPackageOptions{fmdug}
⟨*package⟩
```

References

[1] Frank Mittelbach. The widows-and-orphans package. *TUG*boat 39:3, 252–262, 2018. https://ctan.org/pkg/widows-and-orphans

[2] LaTeX3 Project Team. A collection of articles on expl3. https://latex-project.org/publications/indexbytopic/l3-expl3/

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/dashundergaps

State secrets in bibliography-style hacking

Karl Berry and Oren Patashnik

BibTeX output, in essence, consists of chunks of information separated by punctuation. The information chunks are things like the formatted author's name, the title of the work being cited, or its publication date. And the punctuation separator between chunks, in BibTeX's standard styles, is either a comma or a period.

But exactly how the bibliography styles (`.bst` files) determine and then output that separating punctuation, for even moderately experienced .bst-file hackers, is somewhat of a mystery, or even a complete secret. This short article demystifies the process, and exposes a secret.

When the bibliography-style output routines are ready to send a chunk of information to the output (`.bbl`) file, they generally don't know what punctuation follows that chunk; it might be a comma or it might be a period, depending on which chunk of information comes next. So the output routines (for BibTeX's standard styles, and for many others) keep the previous information chunk on its working stack, and when the current chunk is ready for output — at which point the correct separator between the previous and current chunks is known — they output that previous chunk, along with the now-known separator, to the `.bbl` file, leaving the current chunk on the stack until the next chunk is ready for output. And the process repeats.

And how do the output routines know what separator to use?

The answer is in a comment in `btxbst.doc`, BibTeX's documentation (and template) file for the standard styles (all code blocks here are reformatted for *TUGboat*):

```
% To tell which separator is needed,
% we maintain an output.state.
```

(Aside: Much of the reason for the mystery/secret is historical. Originally that documentation line from `btxbst.doc` was stripped when the standard `.bst` files were generated. Over the years, other styles — for example, those generated from `makebst` — similarly did not contain that documentation line. Thus it remained a sort of secret, hidden away in `btxbst.doc`.)

So it's the `output.state` variable that determines between-chunk punctuation.

And how does a style hacker who wants to introduce a new punctuation mark implement that?

Answer: Create a new output state.

A recent IEEE-like bibliography style shows how that's done. That style wanted to separate any url (which normally appears at the end of the entry) from previous information with a semicolon.

An example formatted entry:

[1] H. Kopka and P.W. Daly, *Guide to LaTeX*, Addison-Wesley Professional, 4th edition, 2003; `amazon.com/dp/0321173856`.

It's the semicolon before the url that's new. Here's a description of the fairly straightforward pieces of new code that `ieeelike.bst` uses to implement the new style. (Recall that the `.bst` language is (mostly) stack-based and uses postfix notation, something like PostScript.)

First there's code to create a new output state, say with the symbolic name `after.url.separator`, so-named because the url chunk that's being created comes after the semicolon that separates it from the previous chunk.

```
INTEGERS {
      output.state before.all mid.sentence ...
      after.url.separator }   % new output state
...
#4 'after.url.separator :=  % unique state value
```

Next, code to change the current output state to our new one when the url chunk is created:

```
FUNCTION {url.block}
{       % change the output state appropriately:
  output.state before.all =
    'skip$     % if entry starts with url (rare)
    { after.url.separator 'output.state := }
  if$
}
...
FUNCTION {format.url}
{ ...
  url.block
  ⟨code to format the url chunk⟩
}
```

Finally, the output-routine code that, for the desired output state, writes out the previous chunk appended with the semicolon separator that will precede the url:

```
    { output.state after.url.separator =
        { "; " * write$ }
```

See `ieeelike.bst` (linked from the *TUGboat* contents page for this issue) for the full context.

And that, in a nutshell, is how to add the new semicolon separator: by creating a new output state. The `output.state` secret has been leaked.

◇ Karl Berry and Oren Patashnik
https://tug.org/bibtex

Experiments with \parfillskip

Udo Wermuth

Abstract

Plain TEX sets the glue parameter \parfillskip in such a way that any length between 1 sp and the full line width is accepted for the width of the material printed in the last line of a paragraph. In certain circumstances, typesetting tradition objects to a last line with text that has a width less than the indentation or to completely filled last lines.

This article analyzes different specifications for the \parfillskip glue based on experiments with twelve one-, two-, or three-line paragraphs supported by theoretical considerations. The analysis shows how TEX's line-breaking procedure acts on the last line of a paragraph and how it can run into problematic situations if \parfillskip has an injudicious specification. This might lead to ugly output.

1 Introduction

TEX has a handful of glue parameters that affect the typesetting of paragraphs. Two of these parameters, \spaceskip and \xspaceskip, replace the two \fontdimen values for the interword glue and the extra space [11, p. 76]. The spaces in the paragraph carry the glue characteristics that are specified by these two parameters if they are nonzero. Two other glue parameters change all the lines in a paragraph: \leftskip and \rightskip add glue to the left or right of every line [11, p. 100]. They are used, for example, to typeset a paragraph narrower, i.e., justified but indented on both sides, or in a shape that shows a straight margin only on one side. All four parameters have the value 0 pt in plain TEX.

The fifth glue parameter is special as it is usually applied in a single place in a paragraph: It is the \parfillskip glue, which has a direct effect only on the last line where it acts as an "additional \rightskip" [11, p. 274]. However, it can affect more than one line in a paragraph. If a paragraph is interrupted by display math mode the line before the display is treated like a last line of a paragraph, although the paragraph has not ended yet. TEX uses the value of \parfillskip that is current when it starts to break either a part or the whole paragraph into lines. So more than one specification of \parfillskip might be applied in a single paragraph. For example, the paragraph starts in a group in which \parfillskip is locally changed and after a displayed equation the group ends but more text follows. The last line before the display then uses a different \parfillskip than the end of the text.

TEX does several things when it has to build a last line: First either an infinite penalty is added or, if the paragraph has a glue item at its end, TEX changes this glue into an infinite penalty item. This penalty prevents a line break in front of the horizontal skip \hskip\parfillskip that is added by TEX to finish the paragraph [12, §816].

Plain TEX sets the value of \parfillskip to 0pt plus 1fil [11, p. 100]. This specification gives stretchability to the last line so that it can contain text whose width is shorter than the line width. As it is a glue parameter that an author is allowed to manipulate any glue specification can be assigned to \parfillskip, for example, a natural width different from 0 pt, a nonzero shrinkability, or a stretchability of finite order. A change in the value has an impact on the line-breaking decisions made by TEX.

In the rest of the article the phrase "length of the last line" means "length of the material in the last line". Normally, the width of the last line is always \hsize (the command \parshape is not discussed in this article). So a "short last line" means that the width of the text in the last line can be called "short". And this word means in this article that the value of \parindent is larger or not much smaller than the width of that text.

Additional \rightskip. The glue \parfillskip is usually only applied to the last line of a paragraph. Therefore it can be used to get special effects for this line. For example, the assignments

```
\leftskip = 0pt plus 1fil
\rightskip = 0pt plus -1fil
\parfillskip = 0pt plus 2fil
```

sets the last line centered without affecting other lines. In all but the last line the \rightskip neutralizes the \leftskip so that there is a net contribution of 0 pt. But the last line has a \leftskip of 0pt plus 1fil and at the right side the sum of \rightskip and \parfillskip which equals 0pt plus 1fil, i.e., on both sides is the same amount of infinite stretchability and the line is centered in the output; see [4] (or [22]).

Of course, there are other ways to manipulate the last line. For example, the end of a paragraph can execute additional typesetting commands if the control sequence \par is redefined. Peter Wilson describes such methods in his columns *Glisterings* [22, 23, 24]. This article analyzes what happens if solely the glue specification of \parfillskip is changed and this analysis already fills quite a few pages.

Contents. First, the default setting of plain TEX for `\parfillskip` is discussed in section 2. Then in the next section the value of `\parfillskip` is set to 0 pt either for a complete document or a single paragraph. It presents also some effects that might occur if the input for a paragraph contains negative infinite stretchability.

In section 4, experiments with finite dimensions for the stretchability of `\parfillskip` are executed. First with a stretchability larger than `\hsize`, second with one that is a fraction of `\hsize`, and third with a negative finite stretchability. Section 5 adds some theoretical results. The next section checks what happens if `\parfillskip` has natural width besides stretchability, and section 7 presents the related theory. It also shows how to make use of the trace data written by `\tracingparagraphs` and how different values for the stretchability can be compared in a certain sense.

Specifications for `\parfillskip` that have natural width and shrinkability but no stretchability are the topic of section 8. Theoretical results about such settings are in section 9. In section 10 all three dimensions of the glue `\parfillskip` are changed to finite nonzero values.

Section 11 looks at a couple of specifications for `\parfillskip` based on the facts learned in the previous sections and compares them to some suggestions made by others. The last section provides a summary of the results.

2 Plain TEX's default `0pt plus 1fil`

The default setting is useful as it works with any last line from normal text. The last line has either badness 0, i.e., it is decent, or it is tight with a badness as high as 100. Therefore, no loose or very loose lines are possible, i.e., glue never stretches. The default `\parfillskip` cannot be the reason for an overfull line and an underfull line can only appear if the last line is empty, e.g., if the paragraph faultily ends with a forced break entered by the author.

On the other hand, very short lines are possible; a hyphenated part of a word suffices to form a tolerated last line. (According to [6, 3.11], the last word of a paragraph should never be hyphenated.) Typesetting tradition recommends having last lines that are longer than the indentation of paragraphs if the start of a paragraph is identified by indentation (see [9, p. 142]), and that an indentation shall be at least 0.5 em. Values of 1 em and 1`\baselineskip` — the natural width of the `\baselineskip` — are recommended in [3, p. 40], and then [3, p. 42] demands at least four letters in the last line. Plain TEX indents by 20 pt, i.e., 2 em in `cmr10`. This is a very high value compared to the above recommendations.

A comment: A specification with the font related unit 1 em should be made after the font for the text was selected. TEX uses for the unit em the quad width of the font that is active when the specification of `\parfillskip` is processed. A switch to a smaller font or a different face in the text does not change `\parfillskip`.

Experiment 1: Description
Show that the last line of a paragraph can be shorter than the indentation if the plain TEX default values for `\parindent` and `\parfillskip` is used.

TEX input

```
1. Please answer if my topic is ``in'' or
``out''. \TeX: in
```

TEX output

• `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
1. Please answer if my topic is "in" or "out". TEX: in □

In the section "TEX output" of an experiment a paragraph that starts with a bullet shows the specification of `\parfillskip` that is used in the following paragraphs either up to the end of the experiment or up to another line that starts with a bullet. The specification is written as a formula, not as a valid TEX assignment. The symbol '□' that is printed in the right margin marks the end of an experiment.

As mentioned above the spaces in the text of the last line are either perfect or they shrink.

Experiment 2: Description
Show that the last line of a paragraph can end at the right margin.

TEX input

```
2. Has the last line of this paragraph
badness~0 and has no interword space to
stretch? Do they shrink now?
```

TEX output

• `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now? □

Of course, the first two experiments show simple things. They are presented mainly for comparison with later experiments that reuse the two texts. Notice that each text starts with a number that identifies the experiment where it was introduced with plain TEX's `\parfillskip`. The first experiment is minimal, as TEX has only one valid way to typeset the text. So no setting of `\parfillskip` can produce a different second line, obeying the `\hsize`. The last line of experiment 2 has badness 2 so its glue shrinks. It will be useful to have a variant of this experiment that has badness 100 in its last line and a tie for the last word.

Experiment 2 continued: TEX input

```
$2'$. Has the last line of this paragraph
badness~0 and has no interword space to
stretch?\kern.557pt!
Do they shrink~now?\kern.557pt!
```

TEX output

• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
2′. Has the last line of this paragraph badness 0 and has no interword space to stretch?! Do they shrink now?! ▯

Here are the lines written by \tracingparagraphs to show that the last line is maximally tight. (The information is written to the transcript file of the run if \tracingparagraphs is set to 1; see [11], pp. 98–99, or [19] for a description of this data.)

Experiment 2 continued: Log file contents

```
1. @firstpass
2. []$\ninerm 2[]$. Has the last line
   of this paragraph badness 0
3. @ via @@0 b=18 p=0 d=784
4. @@1: line 1.1 t=784 -> @@0
5. and
6. @ via @@0 b=64 p=0 d=5476
7. @@2: line 1.3 t=5476 -> @@0
8. has no interword space to stretch?!
   Do they shrink now?!
9. @\par via @@2 b=100 p=-10000 d=12100
10. @@3: line 2.3- t=17576 -> @@2 ▯
```

If the \parindent is 0 pt and if the \parskip does not separate paragraphs with a noticeable vertical glue item a clear indication of the end of the paragraph is missing. In [9, p. 143], white space of at least 1 em is recommended at the end of the last line of a paragraph with justified margins. Ragged-right text requires much larger white space at the end of the last line, which is of course dependent on the amount that the right margin changes.

German typesetters learned the old rule never to leave white space at the end of the last line that is less than 1 em wide if paragraphs are indented. In such a situation it was considered better to keep the right margin straight. Nowadays the rule is not recommended anymore [9, p. 142] so it is ignored in this article. (According to [5], or see [22], a similar rule with \parindent instead of 1 em existed in Russia.)

Thus we have two observations of typographic trouble that the default setting of \parfillskip can create in some situations.

O1 (Short line): The last line might be shorter than the \parindent and such an event leaves so much white space that it can look like an empty line between paragraphs.

O2 (Completely filled): The text fills the last line completely without a visual indication that it is the last line of the paragraph, especially, if the \parindent is 0 pt and the \parskip does not separate paragraphs by an empty line. White space of width 1 em should be at the end of the last line in such a setup.

The first observation is especially bad if vertical space is used to structure the text [9, p. 142].

The two observations can be formulated more formally as recommendations that should be obeyed:

$$\texttt{\parindent} < \text{length of last line} \quad (1)$$

$$\text{length of last line} \le \texttt{\hsize} - 1\,\text{em}. \quad (2)$$

These recommendations are extracted from different books by different experts. This is usually not a good approach as each expert has a unique collection of typographical rules and two such collections might contain conflicting rules. For example, page 21 of [3] ends with a first line of a paragraph; something that [6, 3.11] forbids. On page 43 of [3] R. Bringhurst explains why he accepts such lines.

The important point for the recommendations (1) and (2) is that they are not suggested to be held simultaneously. Therefore the two recommendations should be treated independently.

In this article, sometimes the text of an experiment is indented, often it starts flush left. Nevertheless every text is used for both recommendations as only the last lines count.

Displays. Display math mode in a paragraph, i.e., material between a pair of doubled dollar signs, can create a problem if the \parfillskip contains only finite dimensions. TEX determines the length of the line directly above such a display and depending on its width either \abovedisplayshortskip or the (usually wider) \abovedisplayskip is put between text and formula except if it is an *alignment display* [11, p. 190]. In this case TEX always applies the latter [12, §1206]. The short skip is used if, more or less, the end of the text and the start of the formula do not overlap and if no equation number is used on the left side [11, p. 189]. After the math material is typeset TEX adds either the glue \belowdisplayskip or \belowdisplayshortskip if there was a decision based on the length of the line before the display; otherwise \belowdisplayskip is used.

Here is how TEX determines which skip to use. As soon as two dollar signs occur TEX computes the dimension \predisplaysize that contains, as one summand, the width of the line before the display. The other two summands are the width by which the line is shifted, for example, if \leftskip is nonzero, and a font-related constant of 2 em. But there are two exceptions: If there is no previous line, i.e., the display starts the paragraph, \predisplaysize is

set to −\maxdimen. If the previous line contains interword glue and that glue stretches or shrinks \predisplaysize is set to \maxdimen [11, p. 188; 12, §1148]. This is necessary to guarantee that TEX does not make machine-dependent decisions.

Discussion. Ok, in some use cases the default setting of plain TEX fails to meet certain recommendations. And this "failure" is not reported by TEX as a warning or an error message. An author who sees in the output an unwanted effect can consider rewriting the paragraph, but also, sometimes a sequence of commands at the end of the paragraph can be used to avoid such a situation.

Recommendation (1) can be supported by two simple rules during the input. First, use a tie in front of the last word if it has fewer than four letters. Second, put longer last words whose last fragment after hyphenation has fewer than four letters in an \hbox. If TEX is not able to typeset this paragraph it reports an overfull line error message and the author can fix the situation. The first rule is easy to follow if the author is used to applying ties as recommended by [11, pp. 91–93]. The second rule is harder to observe and it requires more typing.

People have suggested non-default settings for \parfillskip in order to obey the two mentioned recommendations better and without new rules for the input. For example, P. Taylor [18, p. 388] suggests using 0.7\hsize instead of 1 fil as the stretchability of \parfillskip to make (1) more likely. W. Schmidt [17] discusses the use of a nonzero natural width in a very special case: if \parindent is 0 pt, always end a paragraph with some white space, i.e., to support (2). He uses the glue `2em plus 1fil`. And F. Mittelbach [16, p. 344] used natural width together with shrinkability to address both recommendations. Let $x :=$ \hsize $- 1.5$\parindent and $y := x - 1$ em and then set \parfillskip to x `plus 0pt minus` y. This combination leaves white space at the end of the paragraph and as the natural width does not cover the whole line width, very short lines are assumed to be unlikely. Of course, it is possible to change all three dimensions of the glue specification. P. Wilson suggests in [24, p. 340] the specification `0.75\hsize plus 0.06\hsize minus 0.75\hsize`; see also [1]. And in [10, p. 1156], Donald E. Knuth and Michael F. Plass discuss the effect of the parameter \looseness and write "The penalty for adjacent lines of contrasting classes seems to work best in connection with looseness if the finishing glue at the paragraph end is set to have a normal space equal to about half the total line width, stretching to nearly the full width and shrinking to zero." (In [14, p. 194], the text was changed to "... a normal space equal to about one-third of the total line width, stretching to the full width and shrinking to zero." I can only guess why the text was changed: more experience with both parameters. \parfillskip became a changeable parameter only a few months before the article was written, and at the same time \looseness was added; see entries 457 and 459 in [13, Ch. 11].) As mentioned before, this article only discusses changes to \parfillskip, not the results of varying two parameters at the same time.

The length of the last line seems to be computable via the abovementioned \predisplaysize if \parfillskip has its default setting. In example 2 of [16, p. 344], a macro is presented to measure the length of the last line based on the idea of adding display math at the end of a paragraph and outputting the \predisplaysize. As noticed by the author, some aspects that the display introduces, such as skips, can be reverted, but \tracingoutput will still show them as their occurrence is not deleted from TEX's memory.

Summary. Short last lines cannot be prevented except by rewriting the text or forcing a manual line break, and at least one ugly looking line is accepted; see experiment 1. Using ties and boxes an author can help TEX avoid typesetting a short last line and output a warning if it cannot be avoided.

Completely (or nearly so) filled last lines cannot be completely prevented either except by the author rewriting the text or adding white space at the end of each paragraph using a tie and an empty \hbox, or forcing a manual line break, and accepting at least one non-perfect line in the paragraph. Or the author makes the simplest of all changes and gives the natural width of \parfillskip a nonnegative value. This last method is discussed in section 6 together with the problems that such a change produces.

There is no general need to use finite dimensions for the stretchability or to switch to shrinkability for \parfillskip as has been suggested. Nevertheless, it seems to be an interesting topic to study, whether such settings have applications and what happens if finite dimensions are used. On the other hand it does not help to have a higher order of infinite stretchability, i.e., 1 fill [11, p. 72]. It might be useful if \leftskip, \rightskip, or the text contains infinite stretchability of the first or second order but otherwise it has no effect different from 1 fil.

3 Zero \parfillskip

The assignment of 0 pt to \parfillskip forces TEX to finish the last line of the paragraph flush right if

this line contains at least one interword space. This contradicts (2) but in some situations such a value makes sense. For example, when `\parshape` is used the last line should contain a zero `\parfillskip`.

This setting is also useful to split a very long paragraph into smaller parts to avoid memory overflow as stated in the answer to exercise 14.15 of *The TEXbook* [11]: Execute `\par` to end a paragraph and use the setting `\parfillskip=0pt`. To keep the distance of lines at `\baselineskip` the glue that TEX inserts between paragraphs must be set to 0 pt too. All parameter changes should be done in a group to avoid a global change. Therefore the complete answer is given as:

```
{\parfillskip=0pt\par\parskip=0pt\noindent}
```

But that works only if it is either entered at a place where TEX has found a valid line break before, or if enough text is available to allow TEX to break at almost any interword glue. Otherwise the spaces in the line that are typeset before the break are often extremely stretched.

Experiment 3: Description
Show that the interword glue of the last line can get extremely stretched.

TEX input

```
3. A short text in 1 line. Or has the paragraph
2 or 3 lines?
```

TEX output

- `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
  3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
- `\parfillskip` ← 0 pt:
  3. A short text in 1 line. Or has the paragraph 2 or 3 lines? ∎

The experiment's text is typeset in the first pass if plain TEX's default `\parfillskip` is used. But TEX reports that the last line of the last paragraph is underfull with badness 10000. It was forced to execute a second pass to check, without success, if the paragraph can be typeset with lines whose badness do not exceed the current tolerance. Plain TEX sets the tolerance for the first pass to 100 and for the second to 200 [11, p. 96]. So, more observations:

O3 (Glue stretches): Interword glue of finite order stretches in the last line if all other glue has finite stretchability and if the sum of `\parfillskip`'s natural width and the width of the material is shorter than `\hsize`.

O4 (2nd pass forced): If the glue in every possible last line of the first pass has to stretch so much that the badness of this line must be larger than `\pretolerance` then TEX is forced to execute a second pass. And depending on the value of `\emergencystretch` and the result of the second pass a third pass might be executed.

O5 (Underfull line): The output paragraph has an underfull line if the glue in the typeset last line has to stretch so much that the badness of that line becomes larger than `\tolerance`. With the plain TEX value of 1000 for `\hbadness` this is not reported for all cases.

The observation O4 about the execution of a second pass can occur with the default setting of `\parfillskip` too. But then it is the *text* that cannot obey the limit `\pretolerance`. Using the plain TEX default setting the text of experiment 3 is typeset in the first pass. It is the changed `\parfillskip` that is responsible for the text of this experiment being typeset a second time in the second pass. So in this article it is said that the default value of `\parfillskip` never *forces* a second pass but it can happen, for example, with a `\parfillskip` of 0 pt.

In contrast to the other observations, O4 states a technical point and not a property of the new last line. Of course, TEX might output the paragraph with different line breaks and the last line that is typeset in the second pass might not have a badness larger than `\pretolerance`; actually all lines of the paragraph might have badness values that are less than or equal to that limit but then at least one hyphen was inserted in the text.

A comparison of both outputs in experiment 3 shows that with plain TEX's default settings three words are placed in the last line but only two when `\parfillskip` is 0 pt. This means that the second output contains a first line with a larger badness because with the plain TEX default settings TEX must minimize the badness of the first line in this experiment. TEX considers the break as forced. It assigns the so-called *artificial demerits* [12, §854], which `\tracingparagraphs` shows as `d=*` [12, §856]. Artificial demerits occur only in the final pass, i.e., usually the second or the third pass. When TEX falls back to artificial demerits for a line this line does not contribute to the total demerits of the paragraph: TEX does not calculate the line demerits, they are set to 0 [12, §855]; compare the `t`-values in lines 10 and 13 in the following trace.

Experiment 3 continued: Log file contents

```
1. @secondpass
2. []\ninerm 3. A short text in 1 line. Or
   has the para-graph
3. @ via @@0 b=25 p=0 d=1225
4. @@1: line 1.1 t=1225 -> @@0
5. 2
6. @ via @@0 b=2 p=0 d=144
7. @@2: line 1.2 t=144 -> @@0
```

```
8. or
9. @ via @@0 b=12 p=0 d=484
10. @@3: line 1.2 t=484 -> @@0
11. 3 lines?
12. @\par via @@3 b=10000 p=-10000 d=*
13. @@4: line 2.0- t=484 -> @@3
```

O6 (No demerits): If TEX is not able to find a valid break for the last line of a paragraph it might classify the break as forced. Then TEX avoids calculating the line demerits and assigns to this line *artificial demerits* which count as 0.

Note however that the paragraph is output differently if `\looseness` is set to −1 because of the sequence in which feasible breakpoints are listed by TEX. An explanation is given in [19], p. 372.

Experiment 3 continued: TEX definitions

```
\looseness=-1
```

TEX output

• `\parfillskip` ← 0 pt:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

An underfull line without demerits also occurs when the text of experiment 1 is typeset with a zero `\parfillskip`; the output is identical to the one shown in experiment 1. But with a very short word on the last line or more shrinkability in the penultimate line a short last line might disappear.

Experiment 4: Description
Show that a short last line might be absorbed by the penultimate line.

TEX input

```
\noindent 4. My keyboArd is broken. When I
press the key for the lowercAse A the screen
repeAts it severAl times: a a a a
```

TEX output

• `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a

• `\parfillskip` ← 0 pt:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a

O7 (Remove short line): To avoid infinite or high badness values TEX might dissolve the original last line and move its material into the formerly penultimate line.

The last line is also absorbed in a second pass even if `\finalhyphendemerits` get applied.

Experiment 4 continued: TEX input

```
\noindent $4'$. My keyboArd is broken; when I
press an 'A' in lowercAse (only) the screen
repeAts it four times: a a a a
```

TEX output

• `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
$4'$. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

• `\parfillskip` ← 0 pt:
$4'$. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

A last line may also be extended, that is, it receives some material from the penultimate line.

Experiment 5: Description
Show that a last line might be extended.

TEX input

```
\noindent 5. With enough interword glue as well
as short words at the end of the 1st line the
2nd can be extended.
```

TEX output

• `\parfillskip` ← 0 pt plus 1fil (plain TEX's default):
5. With enough interword glue as well as short words at the end of the 1st line the 2nd can be extended.

• `\parfillskip` ← 0 pt:
5. With enough interword glue as well as short words at the end of the 1st line the 2nd can be extended.

The shifting of "at" from the penultimate line into the last line allows TEX to typeset the second paragraph in the second pass. Its last line has the fitness class *very loose* and the first line stays in the class *decent*, so `\adjdemerits` are applied to the last line.

O8 (Extend last line): To avoid infinite or high badness values TEX might pack more material into the original last line to reduce its badness.

Changing `\parfillskip` once. The general solution to assign 0 pt to `\parfillskip` for a single paragraph is to enter

```
\hskip-\parfillskip\ \par
```

at its end. This technique can be used in situations where

$$\texttt{\hsize} - 1\,\text{em} < \text{length of last line} < \texttt{\hsize}$$

to obey the abovementioned (outdated) tradition when (2) is violated.

Note the control space in front of the `\par` to avoid the glue from our `\hskip-\parfillskip`, that would otherwise appear at the end of the paragraph, from being removed by TEX as described above. Note also that `\par` is used here to signal the end of the paragraph but an empty line does the job too; the paragraph must only end after the control space. And finally, note that the bare minus sign is allowed but not `-1`, as any factor in front of the glue coerces it into a dimension, that is, only the natural width remains (see exercise 24.3 in [11]).

Negative infinite stretchability. TeX contains a primitive command `\hfilneg` that provides a horizontal skip with a stretchability of −1 fil. It cancels the stretchability of a skip with 1 fil when it is used in the same line. Therefore the above general solution to zeroize the default `\parfillskip` for a single paragraph can be formulated shorter as

```
\hfilneg\ \par
```

(compare to exercise 14.13 of [11]).

A digression. The use of negative infinite stretchability *in* a text — not just to cancel `\parfillskip`'s stretchability — affects the list of break candidates for the last line in probably unexpected ways.

Experiment X: Description

Show that negative infinite stretchability in the text can make TeX typeset fewer or more lines if `\parfillskip` is 0 pt.

TeX input

```
Xa. There's no need to set interword space by
hand: a \penalty10000\hskip3.33333pt plus -1fil
b. And be careful with negative stretchability!
```

TeX output

- `\parfillskip` ← 0 pt plus 1fil (plain TeX's default):

    Xa. There's no need to set interword space by hand:
a   b. And be careful with negative
stretchability!

- `\parfillskip` ← 0 pt:

    Xa. There's no need to set interword space by hand:
a   b. And be careful with negative stretchability! ▯

TeX reports for the three-line paragraph 9804 demerits; 100 demerits more than for the two-line solution. A line with positive or negative infinite stretchability receives badness 0 [12, §852] if the glue does not need to shrink.

It is possible to construct an experiment that produces a longer paragraph when the stretchability of `\parfillskip` is set to 0 pt. But as in experiment Xa it contains some tricks.

Experiment X continued: TeX definitions

```
\def\X#1(#2)[#3+#4-#5]{% repeat a letter
 \penalty10000
 \xleaders\hbox to #1pt{\hss #2\hss}%
   \hskip#3\hsize plus #4fil minus #5\hsize
 \penalty10000 }
```

TeX input

```
\noindent Xb. \hbox{Myy}\X4.9(y)[0.16+1-0.15]\
     \hbox{old electrical}\penalty5\
\hbox{typewriter see}\X3.9(e)[0.14+0-0.1]%
     \hbox spread 1.5pt{ms to}
\hbox{hav}\X5(v)[0.13+1-0.13]\hbox{e a bad}
\hbox{bug. It repeats letters in mh}%
     \X4.7(h)[0.185+1-0]\null\penalty5\
\vphantom{h}\X4.7(h)[0.18+{-1}-0.5]!
\hbox{an unpredictable way \dots\ !}
```

TeX output

- `\parfillskip` ← 0 pt plus 1fil (plain TeX's default):

Xb. Myyyyyyy old electrical typewriter seeeeeeeeems to
havvvve a bad bug. It repeats letters in mhhhhhhhhh!
an unpredictable way . . . !

- `\parfillskip` ← 0 pt:

Xb. Myyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy old electrical
typewriter seeeeeeeeeems to havvvvvvvvvvvvvve a bad
bug. It repeats letters in mhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhh! an unpredictable way . . . ! ▯

Although not the topic of this section a positive natural width can be used with experiment Xb to show a typographical disaster. The text sticks out into the left margin!

Experiment X continued: TeX output

- `\parfillskip` ← 120 pt:

Xb. Myyyyyyyyyyyyyyyyyyyyyyyyyyyyyy old electrical
typewriter seeeeeeeeeems to havvvvvvvvvvvvvve a bad
bug. It repeats letters in mhhhhhhhhhhhhhhhhhhhhhhhhh
! an unpredictable way . . . ! ▯

And here is another anomaly: The paragraph might have the same last line as with the default setting but the number of lines is larger.

Experiment X continued: TeX definitions

```
\looseness=2
```

TeX output

- `\parfillskip` ← 0 pt plus 1fil (plain TeX's default):

Xb. Myyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy old electrical
typewriter seeeeeeeeeems to havvvvvvvvvvvvvve a bad
bug. It repeats letters in mhhhhhhhhhhhhhhhhhhhhhhhhh
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh!
an unpredictable way . . . ! ▯

This new paragraph has 550 total demerits instead of 525. No setting of `\parfillskip` alone will force TeX to typeset these five lines.

In the rest of this article only "normal" texts are considered. No text will contain spaces with infinite stretchability, negative dimensions, or a shrinkability larger than the natural width. Of course, in the specification of `\parfillskip` such settings will still be considered.

Discussion. A couple of observations were made in this section: Some are bad, some neutral, some are typographical, other just technical. An underfull line with extremely stretched glue is typographically bad. A forced second pass is primarily technical, although now hyphenation and wider spaces are possible. Normally, a second pass should only be triggered either by content or by a command entered by the author, not unpredictably because of the setting of a parameter. But in this article the influence of `\parfillskip` is studied more or less independently of other aspects. Therefore settings that force a second pass are considered too.

Table 1: Observations on zero glue

Specification of `\parfillskip`			
	natural width	0pt	0pt
	stretch	1fil	0pt
	shrink	0pt	0pt
Observation			
1	Short line	1	(1)
2	Completely filled	2	+ if the line contains glue
3	Glue stretches	−	3
4	2nd pass forced	−	3
5	Underfull line	−	3
6	No demerits	−	3
7	Remove short line	−	4
8	Extend last line	−	5

Legend: −/ /+: never/don't care/always
n/(n): (implicitly) shown in example number n

Table 1 summarizes the observations and lists an experiment in which the observation occurs. The reason for the observation must be the setting of `\parfillskip`. The header lines state the specification of `\parfillskip` that was used in the experiments. If the specification in an experiment differs from those given in the headlines but the observation is the same, the number of the experiment is placed in parentheses. For instance, this is done for experiment 1 with the observation "Short line". The output is identical to the result shown in experiment 1 but the second line has badness 10000 and artificial demerits. For experiment 3 these facts have made a completely different and ugly output, for experiment 1 there is no visible indication of a problem for TEX. Nevertheless because of such cases the *always* in the last column of Table 1 needs a comment.

A zero `\parfillskip` can be used to achieve an aesthetic effect for a whole document, as in [7]: All paragraphs end flush right and they are separated by an empty line. This layout is part of a general page design approach [8]. Such a strict requirement must have created a lot of work and the willingness to rephrase the text so that bad things do not happen.

One of the bad things with `\parfillskip=0pt` is the occurrence of artificial demerits. TEX does not show them as an error message, only the underfull line is reported as a warning. The specification of `\parfillskip` should not be the reason for their appearance. Again, artificial demerits can also occur with plain TEX's default settings. But in such a case it is not the specification of `\parfillskip` that causes the problem; it is the content itself. But then the output might not have a visual problem.

The all-zero specification might leave too much white space at the beginning of math displays if a line in which the glue stretches precedes a short centered formula without an equation label at its left.

If the problem occurs it can be fixed by setting `\abovedisplayskip` and `\belowdisplayskip` to their `\...shortskip` variants directly after the two dollar signs. This change is done inside a group so the old values are restored after the closing `$$`. Or all paragraphs that contain displayed material start preemptively within a group, in which the default `\parfillskip` is active and which ends after the last display. But the simplest input rule is to enter always, for example, `\hfil\ $$` (with a control space) instead of the opening `$$`.

It should be noted that `\parfillskip` is relevant for line breaking and therefore the number of lines is affected by its specification. Of course, it is not directly a problem that paragraphs get shorter or longer if `\parfillskip` is changed. But in the first case recommendation (2) might be violated and in the second case it is recommendation (1) that might not be obeyed.

Summary. The value 0 pt for `\parfillskip` supports (1) if possible but might fail spectacularly if the last lines contains interword glue. Recommendation (2) is only obeyed with underfull last lines that contain no glue. In general, this setting might be used in certain circumstances to show a special effect but it needs too much care and work to be applied for a longer document.

4 Using finite stretchability

Of course, the stretchability of 1 fil in the default setting of `\parfillskip` is not needed to fill the last line; simply `\hsize` has enough stretchability. But this setting does not have the same properties as infinite stretchability. One remarkable difference was already mentioned above: the spacing around display math.

Finite stretchability: $\nu \times$ `\hsize`. As mentioned earlier: With the default setting of `\parfillskip`, TEX creates, under normal conditions (e.g., a last line that is not empty), either a last line with badness 0 or a tight last line with badness up to 100. When the input

```
\noindent\hbox to 1sp{\hss}        (*)
```

is used with finite stretchability the maximum difference for the badness of a non-tight line is reported for *TUGboat*'s column width, namely, 225 pt. With the definition `\parfillskip=0pt plus` ν`\hsize` the following values are found:

$\nu =$	1	2	3	4	5	6
badness for input (*) =	100	12	4	2	1	0

Thus, there is no difference between a stretchability of 1 fil and a stretchability of 6`\hsize` in the

line breaks if \leftskip, \rightskip, and the input for the paragraph do not contain infinite glue. Note, a zero badness does not imply that the interword glue has its natural width. The setting 2\hsize still creates a last line of fitness class *decent*, that is, no \adjdemerits can be charged to the last line if it is not already charged using infinite stretchability.

One argument for using a finite stretchability is to make (1) more likely; that is, for a short last line help O8 "Extend last line" to occur. The idea is: With finite stretchability a higher badness is assigned to short lines as a lot of stretchability must be used and TEX has a reason to select line breaks putting more material into the last line in order to reduce its badness value. So short lines should be less likely.

The texts of experiments 1 and 2 produce the identical output as before with TEX's default setting for all ν so they are not shown here. The biggest change of badness values is seen for the step from 2\hsize to 1\hsize. With these values it should be easiest to find an experiment that produces a longer last line.

Experiment 6: Description
Show a noticeable difference between the finite stretchabilities of 2\hsize and 1\hsize.

TEX input

```
\noindent 6. One line or two for this text?
That is the question, or?
```

TEX output

• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
6. One line or two for this text? That is the question, or?
• \parfillskip ← 0 pt plus 2\hsize:
6. One line or two for this text? That is the question, or?
• \parfillskip ← 0 pt plus 1\hsize:
6. One line or two for this text? That is the question, or?

The single line, a tight line, has badness 65 and 5625 demerits. A second line with the stretchability of \hsize would receive badness 84, demerits 8836. Finite stretchability might avoid a short last line and produce a paragraph that has one line less. That line ends flush right violating (2). This effect was seen before in O7 "Remove short line".

For this observation the badness of the last line might even be smaller. As seen above, the badness of the last line might become 1 if the stretchability of \parfillskip is set to 5\hsize. This difference is important enough to produce other line breaks than the default setting.

Experiment 7: Description
Show that the badness 1 for the last line can be a reason to make a paragraph shorter.

TEX output

• \parfillskip ← 0 pt plus 6\hsize:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
• \parfillskip ← 0 pt plus 5\hsize:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a

The badness and the demerits for the three lines are: 10/400, 1/121, and 0/100; the *path demerits* [20], i.e., the sum of the line demerits, is therefore $400 + 121 + 100 = 621$. In the paragraph with two lines the values are 10/400 and 5/225 with path demerits of 625. In this case a third line would get 1/121 instead of 0/100. Then the path demerits are 642, so this path is less attractive for TEX.

But of course, the original idea for longer lines can be shown too. In the next experiment the two-lines paragraph's first line is decent and with finite stretchability it stays decent but the treatment of glue changes.

Experiment 8: Description
Show that \parfillskip=0pt plus \hsize might create a longer last line.

TEX input

```
\noindent 8. One line or two for this text?
That's the question, or not?
```

TEX output

• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
8. One line or two for this text? That's the question, or not?
• \parfillskip ← 0 pt plus \hsize:
8. One line or two for this text? That's the question, or not?

The badness values of the two first lines are 1 and 8, resp. This increase is compensated for in the case with a stretchability of \hsize by the decrease of the badness in the second line from 78 for the single word to 64 with two words. Of course, the interword glue in the last line of the second paragraph has to stretch. And again the badness value 1 is sufficient to make a difference.

Experiment 9: Description
Show that the badness 1 for the last line can be a reason to extend it.

TEX input

```
\noindent 9.\ Give me 5! As a factor for
width; to stretch! I need a five (5)!
```

TEX output

• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
9. Give me 5! As a factor for width; to stretch! I need a five (5)!
• \parfillskip ← 0 pt plus 5\hsize:
9. Give me 5! As a factor for width; to stretch! I need a five (5)!

Finite stretchability: (ν/10)×\hsize. The finite stretchability might be less than \hsize to support (1) even more: Now the white space provided by the stretchability of \parfillskip cannot cover the whole line width. That means that a short last line represents an underfull line and TEX will create such a line only if there is no other way to break the text.

In this case a successful first pass cannot be guaranteed. Let's check what badness β is produced with input (∗) if the stretchability is reduced to ν/10 of \hsize:

$\nu =$	3	4	5	6	7	8	9
$\beta =$	3701	1558	800	463	291	195	137

The values for $\nu = 1$ and $\nu = 2$ result in an infinite badness of 10000. Stretchability lower than \hsize might force TEX to execute a second pass; see O4. Values greater than or equal to 0.8\hsize obey plain TEX's \tolerance. Otherwise underfull lines as described in O5 might be produced. With the plain TEX value of 1000 for \hbadness [11, p. 29] this is only reported for a stretchability somewhat less than 0.5\hsize; for exact numbers see section 5.

But underfull lines are not the only problem with small ν; artificial demerits might occur.

Experiment 10: Description
Show that a smaller fraction for the stretchability can make the last line of a paragraph shorter again.

TEX output
• \parfillskip ← 0 pt plus 0.7\hsize:
8. One line or two for this text? That's the question, or not?
• \parfillskip ← 0 pt plus 0.5\hsize:
8. One line or two for this text? That's the question, or not? □

A stretchability of 0.7\hsize can make the last line of a paragraph longer as it was shown with a stretchability of \hsize, but the stretchability of 0.5\hsize fails in this experiment as artificial demerits are reported by TEX. The last line must become nearly 40% filled with such a small stretchability to get a badness value less than 200 (and then the change is unnecessary to avoid a short last line).

Experiment 11: Description
Show that a smaller fraction for the stretchability can make the last line of a paragraph longer.

TEX input
```
11. Sure, this text needs always two lines with
the current line width.
```

TEX output
• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
11. Sure, this text needs always two lines with the current line width.
• \parfillskip ← 0 pt plus 0.7\hsize:
11. Sure, this text needs always two lines with the current line width.
• \parfillskip ← 0 pt plus 0.5\hsize:
11. Sure, this text needs always two lines with the current line width. □

TEX needs a second pass only in the third case. To extend the last line a high price in demerits must be paid; the total demerits are, in sequence: 200, 8200, and 51410.

Negative finite stretchability. There is an important difference if the negative stretchability is infinite or finite [15]. The first case produces last lines of badness 0 as discussed in section 3. But if the stretchability for the calculation of the badness is finite and negative the badness of the last line is set to 10000 [12, §852, §108] and artificial demerits are applied. As the line demerits for the last line are not calculated they do not influence TEX's line breaking decisions.

Experiment 12: Description
Show the difference between positive and negative finite stretchability.

TEX output
• \parfillskip ← 0 pt plus 0.7\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
• \parfillskip ← 0 pt plus 0.5\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
• \parfillskip ← 0 pt plus −0.7\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines? □

The finite stretchability of 0.7\hsize makes the last line longer, but the negative stretchability shortens it. The first paragraph has 19994 demerits, the third only 484 as the last line does not count because of artificial demerits. As seen before, TEX picks a line break that does not even minimize the line demerits of the first line and it produces an underfull last line. The paragraph with stretchability 0.5\hsize has the same problem. When the interword spaces of the last lines for these two paragraphs are closely inspected then the space in the last one seems to be quite small.

Let's execute a test similar to a situation of an answer to a multiple choice question.

Experiment 13: Description
Show that a space vanishes with negative stretchability.

TEX input
```
\noindent 13. a
```

TEX output
• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
13. a

• `\parfillskip` ← 0 pt plus −0.7`\hsize`:
13a

The space between "13." and "a" disappears, overwriting the period. The line is underfull and the log file shows that the stretchability of the space erodes the natural width.

Experiment 13 continued: Log file contents

```
1. \hbox(5.79999+0.0)x225.0, glue set -1.33734
2. .\ninerm 1
3. .\ninerm 3
4. .\ninerm .
5. .\glue 4.11108 plus 4.62497 minus 0.34259
6. .\ninerm a
7. .\penalty 10000
8. .\glue(\parfillskip) 0.0 plus -157.49931
9. .\glue(\rightskip) 0.0
```

Note the value after `glue set` is negative. It is not a minus separated by a space from the number which is used to signal shrinkability [11, p. 79]. Therefore the "a" is moved $1.33734 \times 4.62497\,\text{pt} > 4.11108\,\text{pt}$ to the left.

O9 (Backspaces): Negative finite stretchability in `\parfillskip` induces negative stretchability for finite interword glue and might create spaces with negative width in the last line.

Discussion. Table 2 shows which experiments document the observations. The stretchability 6`\hsize` cannot be distinguished from 1 fil if no infinite glue appears in the paragraph. But the interword glue in a line with spaces in front of a display stretches so that the vertical spacing is wrong.

The reason for parentheses in the last column of Table 2 is the non-specific stretch value; therefore experiment 13, which uses a concrete value, appears in parentheses. The output is so bad that other observations are not analyzed for these specifications so a lot of entries are empty.

Table 2: Observations for finite stretchability

Specification of `\parfillskip` ($h \equiv$ `\hsize`)									
	natural width	0pt	0pt	0pt	0pt	0pt	0pt	0pt	0pt
	stretch	1fil	$6h$	$5h$	h	$.8h$	$.7h$	$.5h$	<0pt
	shrink	0pt	0pt	0pt	0pt	0pt	0pt	0pt	0pt
Observation									
1	Short line	1	7	(1)	(1)	(1)	(1)	(1)	
2	Completely filled	2	(2)	7	6	(2)	(2)	(2)	
3	Glue stretches	–	–	(13)	8	(8)	10	11	
4	2nd pass forced	–	–	–	–	(1)	(1)	11	+
5	Underfull line	–	–	–	–	–	(1)	12	+
6	No demerits	–	–	–	–	–	(1)	12	+
7	Remove short line	–	–	7	6	(6)	(6)	(6)	
8	Extend last line	–	–	9	8	(8)	12	11	
9	Backspaces	–	–	–	–	–	–	–	(13)

Legend: −/ /+: never/don't care/always
n/(n): (implicitly) shown in example number n

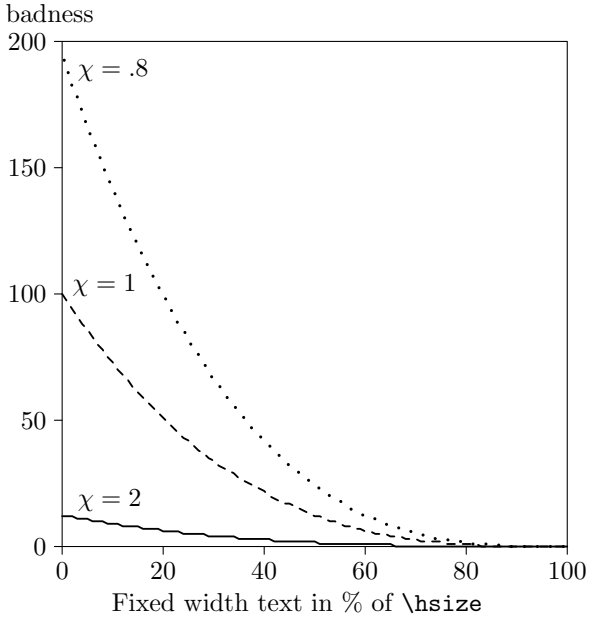


Figure 1: Graphs for $\chi = 0.8$, $\chi = 1$, and $\chi = 2$ when `\parfillskip` ≡ `0pt plus` χ`\hsize`

One reason to select a finite stretchability for `\parfillskip` is to avoid a short last line. Experiments show that a smaller value for the stretchability might create longer last lines. Nevertheless, TEX might also respond with a paragraph that is not changed at all or becomes one line shorter compared to the number of lines with the default setting for `\parfillskip`. Such a paragraph then contains a completely filled last line, i.e., the line is not short.

Figure 1 shows how much badness a last line gets if the text has no additional stretchability. With a stretchability of 0.8`\hsize` a line must cover at least 60% of the line width to become decent (so that spaces do not stretch more than 50% [11, p. 97]).

The smaller the stretchability, the more likely that longer last lines are produced. But too small means artificial demerits can occur and then the last line is not improved at all. A stretchability like `\hsize` − `\parindent` seems to be an ideal choice as it forces the second pass only if the last line with the default setting of `\parfillskip` is shorter than the indent. But a smaller value might pull more strongly, so 0.8`\hsize` seems to be a better value in this respect. A second pass is avoided if the stretchability is at least `\hsize`.

Summary. To support (1) a second pass might be acceptable if the length of the last line is shorter than the indent. A stretchability of `\hsize` or a large fraction of `\hsize` seems to be the best choice

although they also tend to remove the original last line and create paragraphs with one line less. Recommendation (2) is violated more often as very short lines are sometimes absorbed by the former penultimate line which then gets completely filled.

5 Theory: Finite stretchability

First, a few words about the notation: Lowercase math Latin letters stand for dimensions; especially, h is used for `\hsize` and p for `\parindent`; bold face Latin represents glue that consists of three dimensions: the natural width, the stretchability, and the shrinkability. For example, $\mathbf{p}$ is the `\parfillskip` and its three dimensions are written with a Latin p and a superscript: p°, p^+, and p^-. Lowercase Greek letters denote numbers, integers, or rational numbers. Several of them are reserved for often used integers: β is a badness, π a penalty, δ additional demerits, λ the `\linepenalty`, and τ the `\tolerance`. The uppercase Greek letter Λ stands for line demerits and Λ_p for path demerits, i.e., the sum of all line demerits of a set of valid line breaks.

The badness of input (∗) was found by runs with *TUGboat*'s column width. This must not be determined by an experiment though. The badness of lines in which the text occupies only 1 sp is computed as an approximation by the following formula (see [11], p. 97), in which the numerator represents the stretchability used to typeset the line and the denominator stands for the available stretchability in the line:

$$\beta \approx 100\Big(\frac{\text{used stretchability}}{\text{available stretchability}}\Big)^3.$$

Therefore

$$\beta \approx 100\Big(\frac{h-1\,\mathrm{sp}}{\nu h}\Big)^3 \le 100\Big(\frac{h}{\nu h}\Big)^3 = \frac{100}{\nu^3}.$$

This computes the following values rounded to one decimal place, or two if that digit is 5 or would be rounded to 5:

$\nu =$	1	2	3	4	5	6
$100/\nu^3 \approx$	100	12.50	3.7	1.56	0.8	0.46

so that the above found badness values for (∗) seem to be the result of rounding the value $100/\nu^3$ to the nearest integer (and a tie-break rule to get 12).

This proves also that the results about 6`\hsize`, i.e., equality to the default line breaks if no infinite stretchability is in the paragraph, and 2`\hsize`, i.e., only decent last lines occur, are valid for all text widths.

The approximation can be used to find the factor ν that will never produce an underfull line. So the question is: For which ν does the badness reach $\tau = 200$? That is, find ν with

$$100\Big(\frac{h-1\,\mathrm{sp}}{\nu h}\Big)^3 = \tau.$$

The 1 sp is very small compared to h; so it can be dropped. (Actually it is sufficient to find a value that *rounds down* to τ, but this difference is small.) Thus

$$1/\nu^3 \approx 2 \quad\Longrightarrow\quad \nu \approx 1/\sqrt[3]{2} \approx 0.79.$$

Well, that is not a big surprise as it was shown above that $0.8h$ creates a last line with badness 195. If `\hbadness` instead of `\tolerance` is used in the computation, ν must be larger than $1/\sqrt[3]{10} \approx 0.46$. This is the value after which plain TEX reports an underfull line; again an expected result.

Okay, something more complicated: Is it possible to identify a short last line through its badness value? Figure 1 shows that the badness falls very fast when the line is not filled with much text. So it should be possible to distinguish the case if the width of the text is less than the `\parindent` p; maybe not exactly but close.

The problem is to find ν such that, say,

$$100\Big(\frac{h-p}{\nu h}\Big)^3 = \beta+1 \quad\text{and}\quad 100\Big(\frac{h-1.1p}{\nu h}\Big)^3 = \beta.$$

The badness β is not important and gets replaced in the first formula by the second:

$$100\Big(\frac{h-p}{\nu h}\Big)^3 - 100\Big(\frac{h-1.1p}{\nu h}\Big)^3 = 1$$

$$\Longleftrightarrow\qquad \frac{100}{h^3}\Big((h-p)^3-(h-1.1p)^3\Big) = \nu^3$$

$$\Longleftrightarrow\quad \frac{100}{h^3}\Big(0.3h^2p - .63hp^2 + 0.331p^3\Big) = \nu^3$$

$$\Longrightarrow\qquad \sqrt[3]{30\frac{p}{h} - 63\frac{p^2}{h^2} + 33.1\frac{p^3}{h^3}} = \nu.$$

For the *TUGboat* format $p = 20\,\mathrm{pt}$ and $h = 225\,\mathrm{pt}$ so that $p/h = 20/225 \approx 0.0889$, and thus

$$\nu = \sqrt[3]{2.667 - 0.4979 + 0.0233} \approx 1.29.$$

With $\nu = 1.29$ the available stretchability gets $\nu h = 290.25\,\mathrm{pt}$ and the badness values are 35 and 34 as

$$100\Big(\frac{205}{290.25}\Big)^3 = 35.23 \quad\text{and}\quad 100\Big(\frac{203}{290.25}\Big)^3 = 34.21.$$

So a short last line with a badness higher than 34 is not longer than the indentation. Of course, the phrase "short last line" is important as completely filled lines can have badness values larger than 34 when they have to shrink a lot. Values larger than 47 are assigned only to tight lines, as $100/1.29^3 \approx 47$.

The result $\nu = 1.29$ is based on the values of h, p, the factor 1.1, and the difference of 1 for the badness values. But both the factor and the difference can be changed and other specifications for p^+ are possible.

The specification with a stretchability using the factor 5 for \hsize has badness 1 up to 14% of the \hsize, so it is an ideal base for a division of short last lines and longer last lines. For longer last lines the badness is 0 so there is minimal influence on the glue. Smaller last lines have badness 1 and the glue does not stretch very much. Nevertheless the important point is that a short last line for a given length can be identified by the badness if the stretchability is a little bit arranged. For example, using the *TUGboat* format short last lines are distinguished from those that are wider than ≈ 25.250527 pt, which is sufficiently larger than 20 pt, i.e., the \parindent, if the stretchability of \parfillskip is 5.17\hsize.

Figure 2 presents a complete set of macros to check the badness of last lines for the *TUGboat* format; \parfillskip is only changed at the end of paragraphs so displays are not affected. False hits for tight lines (or short lines above a heading) are possible. The author has then the option to enter \lastlineaccepted at the end of the paragraph to suppress the error message in the subsequent runs.

6 Adding natural width to \parfillskip

Recommendation (2) is always obeyed if the natural width of \parfillskip is at least 1 em or 10 pt. The text of experiment 1 is unaffected, but experiment 2 changes with or without stretchability.

Experiment 14: Description
Show that natural width for \parfillskip might produce short last lines.

TEX output

- \parfillskip ← 10 pt plus 1fil:

  2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?

- \parfillskip ← 10 pt:

  2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?

Of course, the new last line in the last paragraph is underfull and has artificial demerits.

O10 (Add short line): The \parfillskip value might create a situation in which TEX prefers or is forced to typeset a new last line. This line might be short.

The natural width must be less than \hsize for normal text, or all last lines will be wider than \hsize and the shrinkability of the material in the last line must be large enough to avoid an overfull line. Otherwise only last lines that contain material of width greater than the difference of \hsize and the natural width shrink or become overfull.

```
% ASSUMPTION: \par has the TeX default
% NOTE: if active a par's end resets \parfillskip
\newif\ifCSLLtrace      % true: show message about
 \CSLLtracetrue   % start and stop on the terminal
\newif\ifCSLLactive      % true: checks are active
\newif\ifCSLLaccepted % true: user said line is OK
\newskip\CSLLpfs    % register to save parfillskip
\newlinechar=`\^^J%        newline in help message
\newhelp\CSLLhelp{checkshortlastline reports a
 warning message: The length^^Jof the line that
 ends at the line number shown in the^^Jlast line
 of the message might be shorter or not
 much^^Jlonger than the indentation or the last
 line is completely^^Jfilled and a little bit
 tight. Then it is a false hit.^^JUse
 \string\lastlineaccepted\space if you can accept
 the line.}
% output message (depending on \CSLLtrace setting)
\def\CSLLmsg(#1){% #1=0/1/2/3: off/on/is off/is on
 \immediate\write\ifCSLLtrace 16 \else -1 \fi
 {>>> \string\checkshortlastline\space
 \ifcase#1 de\or\or not \or already \fi activated;
 \string\parfillskip=\the\parfillskip}}
% execute badness test with customized parfillskip
\def\checkshortlastline{%
 \parfillskip=0pt plus 5.17\hsize  % TUGboat value
 \endgraf\parfillskip=0pt plus 1fil
 \ifnum\badness=1 \errhelp=\CSLLhelp         % HIT
  \errmessage{Notice: Short (or tight) last line}%
 \fi}
% start the check but don't do it twice
\def\activatecheckshortlastline{%
 \ifCSLLactive\CSLLmsg(3)\else\CSLLactivetrue
  % 1st: new def of \par: in hmode do the check
  \def\par{\ifhmode\ifinner\else % horizontal mode
     \ifCSLLaccepted \CSLLacceptedfalse % user: OK
     \else\checkshortlastline\fi\fi   % else check
     \fi\endgraf}%    reset \badness with \endgraf
  % 2nd: save \parfillskip and use default value
  \CSLLpfs=\parfillskip \parfillskip=0pt plus 1fil
  \CSLLmsg(1)\fi}%  write message about activation
\def\stopcheckshortlastline{%  stop check but only
 \ifCSLLactive\CSLLactivefalse  % if it is running
  \let\par=\endgraf % undo changes: reset \par and
  \parfillskip=\CSLLpfs \CSLLmsg(0)%  \parfillskip
 \else\CSLLmsg(2)\fi}
% accept a line that creates a false hit
\let\lastlineaccepted=\CSLLacceptedtrue
```

Figure 2: Macros to check badness of last lines

Experiment 15: Description
Show that a positive natural width for \parfillskip might produce overfull lines.

TEX input

```
\noindent 15. Each interword glue in this
single-line text shrinks.
```

TEX output

• \parfillskip ← 0 pt plus 1fil (plain TEX's default):
15. Each interword glue in this single-line text shrinks.
• \parfillskip ← 10 pt plus 1fil:
15. Each interword glue in this single-line text shrinks.
• \parfillskip ← 10 pt:
15. Each interword glue in this single-line text shrinks.

The last two paragraphs have a badness larger than 10000 as the shrinkability of the line is not large enough to fit the line width [12, §851]. In this case \tracingparagraphs reports no value but outputs the badness as *. Without a valid badness the demerits are set to * too.

Experiment 15 continued: Log file contents

```
1. @secondpass
2. \ninerm 15. The in-ter-word glue in this
   one-line para-graph shrinks.
3. @\par via @@0 b=* p=-10000 d=*
4. @@1: line 1.3- t=0 -> @@0
```

O11 (Glue shrinks early): Interword glue shrinks in last lines early if \parfillskip has a net contribution > 0 pt. When the distance of the natural width of the material in the last line to the right margin is smaller than the natural width of \parfillskip the interword glue starts to shrink.

O12 (Overfull line): If \parfillskip's net contribution to the last line is always greater than 0 pt TEX might typeset an overfull last line. As the total shrinkability in the last line cannot make the text fit into the line width, TEX computes neither the badness nor the demerits for the line.

Overfull lines are signaled by plain TEX as a warning in the log file, on the terminal, and with a black rectangle in the output. (For the experiments in this article a very thin, 1 pt wide, rule is used.) But there is another way to produce an "overfull line" without notification by TEX.

Experiment 16: Description
Show that negative natural width for \parfillskip can create last lines that are wider than \hsize.

TEX output

• \parfillskip ← −15 pt plus 1fil:
1. Please answer if my topic is "in" or "out". TEX:in

O13 (Stick out right): If the glue specification of \parfillskip has the effect of extending the \hsize then the last line might stick out into the right margin without marking this line as overfull.

Negative natural width can be useful in certain applications. For example, in [2, p. 112], a macro is presented that uses \rightskip to provide stretchability in all lines and a negative natural width for \parfillskip to put some material flush right into the white space that the ragged-right setting leaves. The macro is used to break long headers in a table of contents; the material that must be placed in the last line is the page number. If its width is called x, \rightskip gets x plus 2em and $-x$ is assigned to \parfillskip. But again, this article experiments only with changes to \parfillskip alone.

Using finite stretchability. The combination of finite stretchability with a natural width is in some respect a contradiction: The finite stretchability is used to make (1) more likely, i.e., to avoid a short last line, but the natural width adds white space so the line does not appear to be short or rather its badness becomes smaller as less stretchability is needed. Nevertheless earlier experiments show that a badness as small as 1 is enough to change TEX's line-breaking decisions. So it should be possible to find cases where the last line is absorbed or extended with a specification having finite stretchability.

Experiment 17: Description
Show that the combination of natural width and finite stretchability for \parfillskip might produce a longer last line.

TEX output

• \parfillskip ← 10 pt plus \hsize:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
• \parfillskip ← 10 pt plus 0.7\hsize:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a

With a small variation of the text the paragraph gets shorter.

Experiment 18: Description
Show that the combination of natural width and finite stretchability for \parfillskip might remove a short last line.

TEX output

• \parfillskip ← 10 pt plus \hsize:
4′. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

A pure finite stretchability of width p^+ is not equivalent to a specification where the sum of natural width and stretchability equals p^+. This is obvious as the stretchability is reduced when a part of it becomes fixed width.

With natural width $\nu' = (1/10)$\hsize and a stretchability of $(\nu''/10)$\hsize for \parfillskip the input $(*)$ of section 4 has the following badness values, β.

$\nu'' =$	4	5	6	7	8	9	19	29	39
$\beta =$	1137	581	336	211	142	100	10	3	1

Experiment 19: Description
Show that a specification with natural width and finite stretchability might not output a single line.

TEX output
• `\parfillskip` ← 0 pt plus 0.7`\hsize`:
6. One line or two for this text? That is the question, or?
• `\parfillskip` ← 0.1`\hsize` plus 0.6`\hsize`:
6. One line or two for this text? That is the question, or? □

Compare the results with experiment 6: The natural width of `\parfillskip` prevents the absorption of the last line in the second paragraph. TEX executes a second pass which makes the short last line longer. A smaller value for the natural width of `\parfillskip` is sufficient in this case though.

Discussion. The use of a small value for the natural width of `\parfillskip` is ideal to assure (2), although overfull lines can occur and interword spaces shrink early. And short last lines might be produced when TEX has to add a line to the paragraph, i.e., (1) is violated in more cases than before.

With the default setting of `\parfillskip` an author has to check all last lines to ensure that neither (1) nor (2) is violated. With a natural width and infinite stretchability all last lines must be checked at most for a violation of (1) as (2) is either guaranteed or it fails with an error message. In some sense overfull lines are not bad as they point out a significant problem. Either an author rewrites the text or some other action is taken.

An overfull last line for a single paragraph can be solved by the technique mentioned in section 3 but now *with* a factor of −1 (or a fraction of −1) and not just a minus sign:

```
\hskip-1\parfillskip\ \par
```

This cancels only the natural width not the stretchability of `\parfillskip`. The line might be filled completely, violating (2).

Table 3 summarizes all the observations made in the experiments of this section. The entries show that problems are possible with infinite stretchability, and the situation does not improve if the stretchability is reduced to a finite value. Although few experiments were shown with a small variation of values for finite stretchability the findings in section 4 also apply here in many aspects. For example, a too small stretchability produces underfull lines and even lines with artificial demerits.

Summary. Using a nonnegative natural width in `\parfillskip`, for example, 10 pt, is an effective way to have all last lines end before they reach the right margin so that recommendation (2) is always obeyed. Overfull lines might occur, which can be fixed either by rewriting the text or canceling the natural width of `\parfillskip`. But the setting has a side effect: Short last lines might be produced even with finite stretchability.

Table 3: Observations for natural width & stretchability

Specification of `\parfillskip` ($h \equiv$ `\hsize`)							
	natural width	0pt	<0pt	10pt	10pt	10pt	10pt
	stretch	1fil	1fil	1fil	h	.7h	0pt
	shrink	0pt	0pt	0pt	0pt	0pt	0pt
Observation							
1	Short line	1		(1)	(1)	(1)	(1)
2	Completely filled	2		–	–	–	–
3	Glue stretches	–		–	17	17	(3)
4	2nd pass forced	–		15	(15)	17	15
5	Underfull line	–		–	–	(1)	(3)
6	No demerits	–		15	(15)	(1)	15
7	Remove short line	–		–	18	(18)	(18)
8	Extend last line	–		–	17	17	(5)
9	Backspaces	–		–	–	–	–
10	Add short line	–		14	(14)	(14)	14
11	Glue shrinks early	–		15	(15)	(15)	15
12	Overfull line	–		15	(15)	(15)	15
13	Stick out right	–	(16)	–	–	–	–

Legend: −/ /+: never/don't care/always
$n/(n)$: (implicitly) shown in example number n

7 Theory: Natural width and stretchability

The formula to compute the badness introduced in section 5 can be used if the sum of the width of the text and p° is not greater than h. The numerator, which contains the used stretchability, has to apply not only the width of the text but also p°. But if the sum of these two values is greater than h then the glue in the last line must shrink and a different calculation is required. Well, the formula stays the same but now the used shrinkability and the available shrinkability form the quotient.

If the length of the text in the last line is named t then the used stretchability is $h-t-p^\circ$ if $t+p^\circ \le h$. Otherwise, i.e., $h < t+p^\circ < 2h$, the required shrinkability is $t+p^\circ-h$. This amount must come from the shrinkability of the text as $p^- = 0\,\text{pt}$. If $t+p^\circ$ is too large then the line gets overfull. Therefore, in most cases the value of p° should not be very large and less than h, unless the text has unusual shrinkability.

In experiment 19 TEX uses different line breaks for a specification with pure stretchability and for one in which the sum of a nonnegative natural width and a smaller stretchability equals the stretchability of the first specification. In the experiment different passes occur. But if the last lines are identical and all glue stretches, both specifications typeset the paragraph in the same pass. The specification with

$p^\circ = 0\,\mathrm{pt}$ has an equal or greater badness in the first pass if the badness values are less than 100; in a second pass, its badness is equal or smaller. Compare the badness values shown before experiment 19, for input (∗), with the corresponding values in section 4.

For the proof of this statement the following shortcut is defined. The sum of all horizontal glue in the last line, i.e., interword spaces and skips, except the `\parfillskip`, is named **g**. It has natural width g°, stretchability g^+, and shrinkability g^-, as with other glue parameters.

If $\nu = \nu' + \nu''$ and the last line contains text of width t with stretchability g^+ the two badness values for the specifications `0pt plus` ν`\hsize` and ν'`\hsize plus` ν''`\hsize` are approximately

$$100\Big(\frac{h-t}{\nu h + g^+}\Big)^3 \quad \text{and} \quad 100\Big(\frac{h-t-\nu' h}{\nu'' h + g^+}\Big)^3.$$

To make sure that the badness values are not larger than the tolerance for the second pass, $\tau = 200$, the relation $\sqrt[3]{2}\nu'' \geq 1 - \nu' - (t + \sqrt[3]{2}g^+)/h$ or, better, $\nu'' \geq (1-\nu')/\sqrt[3]{2}$ must hold.

In a first pass with badness < 100 the quotient of the second specification that is cubed obeys

$$0 \leq \frac{h-t-\nu' h}{\nu'' h + g^+} < 1 = \frac{\nu' h}{\nu' h}.$$

The quotient of the first specification, $(h-t)/(\nu h + g^+)$, is the *mediant*, i.e., the quotient of the sum of the numerators and the denominators of the two quotients on the left and the right of 1, and therefore

$$\frac{h-t-\nu' h}{\nu'' h + g^+} < \frac{h-t}{\nu h + g^+} < \frac{\nu' h}{\nu' h} = 1.$$

A similar argument shows

$$\frac{h-t-\nu' h}{\nu'' h + g^+} > 1 \Longrightarrow \frac{h-t-\nu' h}{\nu'' h + g^+} > \frac{h-t}{\nu h + g^+} > 1.$$

The relations are not changed when they are cubed and multiplied by 100 to get the badness. TEX's computation [12, §108] is monotone but only an approximation: Both relations "less than" and "greater than" should also be "or equal to".

Using the `\tracingparagraphs` output. TEX has a parameter to report line-breaking decisions and write them to the log file: `\tracingparagraphs` [19]. Its output has already been shown several times, for example, in the discussion after experiment 3. This trace shows all the ways for TEX to typeset the text of a paragraph with the current values of the line-breaking parameters. The glue `\parfillskip` is not explicitly mentioned in a trace but the possible last lines carry in their badness and thus in their line demerits the influence of this parameter.

Experiment 16 sets `\parfillskip` to a strange value, creating for the last line a solution that is not part of the network of line breaks that is built from the default settings of plain TEX. But the effect is limited to the last line — and its previous line if it becomes the last line — as no other line breaks can be affected by the bad setting of `\parfillskip`, i.e., all sets of line breaks up to the start of the penultimate line are also valid line breaks for the default setting. A second important aspect of giving `\parfillskip` finite values is the interaction with infinite stretchability in the text. For example, the trace for experiment Xb with the default setting of `\parfillskip` and a positive value for `\looseness` to force a second pass lists five different paths; with a stretchability of 6`\hsize` this number increases to nine. But as stated before, input with infinite stretchability is not analyzed in this article.

Therefore only the trace of a "normal text" with a reasonable `\parfillskip` is considered. Experiment 6 with a forced second pass outputs this trace:

```
1. @secondpass
2. \ninerm 6. One line or two for this text?
    That is the ques-
3. @\discretionary via @@0 b=84 p=50 d=11336
4. @@1: line 1.1- t=11336 -> @@0
5. tion,
6. @ via @@0 b=2 p=0 d=144
7. @@2: line 1.2 t=144 -> @@0
8. or?
9. @\par via @@0 b=65 p=-10000 d=5625
10. @@3: line 1.3- t=5625 -> @@0
11. @\par via @@2 b=0 p=-10000 d=100
12. @\par via @@1 b=0 p=-10000 d=5100
13. @@4: line 2.2- t=244 -> @@2
```

This trace contains three paths which are described in Table 4. All the paths have been selected with some specification of `\parfillskip`: The single line and the first two-line solution that is typeset with the default setting are shown in experiment 6, while

Table 4: Badness, penalties, and additional demerits of the line breaks for the three paths of the trace listing

		`\par via @@` (∗ is typeset)		
`@@`	Class	0_0	$*2_0$	1_0
1	l			84_{50}
2	d		2	
3	t	65		
4	d		0	0^{f}
# lines =		1	2	2
$\sum$ badness =		65	2	84
# a/d/f =		0/0/0	0/0/0	0/0/1
$\Lambda_{\mathrm{p}}(10)$ =		5625	244	16436
Last line with `\parfillskip` $\equiv \mathbf{p} = (0\,\mathrm{pt}, 0.8h, 0\,\mathrm{pt})$:				
class : data =		∗ t : 65	v : 164^{a}	v : 115^{f}
$\Lambda_{\mathrm{p}}(10)[\mathbf{p}]$ =		5625	40420	31961
stretch/pt =		n/a	0	1.92706

the third path is typeset in experiment 19. In the following, we analyze how a setting of `\parfillskip` can be determined to select one of the possible paths given the data of Table 4.

Here is a brief description of the table. The headline shows the three paths that have lines in the trace which start with `@\par`, i.e., lines 9, 11, and 12 and list the feasible breakpoint. The typeset `\par` column gets an asterisk. Two lines at the left show in each row the number of the feasible breakpoint and the first letter of the associated fitness class (coded as a number in the trace): very loose, loose, decent, tight. Each feasible breakpoint of a path gets an entry of the form

$$\text{badness}\ \begin{matrix}\text{additional demerits as letters a, d, and f}\\ \text{penalty at the break if} \neq -10000\end{matrix}$$

in which the applied additional demerits must be determined by a calculation using the demerits `d` of the trace line [21]. The other two values are listed directly in the trace line as `b` and `p`. The four rows at the end of the table, i.e., the number of lines, the sum of the badness values, the occurrences of the additional demerits, and the path demerits, are calculated from the entries in each column.

As mentioned earlier, the sum of all line demerits for a set of line breaks is called the path demerits; the smallest value of all path demerits are the total demerits of the paragraph and the associated set of line breaks is used by TEX to typeset the text [11, p. 97].

The following formula calculates the line demerits Λ_ι for line number ι [11, p. 98; 20, section 2]:

$$\Lambda_\iota = (\lambda + \beta_\iota)^2 + \operatorname{sgn}(\pi_\iota)\pi_\iota^2 + \delta_\iota$$

where λ is the `\linepenalty`, β_ι stands for the badness of the line, π_ι represents the penalty at the line break, and δ_ι is the sum of the applicable additional demerits, i.e., the values of `\finalhyphendemerits`, `\doublehyphendemerits`, and `\adjdemerits`.

The table must be extended compared to previous instances; see [20] and [21]. To get badness values that distinguish last lines of different length, a second trace is generated with the finite stretchability of 0.8`\hsize` for `\parfillskip`. (The trace can be generated directly with this `\parfillskip` and only one additional line would be needed.) The fitness class and the new entry for the last line, the new path demerits, and the sum of all stretchability of the material in each last line are shown in three more lines. The last line in Table 4 contains data that cannot be extracted from the trace. It is shown to allow exact computations but the influence is so small that the data can be dropped.

With the default `\parfillskip` — and therefore also with a stretchability of 6`\hsize` — the path of column $\mathtt{2}_0$ is typeset and with the 0.8`\hsize` column $\mathtt{0}_0$ is chosen. (That is known, as the stretchability of $p^+ = h$ proved it in experiment 6.) But what is the maximum value for p^+ to select column $\mathtt{0}_0$? What setting for `\parfillskip` causes TEX to select column $\mathtt{1}_0$? These theoretical questions are answered in the next subsections.

Not all paths shown in a path table might be distinguishable through settings for `\parfillskip` as the difference can lie in the line breaks before the last line. Two sets of line breaks cannot be distinguished by a setting of `\parfillskip` if their last lines are identical. In this case the column head of the path table shows the same feasible breakpoint number but with a different subscript. No such case appears in Table 4.

And the final remark: A last line can be selected with certainty by a setting of `\parfillskip` only if its badness with **p** differs from all other last lines. A change of the stretchability of `\parfillskip` produces different line demerits for last lines only if the change assigns different badness values to the last lines. For this reason the `\parfillskip` $\mathbf{p} = (0\,\mathrm{pt}, 0.8h, 0\,\mathrm{pt})$ was selected in the path table as it often assigns different badness values to short lines of different lengths.

Absorb short last line. To get path $\mathtt{0}_0$ of Table 4 instead of $\mathtt{2}_0$ their demerits must obey the relation

$$\Lambda_{\mathrm{p}[\mathtt{2}_0]}(10)[\mathbf{p}'] > \Lambda_{\mathrm{p}[\mathtt{0}_0]}(10)[\mathbf{p}']$$

for the unknown `\parfillskip` $\mathbf{p}'$. The constant `\linepenalty` is not changed and therefore "(10)" is dropped together with the $\mathbf{p}'$ in the notation and only the prime is moved to Λ_p to signal that a different `\parfillskip` is used. So the notation of the relation is simplified to

$$\Lambda'_{\mathrm{p}[\mathtt{2}_0]} > \Lambda'_{\mathrm{p}[\mathtt{0}_0]}.$$

Note: $\Lambda_{\mathrm{p}[\mathtt{0}_0]}$ is not changed when the stretchability changes as it is a tight line; i.e., $\Lambda'_{\mathrm{p}[\mathtt{0}_0]} = \Lambda_{\mathrm{p}[\mathtt{0}_0]}$.

Let's say that with p'^+ the badness of the last line in path $\mathtt{2}_0$ changes from β to $\beta\chi$. To get an approximation for p'^+, the formula for the approximation of badness values is used twice. First calculate

$$\beta \approx 100\Big(\frac{\text{used stretchability}}{0.8h + g^+}\Big)^3$$

where g^+ is the stretchability in the text.

For all p'^+ the used stretchability is the same; only the available stretchability changes. Instead of $0.8h+g^+$ the available stretchability is now $p'^+ + g^+$.

Therefore

$$\beta\chi \approx 100\Big(\frac{\text{used stretchability}}{p'^{+}+g^{+}}\Big)^3.$$

Dividing the quotients for $\beta\chi$ and β gives a formula for p'^{+}. First

$$\chi = \frac{\beta\chi}{\beta} \approx \Big(\frac{0.8h+g^{+}}{p'^{+}+g^{+}}\Big)^3$$

and thus

$$p'^{+} \approx \frac{0.8h+g^{+}}{\sqrt[3]{\chi}} - g^{+} = \frac{0.8}{\sqrt[3]{\chi}}h - (1-\frac{1}{\sqrt[3]{\chi}})g^{+}.$$

The value of h is known and g^{+} is given in the path table. Only χ must be determined from the above stated inequality.

To calculate χ the change of the line demerits must be analyzed using the formula for line demerits. The relation $\Lambda'_{\mathrm{p}[2_0]} > \Lambda'_{\mathrm{p}[0_0]} = \Lambda_{\mathrm{p}[0_0]}$ becomes

$$\Lambda_{\mathrm{p}[2_0]} - (\lambda+\beta)^2 - \delta_a[\beta\chi < 100] + (\lambda+\beta\chi)^2 > \Lambda_{\mathrm{p}[0_0]}$$

as (i) only the last line changes its demerits as it changes its badness, (ii) penalties in the last line are not affected, and (iii) the `\adjdemerits`, δ_a, might disappear if the last line is no longer very loose. (This is expressed with *Iverson's convention*: If the condition in the brackets is true then the value of the bracket is 1, otherwise 0.) Actually the bracket can be set to 1 as it is already known that the badness is less than 100 because `\hsize` is sufficient for p'^{+}.

$$-2\lambda\beta - \beta^2 + 2\lambda\beta\chi + \beta^2\chi^2 > \Lambda_{\mathrm{p}[0_0]} - \Lambda_{\mathrm{p}[2_0]} + \delta_a$$

$$\iff \qquad \chi^2 + 2\frac{\lambda}{\beta}\chi - 1 - 2\frac{\lambda}{\beta} > \frac{\Lambda_{\mathrm{p}[0_0]} - \Lambda_{\mathrm{p}[2_0]} + \delta_a}{\beta^2}$$

Adding $0 = (\lambda/\beta)^2 - (\lambda/\beta)^2$ to the left side generates two quadratic terms on this side. Now the inequality can be solved for χ.

$$\Big(\chi + \frac{\lambda}{\beta}\Big)^2 > \frac{\Lambda_{\mathrm{p}[0_0]} - \Lambda_{\mathrm{p}[2_0]} + \delta_a}{\beta^2} + \Big(1+\frac{\lambda}{\beta}\Big)^2$$

$$\implies \qquad \chi > \sqrt{\frac{\Lambda_{\mathrm{p}[0_0]} - \Lambda_{\mathrm{p}[2_0]} + \delta_a + (\lambda+\beta)^2}{\beta^2}} - \frac{\lambda}{\beta}.$$

The last step is of course valid only if $\Lambda_{\mathrm{p}[0_0]} - \Lambda_{\mathrm{p}[2_0]} + \delta_a + (\lambda+\beta)^2 \geq 0$. This condition means that the path demerits $\Lambda_{\mathrm{p}[2_0]}$ without the line demerits of its last line must not be greater than the line demerits of the path $\mathtt{0}_0$. This is of course true, otherwise path $\mathtt{2}_0$ would not be typeset instead of the single line.

With plain TEX's value $\delta_a = 10000$ [11, p. 98] and with the numbers listed in Table 4, χ must obey

$$\chi > \sqrt{\frac{5625 - 40420 + 10000 + 174^2}{164^2}} - \frac{10}{164}$$

$$= \sqrt{\frac{5481}{26896}} - \frac{10}{164} \approx 0.3904.$$

Thus $\chi = 0.391$ is sufficient. The badness becomes $\approx 0.391 \times 164 \approx 64$ for the new last line of column $\mathtt{2}_0$. Thus, as $g^{+} = 0\,\mathrm{pt}$ for column $\mathtt{2}_0$,

$$p'^{+} \approx \frac{0.8}{\sqrt[3]{0.391}}h \approx 1.0940h.$$

Experiment (verification): Description
Show that the computed value — with an accuracy of two digits after the decimal point — selects the requested path.

TEX output
• `\parfillskip` ← 0 pt plus 1.09`\hsize`:
6. One line or two for this text? That is the question, or? ▯

Of course, badness is only a heuristic and there is only an approximation for its computation, so the result cannot be exact. The factor for h in p'^{+} so that $\mathtt{0}_0$ is used instead of $\mathtt{2}_0$ is ≈ 1.09088.

Extend short last line. With 0.8`\hsize` the path demerits of column $\mathtt{1}_0$ are lower than the path demerits of $\mathtt{2}_0$; that means the path of $\mathtt{1}_0$ is now preferred to $\mathtt{2}_0$. But the path $\mathtt{0}_0$ receives the lowest demerits. The latter choice can be eliminated with a positive natural width, for example, 0.02`\hsize` is sufficient for the stretchability 0.8`\hsize` — or better 0.78`\hsize` according to the above theory — to switch to $\mathtt{1}_0$ as the single line is already very tight. (Note as the badness is 65 the glue in the single line shrinks already by more than 86%. So a small natural width excludes this path.)

Nevertheless it is interesting to find a maximal value for the stretchability to make TEX choose $\mathtt{1}_0$. The approach of the previous subsection can be applied here too. But as now two badness values are changed, there are two factors χ and χ'. As the penultimate line in $\mathtt{1}_0$ has fitness class "loose" there is no δ'_a.

$$\Lambda_{\mathrm{p}[2_0]} - (\lambda+\beta)^2 + (\lambda+\beta\chi)^2 - \delta_a[\beta\chi < 100] > \Lambda_{\mathrm{p}[1_0]} - (\lambda+\beta')^2 + (\lambda+\beta'\chi')^2$$

The value χ' depends on the stretchability of the text in last line, i.e, on the 1.92706 pt as noted in Table 4. But to simplify the calculations χ' is replaced by χ as they are nearly equal in this case. As shown before,

$$\chi \approx \Big(\frac{0.8h+g^{+}}{p'^{+}+g^{+}}\Big)^3 \quad \text{and} \quad \chi' \approx \Big(\frac{0.8h+g'^{+}}{p'^{+}+g'^{+}}\Big)^3$$

so that

$$\frac{\chi'}{\chi} = \Big(\frac{0.8h+g'^{+}}{0.8h+g^{+}} \cdot \frac{p'^{+}+g^{+}}{p'^{+}+g'^{+}}\Big)^3.$$

Both quotients on the right side are ≈ 1 as a short line with at most one space is extended to a little bit longer line with maybe one more space. Here $g^{+} = 0\,\mathrm{pt}$ and $g'^{+} = 1.92706\,\mathrm{pt} < 0.0086h$. To

drop such small values compared to $0.8h$ and p'^{+} in a calculation that is based only on approximations should not do much harm.

This time it is expected that $\beta\chi > 100$ holds so the term with δ_a is dropped; i.e., the value of `\adjdemerits` is still applied in the new set of line breaks. Therefore

$$(\beta^2 - \beta'^2)\chi^2 + 2\lambda(\beta - \beta')\chi > \Lambda_{\mathrm{p}[1_0]} - (\lambda + \beta')^2 - \Lambda_{\mathrm{p}[2_0]} + (\lambda + \beta)^2.$$

The formulas get rather wide for this small column width so instead of stating the difference between path demerits and part of the line demerits of the last line, the line demerits of the first line are used. That is

$$\Lambda'_1 = \Lambda_{\mathrm{p}[1_0]} - (\lambda + \beta')^2 - \delta_f$$
$$\Lambda_1 = \Lambda_{\mathrm{p}[2_0]} - (\lambda + \beta)^2 - \delta_a.$$

So δ_a gets back and δ_f is introduced into the inequality:

$$(\beta^2 - \beta'^2)\chi^2 + 2\lambda(\beta - \beta')\chi > \Lambda'_1 + \delta_f - \Lambda_1 - \delta_a.$$

After the division with $\beta^2 - \beta'^2 > 0$ and the addition of $\lambda^2/(\beta + \beta')^2$ this relation becomes

$$\Big(\chi + \frac{\lambda}{\beta+\beta'}\Big)^2 > \frac{\Lambda'_1 + \delta_f - \Lambda_1 - \delta_a}{\beta^2 - \beta'^2} + \frac{\lambda^2}{(\beta+\beta')^2}$$

and then

$$\chi > \sqrt{\frac{\Lambda'_1 + \delta_f - \Lambda_1 - \delta_a}{\beta^2 - \beta'^2} + \frac{\lambda^2}{(\beta+\beta')^2}} - \frac{\lambda}{\beta+\beta'}.$$

With the numbers of Table 4, $\Lambda'_1 = 94^2 + 50^2 = 11336$ and $\Lambda_1 = 12^2 = 144$, as well as $\delta_f = 5000$ [11, p. 98], so that

$$\chi > \sqrt{\frac{11336 + 5000 - 144 - 10000}{164^2 - 115^2} + \frac{10^2}{279^2}} - \frac{10}{279}$$
$$\approx 0.6739 - 0.0358 = 0.6381$$

and then follows as before

$$p'^{+} = \frac{0.8}{\sqrt[3]{\chi}}\,h \approx \frac{0.8}{\sqrt[3]{0.639}}\,h \approx 0.929h.$$

Therefore the specification 0.02`\hsize` for the natural width and 0.90`\hsize` for the stretchability of `\parfillskip` avoids the selection of column $\mathtt{0}_0$ and makes TEX typeset $\mathtt{1}_0$. Again the result is not exact: Setting $p'^{+} = 0.9086$`\hsize` together with a natural width of 0.02`\hsize` does the job too.

Experiment (verification): Description
Show that the abovementioned values for natural width and stretchability of `\parfillskip` typeset the selected path.

TEX output
• `\parfillskip` ← 0.02`\hsize` plus 0.90`\hsize`:
6. One line or two for this text? That is the question, or? ∎

Comparing finite stretchability. The length of a short last line has no influence on TEX's line-breaking decisions with the default plain TEX settings. TEX picks the set of line breaks that gives the smallest sum of line demerits for all but the last line. If the text has μ lines then

$$\sum_{\iota=1}^{\mu-1} \Lambda_\iota + \delta_\mu < \sum_{\iota=1}^{\mu-1} \Lambda'_\iota + \delta'_\mu$$

where the non-primed line demerits Λ_ι represent the typeset output. Then 100 demerits have to be added for both short last lines to get the path demerits.

With a finite stretchability, different lengths of the last line generate different line demerits, so instead of 100 demerits on each side of the inequality different numbers are added, and that might change the relation symbol from less than to greater than for the path demerits. This means that finite stretchabilities can be compared by the maximum value of demerits for all but the last line by the point where the relation symbol change will happen. That the demerits can increase significantly was seen with experiment 11 in section 4.

Here are the differences for the line demerits of short last lines for four stretchabilities (without looking at the stretchability of the text):

p^+	$2\% \to 4\%$	$4\% \to 6\%$	$6\% \to 8\%$	$8\% \to 10\%$
h	1212	955	905	855
$.9h$	2421	1524	1920	1356
$.8h$	3740	3883	3320	2817
$.7h$	8800	8789	6804	6855

Therefore to extend the last line from $0.02h$ to $0.04h$ with $p^+ = h$ TEX saves 1212 demerits so that the sum of line demerits for the rest of the paragraph might have up to 1211 demerits more than the original typeset lines. And it can be $1212 + 955 - 1$ demerits if the last line can be extended to $0.06h$.

For $p^+ = 0.8h$ and $p^+ = 0.7h$ all, and for $p^+ = 0.9h$ all but the last listed line length produce very loose last lines if the text in this line has no stretchability, and `\adjdemerits`, $\delta_a = 10000$, are charged except the penultimate line is loose. In this case it can be said that $p^+ = 0.9h$ tolerates ≈ 2.0 times more demerits than $p^+ = h$ when a line of $0.02h$ is extended. The factor for $p^+ = 0.8h$ lies at ≈ 3.5 and for $p^+ = 0.7h$ it is ≈ 8.0. When the default `\adjdemerits` are applied to the last line only $p^+ = 0.7h$ starting at $0.06h$ has positive factors.

8 Replacing stretchability by shrinkability

The overfull line problem that exists in the combination of natural width with stretchability can be

addressed by specifying natural width together with finite shrinkability instead of stretchability in the glue parameters of \parfillskip. Recall that in the case of stretchability the interword space can change without limit but with shrinkability, only its given maximum is allowed. Also, the shrinkability cannot be of infinite order as TEX throws an error with such a specification [12, §853, §826]. Besides this, there is a fundamental difference between a specification that has stretchability and one with natural width and shrinkability. For example, if all values are specified as \hsize then in the first case short lines have a high badness as more stretchability must be used. In the second case lines that contain a lot of text have greater badness as the shrinkability gets higher to compensate for the natural width.

Shrinkability larger than the natural width. A shrinkability that is larger than the natural width might make TEX put some text into the right margin as it creates a situation similar to negative natural width with stretchability; see experiment 16.

Experiment 20: Description
Show that the shrinkability of \parfillskip should not be larger than its natural width.

TEX output

- \parfillskip ← \hsize minus 3\hsize:
  1. Please answer if my topic is "in" or "out". TEX: in
- \parfillskip ← 0.9\hsize minus 0.9\hsize + 1 em:
  1. Please answer if my topic is "in" or "out". TEX: in

Note again that TEX gives no warning or error message in these cases.

Shrinkability equals natural width. When the input (∗) of section 4 is processed by TEX with a setting of \parfillskip=ν\hsize minus ν\hsize the badness β is:

$\nu =$	1	2	3	4	5	6	7	8	9
$\beta =$	0	12	30	42	51	57	63	66	70

As with this specification more text in the last line of the paragraph results in a higher badness value, it seems that in this case the input

```
\noindent\hbox to 0.9\hsize{\hss}    (**)
```

is also of interest, i.e., the line is 90% filled without additional shrinkability.

$\nu =$	1	2	3	4	5	6	7	8	9
$\beta =$	73	86	90	92	94	95	95	96	96

In contrast to a specification with natural width and stretchability, this specification can cancel the natural width. Nevertheless, short new lines are preferred by TEX here too as the badness of long last lines is so high. That is, the effect of O10 "Add short line" is also seen with this specification.

Experiment 21: Description
Show that short last lines can occur with natural width and the same amount of shrinkability for \parfillskip.

TEX output

- \parfillskip ← \hsize minus \hsize:
  2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?
  4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a

The text of experiment 4 demonstrates that even a single letter on the last line is not moved to the penultimate line to form the new last line. Actually it will never happen that a paragraph with a single word of width up to 0.17\hsize in the last line is reduced by one line with the specification of \parfillskip used in experiment 21; see the next section. Nevertheless, it does not mean that a line that ends at the right margin is impossible.

Experiment 22: Description
Show that a last line might end flush right.

TEX output

- \parfillskip ← \hsize minus \hsize:
  2′. Has the last line of this paragraph badness 0 and has no interword space to stretch?! Do they shrink now?!

If the specification uses a fraction of \hsize, \parfillskip=(ν/10)\hsize minus (ν/10)\hsize, the input (∗) always produces last lines with infinite badness, as the text width plus the natural width of \parfillskip is shorter than \hsize and there is no stretchability. Such specifications have infinite badness for very short lines but the badness becomes 0 at the moment the text of the line reaches $(1-\nu/10)$\hsize to make a perfect fit with the line width. Longer lines get more and more badness.

When such a specification is confronted with a situation in which an interword space is available in a short last line, this glue has to stretch.

Experiment 23: Description
Show that a short line gets extremely spaced out if it has to stretch.

TEX output

- \parfillskip ← 0.8\hsize minus 0.8\hsize:
  3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
- \parfillskip ← 0.7\hsize minus 0.7\hsize:
  3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

The second specification produces a different line break in the text of experiment 3 than plain TEX's default setting or a stretchability of 0.7\hsize does; see experiment 12. It is like experiment 3 with

\parfillskip set to 0 pt except that here the natural width limits the amount by which the space has to stretch. Of course, the last line has artificial demerits. This effect helps to avoid short last lines, for example, with texts 2, 4, and 6. They cannot have a second line as it would be a line with artificial demerits, i.e., a line that TEX does not consider as long as there are other ways to typeset the text.

Experiment 24: Description
Show that a shorter paragraph is possible with natural width and shrinkability.

TEX output

• \parfillskip ← 0.7\hsize minus 0.7\hsize:
2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
6. One line or two for this text? That is the question, or?

Note that the text of experiment 6 does not end flush right; its badness is 97. All glue items with shrinkability are involved in the process to shrink the line, and only with a badness of exactly 100 does the natural width of \parfillskip vanish completely.

Shrinkability smaller than natural width. If the shrinkability is smaller than the natural width recommendation (2) is supported: There is always some white space at the end of the last line. This is similar to the case of natural width and stretchability. But as the results of section 6 show, a positive contribution to the width of the last line by \parfillskip can produce overfull lines.

The following specification

```
\parfillskip=\hsize minus \hsize
\advance\parfillskip by -1.25\parindent
                    minus -1.25\parindent
\advance\parfillskip by 0pt minus -10pt
```

states that the natural width needs text with the width of 1.25\parindent to fill the line and if there is a lot of text the shrinkability must leave at least 10 pt of white space, as this is the amount by which it is shorter than the natural width.

Experiment 25: Description
Show some effects of the above specification.

TEX output

• \parfillskip ← \hsize − 1.25\parindent minus \hsize − 1.25\parindent − 10 pt:
1. Please answer if my topic is "in" or "out". TEX: in
2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?
6. One line or two for this text? That is the question, or?
15. Each interword glue in this single-line text shrinks.

Texts 1 and 2 both produce an underfull last line with badness 10000 and no demerits.

When lines are very short and contain only one space this space must stretch a lot. The text must at least cover the width that is subtracted from \hsize in \parfillskip's natural width.

Experiment 26: Description
Show that with such settings a last line must be as wide as the amount that \parfillskip's natural width is smaller than \hsize.

TEX output

• \parfillskip ← \hsize − 1.1\parindent minus \hsize − 1.1\parindent − 10 pt:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
13. a

• \parfillskip ← \hsize − 1.25\parindent minus \hsize − 1.25\parindent − 10 pt:
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
13. a

In the first case the text of experiment 13 has badness 3, in the second it is 88. A larger factor, for example, 1.5, in front of the \parindents creates an underfull last line with badness 10000 and no demerits.

Negative values. A negative natural width for \parfillskip behaves as if the line width increases and overfull lines without the overfull rule are easily produced.

Experiment 27: Description
Show that lines might not end at the right margin if the natural width is negative.

TEX output

• \parfillskip ← −10 pt minus −10 pt:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

• \parfillskip ← −10 pt minus \hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

Negative shrinkability outputs an underfull last line if the sum of the text width and the natural width is < \hsize; overfull lines are possible too.

Experiment 28: Description
Show that last lines can be underfull or overfull if the shrinkability is negative.

TEX output

• \parfillskip ← 0.75\hsize minus −0.1\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

• \parfillskip ← 0.87\hsize minus −0.1\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?

With such specifications the stretch- or shrinkability of the text in the last line must work together perfectly with the natural width to output valid last lines; otherwise the line is under- or overfull.

Discussion. Specifications with natural width and shrinkability for `\parfillskip` can replace a specification that uses only stretchability or stretchability and natural width. But they have different properties. Long last lines receive high badness values so TEX now prefers line breaks that create a short last line. This contradicts recommendation (1). By the same token a short last line is not absorbed by the penultimate line to form a new last line.

Figure 3 shows the badness values for three specifications when fixed width text is used; note that the scale of the y-axis is not the same as in Fig. 1. The length of the text is again reported on the x-axis as a percentage of the `\hsize`. The first 30 values for `0.7\hsize minus 0.7\hsize` have infinite badness.

A setting where the natural width equals the shrinkability seems to be the best choice. If the value is too small, for example, only 0.7`\hsize`, problems like underfull or extremely spaced out last lines easily occur. If the shrinkability and the natural width both equal `\hsize`, recommendation (2) might not be obeyed, although it seems to be rare as in all last lines the glue shrinks. The white space of the natural width disappears only when the badness equals 100. If the shrinkability is smaller than the natural width, recommendation (2) is always obeyed although now overfull lines might occur. Table 5 lists all observations found in the experiments.

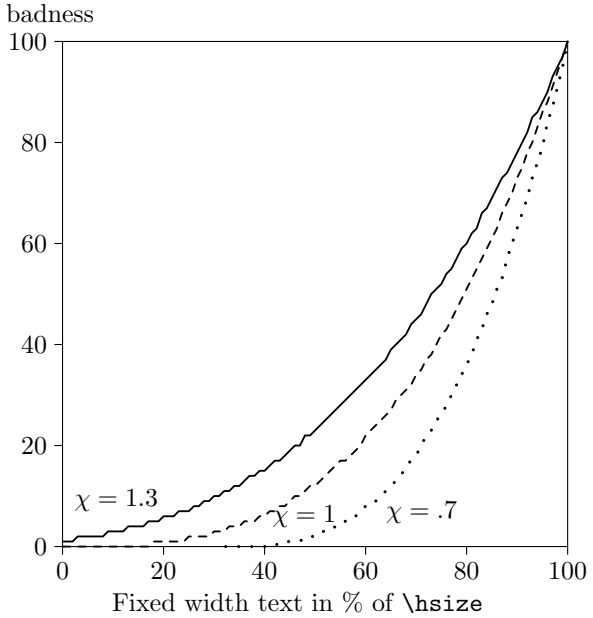


Figure 3: Graphs for $\chi = 0.7$, $\chi = 1$, and $\chi = 1.3$ when `\parfillskip` $\equiv \chi$`\hsize minus` χ`\hsize`

Table 5: Observations for natural width & shrinkability

Specification of `\parfillskip` ($h \equiv$ `\hsize`, $x > 0\,\text{pt}$, $y \equiv h - 1.25$`\parindent`, $z \equiv y - 10\,\text{pt}$)

	natural width	0pt	x	h	$.7h$	y	-10pt	x	
	stretch	1fil	0pt	0pt	0pt	0pt	0pt	0pt	
	shrink	0pt	$>x$	h	$.7h$	z	x	$-.1h$	
Observation									
1	Short line	1		21	24	25			
2	Completely filled	2		22	(22)	–			
3	Glue stretches	–		–	23	26			(28)
4	2nd pass forced	–		–	23	25			(28)
5	Underfull line	–		–	23	25			(28)
6	No demerits	–		–	23	25			(28)
7	Remove short line	–		–	24	–			
8	Extend last line	–		–	–	25			
9	Backspaces	–		–	–	–			
10	Add short line	–		21	–	25			
11	Glue shrinks early	–		+	24	25			(28)
12	Overfull line	–		–	–	25			(28)
13	Stick out right	–	(20)	–	–	–	(27)		

Legend: $-$/ /+: never/don't care/always
$n/(n)$: (implicitly) shown in example number n

As the glue in the last line of a paragraph usually shrinks the problem with the spacing around displayed equations exists here too.

Summary. The use of natural width and shrinkability in the specification of `\parfillskip` works well but the dimensions must be selected carefully and short last lines might occur more often than before. If the values of natural width and shrinkability are equal and smaller than `\hsize`, many problems occur when the last line is not filled with enough text. If the shrinkability is larger or smaller than the natural width, overfull lines might appear.

9 Theory: Natural width and shrinkability

As mentioned in section 7, the formula to calculate the approximation for the badness in cases that involve glue that shrinks is the same as the one used in section 5 but now with a quotient built from the used shrinkability and the available shrinkability. For example, let's find the width t that a text without shrinkability in the last line can have so that the badness computes to 0 when $p^\circ = p^- = h$. The last line has a width of $t + h$ as p° is added to the width of the material. So the used shrinkability is t to make this sum fit the line width h. The available shrinkability is $p^- = h$ as there is no shrinkability in the text by assumption. Thus:

$$100\Big(\frac{t}{h}\Big)^3 < 0.5 \quad\Longrightarrow\quad t < \sqrt[3]{1/200}h \approx 0.17h.$$

To state it the other way: A line that contains material with a width up to $0.17h$ has badness 0 for sure when $\mathbf{p} = (h, 0\,\mathrm{pt}, h)$; see also Fig. 3.

It was stated above that a paragraph will never be shortened by one line when $p^\circ = p^- = h$ and the last line has a single word of width $\le 0.17h$, i.e., the value just found. Such a statement cannot be shown by experiment; it requires proof.

But first a note on the notation: Λ_ι is written for the line demerits of line number ι that TEX typesets if `\parfillskip` has the default setting; $\bar\Lambda_\iota$ is written for the line demerits of line ι with the changed `\parfillskip`. A prime is added to Λ and $\bar\Lambda$ if another set of line breaks than the typeset ones is considered.

Assume there is a set of line breaks that create $\mu-1$ lines instead of the μ lines that are typeset with the default `\parfillskip`. The sum of the line demerits for the shorter paragraph must be larger than the total demerits; otherwise it would be typeset:

$$\sum_{\iota=1}^{\mu-1} \Lambda'_\iota > \sum_{\iota=1}^{\mu} \Lambda_\iota.$$

It must be shown that this relation is kept with the changed `\parfillskip`; i.e., a proof for

$$\sum_{\iota=1}^{\mu-1} \bar\Lambda'_\iota > \sum_{\iota=1}^{\mu} \bar\Lambda_\iota$$

is needed. Then no shorter paragraph is considered by TEX if $\mathbf{p} = (h, 0\,\mathrm{pt}, h)$.

With the new setting of `\parfillskip` all lines except the last keep their line demerits as the line breaks are not changed. The line demerits of the last line are either unchanged or they increase because of the larger badness and maybe the addition of `\adjdemerits`. Therefore

$$\sum_{\iota=1}^{\mu-1} \bar\Lambda'_\iota = \sum_{\iota=1}^{\mu-2} \Lambda'_\iota + \bar\Lambda'_{\mu-1} \ge \sum_{\iota=1}^{\mu-2} \Lambda'_\iota + \Lambda'_{\mu-1} = \sum_{\iota=1}^{\mu-1} \Lambda'_\iota.$$

On the other hand, for the line breaks with μ lines the demerits do not change, as the last line is so short that its badness does not increase, as shown above. That is $\bar\Lambda_\mu = \Lambda_\mu$ and

$$\sum_{\iota=1}^{\mu} \bar\Lambda_\iota = \sum_{\iota=1}^{\mu-1} \Lambda_\iota + \bar\Lambda_\mu = \sum_{\iota=1}^{\mu-1} \Lambda_\iota + \Lambda_\mu = \sum_{\iota=1}^{\mu} \Lambda_\iota.$$

Therefore

$$\sum_{\iota=1}^{\mu-1} \bar\Lambda'_\iota \ge \sum_{\iota=1}^{\mu-1} \Lambda'_\iota > \sum_{\iota=1}^{\mu} \Lambda_\iota = \sum_{\iota=1}^{\mu} \bar\Lambda_\iota$$

and TEX does not typeset the $\mu - 1$ lines. QED.

Using the trace data. With specifications of natural width and shrinkability last lines cannot be extended or absorbed unless the specification makes other solutions impossible for TEX, i.e., the badness of the other last lines must become larger than the tolerance. Experiment 24 shows this effect for the text of experiment 6, for which the possible paths are documented in Table 4.

The default setting selects path $\mathtt{2}_0$. To get to path $\mathtt{0}_0$ by a specification for $\mathbf{p}$ with $p^\circ = p^-$ and $p^+ = 0\,\mathrm{pt}$, the short last line of $\mathtt{2}_0$ must receive infinite badness, or in other words p° must be smaller than the used stretchability. Let's name the used stretchability u; then

$$\beta \approx 100\Big(\frac{u}{0.8h + g^+}\Big)^3$$

$$\Longrightarrow \qquad u \approx \sqrt[3]{\frac{\beta}{100}} \cdot 0.8h + \sqrt[3]{\frac{\beta}{100}}\, g^+.$$

With the numbers of Table 4, u is easily computed as $\sqrt[3]{1.64}\cdot 0.8h + 0\,\mathrm{pt} \approx 0.9434\mathrm{h}$ and therefore $p^\circ = p^- = 0.94$ will typeset $\mathtt{0}_0$. (The exact value is 0.94394.)

Experiment (verification): TEX output

• `\parfillskip` ← 0.94`\hsize` minus 0.94`\hsize`:
6. One line or two for this text? That is the question, or? ∎

Note the badness increases to 98, so the line does not end flush right.

To get the path of column $\mathtt{1}_0$ the column $\mathtt{0}_0$ must be eliminated, for example, by making p° a little bit larger than p^-; the constant $0.02h$ was used in section 7, although this is rather large. Instead of badness 164 the last line now has badness 115. For an exact computation the value $g^+ = 1.97206\,\mathrm{pt}$ should be used; multiplied with $\sqrt[3]{1.15}$ it is $\approx 0.0089h$. As $\sqrt[3]{1.15} \cdot 0.8h \approx 0.8381h$ the setting $p^\circ = p^- + 0.02h = (0.8381 + 0.0089)h = 0.8470h$ typesets path $\mathtt{1}_0$; $p^\circ = 0.83722h$ is sufficient.

Experiment (verification): TEX output

• `\parfillskip` ← 0.84`\hsize` minus 0.82`\hsize`:
6. One line or two for this text? That is the question, or? ∎

10 Adding stretchability back again

Experiments 23 and 26 show that spaces in a short line sometimes stretch a lot if natural width and shrinkability are specified for `\parfillskip`. In such a case, can a specification for stretchability help?

With finite dimensions in the specification of `\parfillskip` the glue of the last line of a paragraph might have its natural width by luck but usually it either has to shrink or to stretch. To avoid underfull lines the natural width and the stretchability should reach `\hsize` (see section 6) and to avoid overfull lines the shrinkability should have the same width as the natural width (see section 8). Therefore

a specification like

$\texttt{\parfillskip} \equiv x$ `plus` `\hsize` $- x$ `minus` x

with $0\,\mathrm{pt} < x < \texttt{\hsize} = h$ seems to qualify, as a first attempt. If the length of the last line is named t, then one the following cases

$$\texttt{\parfillskip} \equiv \begin{cases} x \texttt{ plus } \texttt{\hsize} - x, & t < h - x \\ x, & t = h - x \\ x \texttt{ minus } x, & t > h - x \end{cases}$$

are used with dimensions from `\parfillskip` depending on t. So short last lines, i.e., the width is less than $h - x$, use natural width and stretchability and long last lines, i.e., the width is greater than $h - x$, deploy natural width and shrinkability.

With $x = 0.75h$ the width of a last line reaches quite early the point where the badness becomes 0; see Fig. 4. The setting $x = 0.25h$ has the opposite effect. From the previous sections it is known that these settings behave quite differently with respect to adding or removing a short last line.

Experiment 29: Description
Show differences for short last lines with these settings.

TEX output

• `\parfillskip` ← 0.75`\hsize` plus 0.25`\hsize` minus 0.75`\hsize`:

2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?
4′. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

• `\parfillskip` ← 0.25`\hsize` plus 0.75`\hsize` minus 0.25`\hsize`:

2. Has the last line of this paragraph badness 0 and has no interword space to stretch? Do they shrink now?
4′. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

The experiments show that both settings might typeset a paragraph differently from the output produced by the default setting. The smaller natural width prefers long last lines and absorbs a short last line, the larger one produces short last lines. Long but not completely filled lines are typeset nearly identically.

Experiment 30: Description
Show a minor difference for a long last line with these settings.

TEX output

• `\parfillskip` ← 0.75`\hsize` plus 0.25`\hsize` minus 0.75`\hsize`:

5. With enough interword glue as well as short words at the end of the 1st line the 2nd can be extended.

• `\parfillskip` ← 0.25`\hsize` plus 0.75`\hsize` minus 0.25`\hsize`:

5. With enough interword glue as well as short words at the end of the 1st line the 2nd can be extended.

It is hard to see the difference, but the last line of the first case has badness 46, in the second paragraph the value is 6. Of course, there are several experiments which do not show a difference except for the spacing.

Experiment 31: Description
Show that some texts are not changed compared to the default and some change for both settings.

TEX output

• `\parfillskip` ← 0.75`\hsize` plus 0.25`\hsize` minus 0.75`\hsize`:

3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
6. One line or two for this text? That is the question, or?

• `\parfillskip` ← 0.25`\hsize` plus 0.75`\hsize` minus 0.25`\hsize`:

3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
6. One line or two for this text? That is the question, or?

The stretchability must not be exactly `\hsize`$-x = h - x$. If the second pass is accepted for short lines the stretchability can be lowered to $(h-x)/\sqrt[3]{2}$ or to $4(h-x)/5$, if the `\tolerance` is 200; $5/4 < \sqrt[3]{2}$ but quite close, so the division with a smaller number makes the stretchability a little bit larger. In fact, the stretchability can be set to any valid value, for example, to $2h$ or 1 fil to have only decent last lines if the glue has to stretch, as only the condition $h - x > 0\,\mathrm{pt}$ for `\parfillskip`'s natural width x must hold. But such large values for the stretchability do not avoid short last lines, etc.; the only visible change might be some white space at the end of long last lines, for example, in the text of experiment 2.

Of course, the stretchability can also be quite small, it might be smaller than the difference of line width and natural width. A large natural width with a very small stretchability might absorb a short last line as such lines are underfull. A shorter natural width and enough stretchability so that no underfull lines are created has the same property.

Experiment 32: Description
Show that a smaller stretchability avoids short last lines.

TEX output

• `\parfillskip` ← 0.75`\hsize` plus 0.06`\hsize` minus 0.75`\hsize`:

4′. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a

6. One line or two for this text? That is the question, or?
• \parfillskip ← 0.25\hsize plus 0.6\hsize minus 0.25\hsize:
4′. My keyboArd is broken; when I press an 'A' in lowercAse (only) the screen repeAts it four times: a a a a
6. One line or two for this text? That is the question, or? □

The first paragraph shows that the penalization of non-short lines can lead to an overreaction. Extended lines can share the same fate.

Experiment 33: Description
Show that in a specification with a very small stretchability last lines are sometimes extended needlessly.

TEX output
• \parfillskip ← 0.75\hsize plus 0.06\hsize minus 0.75\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
8. One line or two for this text? That's the question, or not?
• \parfillskip ← 0.25\hsize plus 0.6\hsize minus 0.25\hsize:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
8. One line or two for this text? That's the question, or not? □

The natural width of \parfillskip was expressed as a fraction of h but a fixed value works too. Then the results depend on the current \hsize.

Experiment 34: Description
Show that a small value of natural width is sufficient.

TEX output
• \parfillskip ← 10 pt plus \hsize − 10 pt minus 10 pt:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
6. One line or two for this text? That is the question, or?
8. One line or two for this text? That's the question, or not?
• \parfillskip ← 20 pt plus 0.8\hsize minus 20 pt:
3. A short text in 1 line. Or has the paragraph 2 or 3 lines?
4. My keyboArd is broken. When I press the key for the lowercAse A the screen repeAts it severAl times: a a a a
6. One line or two for this text? That is the question, or?
8. One line or two for this text? That's the question, or not? □

Note all paragraphs for the experiments in this section are typeset in the first pass, except 4′ in experiments 29 and 32 and both 8s in experiment 33.

Discussion. The definition of all three dimensions in the glue specification of \parfillskip combines two specifications: natural width & stretchability and natural width & shrinkability. If the natural width equals the shrinkability, and the stretchability together with the natural width never outputs underfull lines, most problems that occur with each individual specification are avoided.

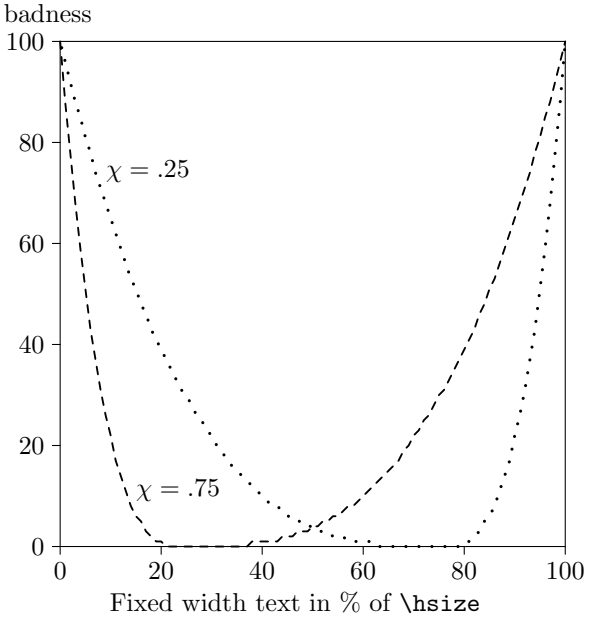


Figure 4: Graphs for $\chi = 0.25$ and $\chi = 0.75$ when \parfillskip ≡ χ\hsize plus $(1-\chi)$\hsize minus χ\hsize

Table 6 lists the experiments for the different specification discussed in this section. Figure 4 has the graphs for $x = 0.25h$ and $x = 0.75h$, i.e., the natural width is once a quarter and once three quarters of \hsize. With a full glue specification a small value for the natural width (and the shrinkability)

Table 6: Observations for complete glue specifications

Specification of \parfillskip ($h \equiv$ \hsize)					
natural width	0pt	.25h	.75h	.25h	20pt
stretch	1fil	.75h	.25h	.6h	.8h
shrink	0pt	.25h	.75h	.25h	20pt
Observation					
1 Short line	1	(1)	29	(1)	(1)
2 Completely filled	2	(22)	(22)	(22)	(22)
3 Glue stretches	–	31	31	33	34
4 2nd pass forced	–	–	–	33	(1)
5 Underfull line	–	–	–	–	–
6 No demerits	–	–	–	–	–
7 Remove short line	–	29	–	32	34
8 Extend last line	–	31	31	33	34
9 Backspaces	–	–	–	–	–
10 Add short line	–	–	29	–	–
11 Glue shrinks early	–	30	30	32	34
12 Overfull line	–	–	–	–	–
13 Stick out right	–	–	–	–	–

Legend: −/ /+: never/don't care/always
n/(n): (implicitly) shown in example number n

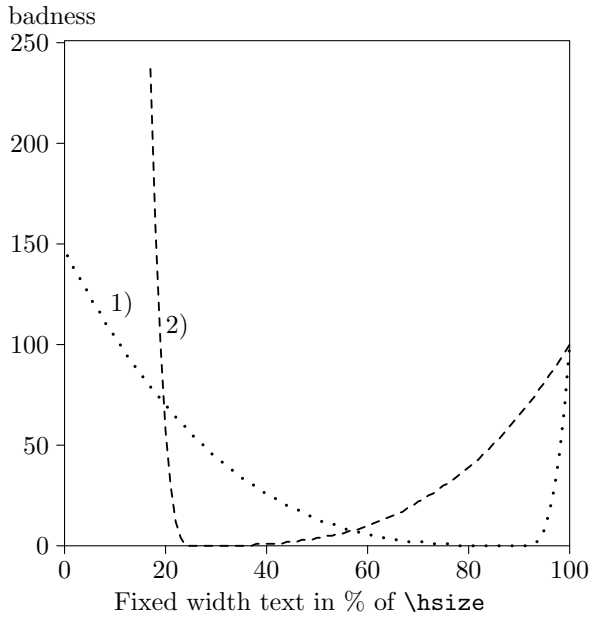


Figure 5: Graphs of two `\parfillskip` specifications 1) `20pt plus 0.8\hsize minus 20pt` and 2) `0.75\hsize plus 0.06\hsize minus 0.75\hsize`

assigns to short lines higher badness values than a large value for the natural width does. The first setting prefers nearly filled lines over short lines, i.e., recommendation (1) is followed more often than recommendation (2). On the other hand the small value needs much more text in the last line before the last line gets decent and the interword glue stretches less than 50%. Figure 5 shows two specifications which might force a second pass; one does not obey the tolerance in all cases.

Summary. Setting all dimensions of the specification to a nonzero value creates a complex scenario which applies two different specifications to short and long lines. The results of the previous sections suggest equality for natural width and shrinkability and to assign so much stretchability that, together with the natural width, it reaches the `\hsize` (in either the first or the second pass). A small value for the natural width works better than a large one although spaces in very short lines stretch a little bit more.

11 A closer look at a few specifications

Let's look at the specifications for `\parfillskip` that are mentioned in the summaries of the previous sections. The experiments in this article are quite short and a setting of `\parfillskip` that makes last lines longer works much better when there are more sets of valid line breaks; paragraphs with plenty of text usually have a greater selection of possible line breaks. Nevertheless, a general specification must be able to handle one- or two-line paragraphs.

I do not list a specification that might throw an error. An author has to decide if a forced second pass based only on the setting of `\parfillskip` is acceptable. Such settings are often successful when the first pass fails to lengthen the last line or to absorb a short last line into the former penultimate line. Values smaller than 0.8`\hsize` for the stretchability and a value smaller than `\hsize` for natural width and shrinkability are possible but then underfull lines and other problems might occur. And it seems questionable to make TEX choose line breaks that output a paragraph that has several thousand demerits more than the optimum instead of giving a hint that the last line is short.

1) `0pt plus 0.8\hsize` (Fig. 1): Instead of 1 fil a finite stretchability is used. Spaces in the last line stretch; in a very short line they stretch more than the specified stretchability as TEX might execute a second pass.

2) `\hsize minus \hsize` (Fig. 3): With this value, spaces in the last line shrink. A new short last line might be produced to avoid a completely filled last line, but such a line is very unlikely.

3) `20pt plus 0.8\hsize minus 20pt` (Fig. 5 with *TUGboat*'s `\hsize`): The specification makes completely filled lines unlikely although not impossible. For short lines it acts similarly to case 1) but for long last lines glue shrinks early and, unless badness 100 is reached, some white space at the end of the line is set.

Comparison to other suggestions. In section 2 some specifications for `\parfillskip` suggested in the literature are listed. How do these compare to the above recommendations? Two of them are not meant for general use in documents but in special cases so a comparison is not quite fair.

The specification `0pt plus 0.7\hsize` creates the same output for the experiments as case 1) except that artificial demerits are applied in experiments 1 and 13 as this specification plays with invalid badness values. Case 2) and the non-general specification `2em plus 1fil` produce the same typeset output except experiments 2′ and 15 generate overfull lines in the latter case.

The glue specification with three dimensions, interpreted as `0.33\hsize plus 0.67\hsize minus 0.33\hsize`, used if `\looseness` is set, behaves similarly to case 3) except that the former prefers additional lines. So experiments 2, 4, and 6 are typeset with one more line. The other complete glue specification, `0.75\hsize plus 0.06\hsize minus`

`0.75\hsize`, also behaves similarly to case 3) with wider white space at the end of lines. Further, experiments 1 and 13 produce artificial demerits because the stretchability is so small; and there is a noticeable difference for text 8 (see experiment 33).

The last specification has the most complex assignment. Informally it is `\hsize` − 1.5`\parindent minus \hsize` − 1.5`\parindent` − 1`em`. This specification produces an overfull line for experiment 2′ and artificial demerits in experiments 1, 2, and 13. In experiment 4′ it absorbs the short line (typeset as the first version shown in experiment 32) otherwise in experiments 4, 6, and 8 last lines are lengthened.

How much does a single space stretch? Several times in the above analysis the stretching of single spaces in a short last line was mentioned. With experiment 13 this can be visualized; of course with the extra stretchability after a period. The specification `0pt plus 5.17\hsize` used in Fig. 2 is included in the comparison.

Experiment 35: Description
Show how much a single space stretches in a short line.

TEX output

• `\parfillskip` ← `\hsize` minus `\hsize`:
13. a
• `\parfillskip` ← 2 em plus 1fil:
13. a
• `\parfillskip` ← 0 pt plus 5.17`\hsize`:
13. a
• `\parfillskip` ← 0.33`\hsize` plus 0.67`\hsize` minus 0.33`\hsize`:
13. a
• `\parfillskip` ← 20 pt plus 0.8`\hsize` minus 20 pt:
13. a
• `\parfillskip` ← 0 pt plus 0.8`\hsize`:
13. a
• `\parfillskip` ← 0 pt plus 0.7`\hsize`:
13. a
• `\parfillskip` ← 0.75`\hsize` plus 0.06`\hsize` minus 0.75`\hsize`:
13. a
• `\parfillskip` ← 1`\hsize` − 1.5`\parindent` minus 1`\hsize` − 1.5`\parindent` − 1 em:
13. a

The last four specifications have artificial demerits, and the last two produce underfull lines.

12 Summary

A change of `\parfillskip`'s specification does not spare an author the need to proofread to check if the last lines of paragraphs are shorter than the indentation or fill the line width.

There is another point on the checklist if finite dimensions are used for `\parfillskip` and the author has not always entered `\hfil\ $$` to start display math mode: If a short last line that contains one space precedes a display then too much vertical space might be used by TEX, as it is forced to apply `\abovedisplayskip` and `\belowdisplayskip`.

Paragraphs with indentation. With the default settings of plain TEX the specification of the glue `\parfillskip` has one typographical problem: Last lines in paragraphs can be shorter than `\parindent`, which has the value 20 pt by default. A reader spots the start of a paragraph easily because of the wide indentation. Thus, completely filled last lines generate no problem for the readability.

If an author wants to avoid last lines of a few characters the simplest way is to use a tie to keep the last short word connected to the text or an hbox to avoid the hyphenation of a word that can leave a fragment of three letters on the last line. If TEX cannot find an acceptable set of line breaks with such input an overfull line message informs the author to rewrite the text or to change TEX's parameters. Note that the ties and the hboxes can make TEX execute a second pass although the default setting would need only a first pass. Thus a longer last line is traded in such cases for higher badness values in other lines and/or hyphens.

To prevent TEX from typesetting short last lines without hints in the input, `\parfillskip`'s specification has to change. For *TUGboat*, the best choice from section 11 seems to be `20pt plus 0.8\hsize minus 20pt`. For other line widths the constant needs to be adjusted.

Paragraphs without indentation. A text that does not signal the start of a paragraph by white space, i.e., through indentation or an empty line between paragraphs, or through other methods, for example, p. 40 in [3] names ornaments and outdented paragraphs, relies on white space at the end of the last line to indicate the end of the paragraph. In this scenario a completely filled last line creates a problem. With plain TEX such lines are possible; see experiment 2.

To avoid a completely filled last line the natural width must make a nonzero contribution. The simple addition of natural width to the default setting of `\parfillskip`, for example, `10pt plus 1fil`, seems to be a valid solution although now overfull lines are possible. A setting of `\parfillskip` in which the natural width equals the shrinkability makes a completely filled last line not quite impossible, but

very unlikely, and without generating overfull lines, for example, `20pt plus 0.8\hsize minus 20pt`.

Which setting was used for this article? I decided to make an experiment and to use the macros shown in Fig. 2, that is, at the end of a paragraph `\parfillskip` is set to `0pt plus 5.17\hsize`. It is the first time that I applied these macros. As the *TUGboat* format uses a large indent I was only interested in avoiding short last lines and accepted completely filled ones. After the text became reasonably stable I activated the macro that checks the length of the last lines except in the experiments (verbatim parts and output), figures, tables, and the list of references. Initially, the macro gave an error six times: In two cases I changed the text, in one case the last word was placed in an hbox, and three last lines were accepted as they are false hits, i.e., the last line is tight or long enough in front of a heading. Later revisions changed some of these lines again.

The text contains several formulas and I enter display math to show them. The macros of Fig. 2 use finite stretchability for `\parfillskip` only for the end of a paragraph; otherwise the default setting is applied. So there is no problem with the vertical space around displays.

References

[1] Paul W. Abrahams, Kathryn A. Hargreaves, Karl Berry, *File* `macros.tex` *of package* `impatient`. `ctan.org/tex-archive/info/impatient`

[2] Wolfgang Appelt: *TEX für Fortgeschrittene*, Bonn, Germany: Addison-Wesley, 1988.

[3] Robert Bringhurst, *The Elements of Typographic Style*, 4th edition, version 4.2, Seattle, Washington: Hartley & Marks, 2016.

[4] Anne Brüggemann-Klein, "Re: Obtaining a funny paragraph shape in TeX", *TEXhax Digest* **89**:42 (1989), 28 April 1989. `ctan.org/tex-archive/info/digests/texhax/89/texhax.42.gz`

[5] David Carlisle (based on a `comp.text.tex` article of Peter Schmitt), "Russian Paragraph Shapes", *Baskerville* **6**:1 (1996), 13–15. `uk.tug.org/wp-installed-content/uploads/2008/12/61.pdf`

[6] *The Chicago Manual of Style*, 15th edition, Chicago, Illinois: University of Chicago Press, 2003.

[7] Jean-luc Doumont, *Trees, maps, and theorems*, Belgium, 2009. `treesmapsandtheorems.com`

[8] Jean-luc Doumont, "Quantum space: Designing pages on grids", *TUGboat* **31**:2 (2010), 248. `principiae.be/pdfs/TUG-X-004-slideshow.pdf` `zeeba.tv/quantum-spaces-designing-pages-on-grids`

[9] Friedrich Forssman & Ralf de Jong, *Detailtypographie*, 3rd edition, Mainz, Germany: Verlag Hermann Schmidt, 2004.

[10] Donald E. Knuth and Michael F. Plass, "Breaking paragraphs into lines", *Software — Practice and Experience* **11** (1981), 1119–1184; reprinted with an addendum as Chapter 3 in [14], 67–155.

[11] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[12] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[13] Donald E. Knuth, *Literate Programming*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 27, 1992.

[14] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[15] Donald Knuth, "TEX's infinite glue is projective", *TUGboat* **28**:1 (2007), 4. `tug.org/TUGboat/tb28-1/tb88knut.pdf`

[16] Frank Mittelbach, "E-TEX: Guidelines for Future TEX Extensions", *TUGboat* **11**:3 (1990), 337–345. `tug.org/TUGboat/tb11-3/tb29mitt.pdf`

[17] Walter Schmidt, "Absätze — einmal anders", *Die TEXnische Komödie* **9**:2 (1997), 19–22. `dante.de/DTK/Ausgaben/komoedie19972.pdf`

[18] Philip Taylor, "Book Design for TEX Users — Part 2: Practice", *TUGboat* **20**:4 (1999), 378–389. `tug.org/TUGboat/tb20-4/tb65tay2.pdf`

[19] Udo Wermuth, "Tracing paragraphs", *TUGboat* **37**:3 (2016), 358–373; Errata in *TUGboat* **38**:3 (2017), 414. `tug.org/TUGboat/tb37-3/tb117wermuth.pdf`

[20] Udo Wermuth, "A note on `\linepenalty`", *TUGboat* **38**:3 (2017), 400–414; Errata in *TUGboat* **39**:1 (2018), 87. `tug.org/TUGboat/tb38-3/tb120wermuth.pdf`

[21] Udo Wermuth, "TEX's 'additional demerits' parameters", *TUGboat* **39**:1 (2018), 81–87. `tug.org/TUGboat/tb39-1/tb121wermuth-adem.pdf`

[22] Peter Wilson, "Glisterings", *TUGboat* **28**:2 (2007), 229–232. `tug.org/TUGboat/tb28-2/tb89glister.pdf`

[23] Peter Wilson, "Glisterings", *TUGboat* **29**:2 (2008), 324–325. `tug.org/TUGboat/tb29-2/tb92glister.pdf`

[24] Peter Wilson, "Glisterings", *TUGboat* **38**:3 (2017), 338–341. `tug.org/TUGboat/tb38-3/tb120glister.pdf`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

The Treasure Chest

This is a selection of the new packages posted to CTAN (`ctan.org`) from August–October 2018, with descriptions based on the announcements and edited for extreme brevity.

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 wide interest or otherwise notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TEX community. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`tugboat (at) tug dot org`

fonts

`firamath` in `fonts`
Fira sans serif with Unicode math support.

`firamath-otf` in `fonts`
Fira in OpenType.

`libertinus` in `fonts`
Wrapper for choosing `libertinus-type1` or `libertinus-otf`.

`libertinus-otf` in `fonts`
Using Libertinus with (XE,Lua)LaTeX.

`libertinus-type1` in `fonts`
Using Libertinus with (pdf)LaTeX.

graphics

`coloremoji` in `graphics`
Directly include color emojis in LaTeX.

`jigsaw` in `graphics/pgf/contrib`
Draw jigsaw pieces.

`metapost-colorbrewer` in `g/metapost/contrib/macros`
`colorbrewer2.org` colors for MetaPost.

`pgf-cmykshadings` in `graphics/pgf/contrib`
Support for CMYK and grayscale shadings.

`pst-feyn` in `graphics/pstricks/contrib`
Draw graphical elements for Feynman diagrams.

`pst-lsystem` in `graphics/pstricks/contrib`
Create images based on a Lindenmayer system.

`pst-marble` in `graphics/pstricks/contrib`
Draw marble-like patterns.

`quantikz` in `graphics/pgf/contrib`
Draw quantum circuit diagrams.

`rank-2-roots` in `graphics/pgf/contrib`
Math drawings arising in representation theory.

info

`ptex-manual` in `info`
First release of manual for Japanese pTeX.

language/japanese

`bxwareki` in `language/japanese/BX`
Convert dates from Gregorian to Japanese calendar.

`plautopatch` in `language/japanese`
Automated patches for (u)pLaTeX.

macros/generic

*`tex-locale` in `macros/generic`
Localization support for (La)TeX.

macros/latex/contrib

`aeb-minitoc` in `macros/latex/contrib`
Create mini-TOCs.

`chs-physics-report` in `macros/latex/contrib`
Lab reports for Carmel (Indiana) High School.

`ditaa` in `macros/latex/contrib`
Embed ASCII art and convert to images.

`grabbox` in `macros/latex/contrib`
Read argument into a box and execute later.

`hybrid-latex` in `macros/latex/contrib`
Allow active Python code in LaTeX.

`kalendarium` in `macros/latex/contrib`
Dates according to classical Latin calendar.

`kvmap` in `macros/latex/contrib`
Create Karnaugh maps with LaTeX.

`returntogrid` in `macros/latex/contrib`
Semi-automatic grid typesetting.

`srdp-mathematik` in `macros/latex/contrib`
Typeset Austrian SRDP in mathematics.

`utexasthesis` in `macros/latex/contrib`
UT Austin graduate thesis style.

`widows-and-orphans` in `macros/latex/contrib`
Identify (typographic) widows and orphans.

macros/xetex/latex

`facture-belge-simple-sans-tva` in `m/xetex/latex`
Simple Belgian invoice without VAT.

`thesis-qom` in `macros/xetex/latex`
Thesis style for University of Qom, Iran.

support

`cluttex` in `support`
Automate LaTeX document processing, with clean directories.

*`colorprofiles` in `support`
Collection of free (libre) ICC profiles.

`ctanbib` in `support`
Make `.bib` entry for CTAN package.

`tlcockpit` in `support`
GUI frontend to TeX Live Manager (`tlmgr`).

Abstracts

***Die TEXnische Komödie* 4/2018**

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). (Non-technical items are omitted.)

Markus Kohm, KOMA-Script für Paketautoren am Beispiel `tocbasic` [KOMA-Script for package authors, with example `tocbasic`]; pp. 50–57

KOMA-Script has never been just a tool for users but has always been a tool for developers as well. Starting with KOMA-Script 3, many integral parts of KOMA-Script were "outsourced" into separate packages. Using `tocbasic` and `nomencl` as examples, this article shows how other developers may profit from this development.

Markus Kohm, `fancyhdr` und `scrlayer` in trauter Zweisamkeit [`fancyhdr` and `scrlayer` united]; pp. 58–67

There are several packages on CTAN to modify the page style of a document, with `fancyhdr` as one of the most favorite. Another, more basic package is the KOMA-Script package `scrlayer`. With its layer model it does much more than what is usually delivered by a page style modification package, so users asked for a way to use the layers of `scrlayer` together with `fancyhdr`. With the experimental package `scrlayer-fancyhdr` this is possible now.

Elke Schubert, Beispiele für Einsatzmöglichkeiten des Paketes `tocbasic` [Examples of the use of `tocbasic`]; pp. 66–73

This article shows a few examples for the use of the `tocbasic` package with standard document classes.

[Received from Herbert Voß.]

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.

Aicart Martinez, Mercè
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LATEX or TEX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

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.

Hendrickson, Amy
57 Longwood Ave. #8
Brookline, MA 02446
+1 617-738-8029
Email: amyh (at) texnology.com
Web: http://texnology.com
LaTeX Macro Writing: Packages for print and e-publishing; Sophisticated documentation for users. Book and journal packages distributed on-line to thousands of authors.

More than 30 years' experience, for major publishing companies, scientific organizations, leading universities, and international clients.

Graphic design; Software documentation; LaTeX used for Data Visualization, and automated report generation; e-publishing, design and implementation; Innovation to match your needs and ideas.

LaTeX training, customized to your needs, on-site — have taught classes widely in the US, and in the Netherlands and Sweden.

See the TeXnology website for examples. Call or send email: I'll be glad to discuss your project with you.

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 packages to meet your needs. Call or email to discuss your project or visit my website for further details.

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, C, C++ 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.

TEXtnik
Spain
Email: textnik.typesetting (at) gmail.com
Do you need personalised LaTeX class or package creation? Maybe help to finalise your current typesetting project? Any problems compiling your current files or converting from other formats to LaTeX? We offer +15 years of experience as advanced LaTeX user and programmer. Our experience with other programming languages (scripting, Python and others) allows building systems for automatic typesetting, integration with databases, ... We can manage scientific projects (Physics, Mathematics, ...) in languages such as Spanish, English, German and Basque.

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.

Webley, Jonathan
Flat 11, 10 Mavisbank Gardens
Glasgow, G1 1HG, UK
07914344479
Email: jonathan.webley (at) gmail.com
I'm a proofreader, copy-editor, and LaTeX typesetter. I specialize in math, physics, and IT. However, I'm comfortable with most other science, engineering and technical material and I'm willing to undertake most LaTeX work. I'm good with equations and tricky tables, and converting a Word document to LaTeX. I've done hundreds of papers for journals over the years. Samples of work can be supplied on request.

2019 TEX Users Group election

Karl Berry
for the Elections Committee

The positions of TUG President and five other members of the Board of Directors will be open as of the 2019 Annual Meeting, which we expect to be held in August 2019 in Palo Alto, California, USA.

The TUG Directors with terms expiring in 2019: Barbara Beeton, Susan DeMeritt, Michael Doob, Cheryl Ponchin, Norbert Preining.

Continuing Directors, with terms ending in 2021: Karl Berry, Johannes Braams, Kaja Christiansen, Taco Hoekwater, Klaus Höppner, Frank Mittelbach, Ross Moore, Arthur Reutenauer, Will Robertson, Herbert Voß.

The election to choose the new President and Directors will be held in early Spring of 2019. Nominations for these openings are now invited. The term of President is two years, and the term of TUG Director is four years. A nomination form is on this page; forms may also be obtained from the TUG office or via `tug.org/election`.

The TUG Bylaws provide that "Any member may be nominated for election to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by ... ballot of the entire membership, carried out in accordance with those same Procedures."

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office; the petition and all signatures must be received by the deadline stated below. Also, a candidate's membership dues for 2019 must be paid before the nomination deadline.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt of complete nomination forms and ballot information is

07:00 a.m. PST, 1 March 2019

at the TUG office in Portland, Oregon, USA. No exceptions will be made. Forms may be submitted by fax, or scanned and submitted by email to `office@tug.org`; receipt will be confirmed by email.

Information for obtaining ballot forms from the TUG website will be distributed by email to all members within 21 days after the close of nominations. It will be possible to vote electronically. Members preferring to receive a paper ballot may make arrangements by notifying the TUG office. Marked ballots must be received by the date noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by early May, and will be announced in a future issue of *TUGboat* and through various TEX-related electronic media.

2019 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2019 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

☐ **TUG President**

☐ **TUG Director**

for a term beginning with the 2019 Annual Meeting.

1. _____ (please print)

_____ (signature) _____ (date)

2. _____ (please print)

_____ (signature) _____ (date)

Return this nomination form to the TUG office via postal mail, fax, or scanned and sent by email. Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received at the TUG office in Portland, Oregon, USA, no later than

07:00 a.m. PST, 1 March 2019.

It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will late or incomplete applications be accepted. In case of any questions about a candidacy, the full TUG Board will be consulted.

Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same physical form.

- ☐ nomination form
- ☐ photograph
- ☐ biography/personal statement

TEX Users Group
Nominations for 2019 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.
(email: `office@tug.org`; fax: +1 815 301-3568)

Calendar

2018

Nov 10 TEXConf 2018, Sapporo, Japan
texconf2018.peatix.com

2019

Jan 4 – 5 College Book Art Association Biennial Meeting, "The Photographic Artists' Book", The University of Arizona, Tucson, Arizona.
www.collegebookart.org

Feb 3 – 6 CODEX VII 2019 Book Fair and Symposium, Richmond, California.
www.codexfoundation.org

Mar 1 **TUG election:** nominations due, 07:00 a.m. PST. tug.org/election

Mar 2 – 4 Typography Day 2019, "Experimental Typography". IDC School of Design, Indian Institute of Technology Bombay, Mumbai, India. www.typoday.in

Mar 1 *TUGboat* **40**:1, submission deadline.

Apr 26 **TUG election:** ballots due.
tug.org/election

May 15 **TUG 2019** deadline for abstracts for presentation proposals.
tug.org/tug2019

Jun 1 **TUG 2019** early bird registration deadline. tug.org/tug2019

Jun 9 **TUG 2019** hotel reservation discount deadline. tug.org/tug2019

Jun 7 – 14 Mills College Summer Institute for Book and Print Technologies, Oakland, California.
millsbookartsummer.org

Jun 19 – 21 The 7th International Conference on Typography and Visual Communication (ICTVC), "Challenging Design Paths", Patras, Greece. www.ictvc.org

Jul 3 – 5 Seventeenth International Conference on New Directions in the Humanities (formerly Books, Publishing, and Libraries), "The World 4.0: Convergences of Knowledges and Machines", University of Granada, Granada, Spain.
thehumanities.com/2019-conference

Jul 4 – 5 International Society for the History and Theory of Intellectual Property (ISHTIP), 11th Annual Workshop, "Intellectual Property and the Visual". Sydney, Australia.
www.ishtip.org/?p=995

Jul 9 – 12 Digital Humanities 2019, Alliance of Digital Humanities Organizations, Utrecht, The Netherlands.
adho.org/conference

Jul 15 – 19 SHARP 2019, "Indigineity, Nationhood, and Migrations of the Book". Society for the History of Authorship, Reading & Publishing. University of Massachusetts, Amherst, Massachusetts. www.sharp2019.com

Jul 28 – Aug 1 SIGGRAPH 2019, "Thrive together", Los Angeles, California.
s2019.siggraph.org

Jul 29 – Aug 2 Balisage: The Markup Conference, Rockville, Maryland. www.balisage.net

---

TUG 2019 Palo Alto, California.

Aug 9 – 11 The 40th annual meeting of the TEX Users Group.
tug.org/tug2019

Sep 23 – 26 19th ACM Symposium on Document Engineering, Berlin, Germany.
www.documentengineering.org/doceng2019

Status as of 1 November 2018

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 39 (2018), No. 3