TUG 2023 program

Friday, July 14	08:00	*registration*	
	08:30	Boris Veytsman, TEX Users Group	*Welcome*
	08:45	Ulrike & Gert Fischer, Carla Maggi, Paulo Cereda, samcarter	*Behind the scenes of the Great TikZlings Christmas Extravaganza*
	09:15	Oliver Kopp, JabRef e.V.	*JabRef as BIBTEX-based literature management software*
	09:45	Jan Šustek	*On generating documented source code by blocks in TEX*
	10:15	Barbara Beeton, *TUGboat*	*What every LATEX newbie should know*
	10:45	*break & registration*	
	11:15	Martin Ruckert, Munich University of Applied Sciences	*News from the HINT project*
	11:45	Dennis Müller, FAU Erlangen-Nürnberg	*An HTML/CSS schema for TEX whatsits*
	12:15	Patrick Gundlach, speedata GmbH	*News from boxes and glue: How do the TEX algorithms help in developing a new typesetting engine?*
	12:45	*lunch*	
	14:00	Joseph Wright, samcarter	*Beamer news*
	14:15	samcarter	*The* `tcolorbox` *inner beamer theme*
	14:30	Boris Veytsman, Chan Zuckerberg Initiative, George Mason Univ., TUG	*The update of the* `nostarch` *class*
	15:00	Ben Davies, Overleaf	*Bumpy road towards a good LATEX visual editor*
	15:30	Didier Verna, EPITA Research Lab	*Interactive and real-time typesetting for demonstration and experimentation*
	16:00	*break*	
	16:30	Eberhard W. Lisse, Omadhina Internet Services Ltd	*Introduction to Typst*
	17:00	Jakub Máca, Petr Sojka, Ondřej Sojka	*Universal syllabic pattern generation*
Saturday, July 15	08:30	Henning Hraban Ramm	*Architectural guides for Bonn — book production with ConTEXt*
	09:00	Thomas Schmitz, Bonn University	*Producing different forms of output from XML via ConTEXt*
	09:30	Vít Novotný	*Markdown 3: What's new, what's next?*
	10:00	Rishi T, Apu V, Hàn Thế Thành, Jan Vaněk, STM Document Engineering	*Primo — The new sustainable solution for publishing*
	10:30	*break*	
	11:00	Ross Moore, Macquarie University	*Tagged PDF, derived HTML and aspects of accessibility*
	11:30	Ulrike Fischer, LATEX Project	*Automated tagging of LATEX documents — what is possible today?*
	12:00	Joseph Wright, LATEX Project	*Supporting backends in expl3*
	12:30	Frank Mittelbach, LATEX Project	*The LATEX Companion, 3rd edition — Anecdotes and lessons learned*
	13:00	*lunch*	
	14:00	Jim Hefferon, St Michael's College	*Using Asymptote like MetaPost*
	14:30	Linus Romer	*Curvature combs and harmonized paths in MetaPost*
	15:00	Rajeesh KV	*Metafont, MetaPost and a complex-script typeface*
	15:30	Victor Sannier, GUTenberg Association	*A METAFONT for rustic capitals*
	16:00	*break*	
	16:30	Ulrik Vieth	*An updated survey of OpenType math fonts*
	17:15	Mikael Sundqvist, Lund University	*Extending OpenType math, making choices*
	17:45	Tom Hejda, Overleaf	*TEX Live and Overleaf revisited*
Sunday, July 16	08:30	Island of TEX	*Living in containers — on TEX Live in a docker setting*
	09:00	Joseph Wright	*Further adventures in Unicode-land: Refining case changing*
	09:30	Oliver Kopp	*The LATEX template generator: How micro-templates reduce template maintenance effort*
	10:00	Island of TEX	*The Island of TEX 2023 — sailing the smooth seas of ideas*
	10:30	*break*	
	11:00	Frank Mittelbach	*38 years with LATEX — A personal picture story*
	12:00 am	Boris Veytsman	*Closing*
	≈ 12:15 pm	*lunch*	

Guided walks

Thursday, July 13	15:00 h	Lobby Hotel Leoninum
Monday, July 17	9:30 h	Lobby Hotel Leoninum

About three hours through the historical centre of Bonn. We'll never be far from the hotel.

Banquet

Date	Saturday, July 15
Start	18:45 h Lobby Hotel Leoninum
	station *Hauptbahnhof*, tram 16 or 66 to *Heussallee/Museumsmeile*
Return	around 22:30 h by tram or walk

The banquet will take place in the two banquet rooms of *Konrad's* restaurant (`www.konrads-bonn.de`) on the 17th floor of the Marriott Hotel, Platz der Vereinten Nationen 4, in the former government district south of the city centre. It's situated near the river bank and offers a splendid view of the city in the north, the Venusberg in the west and the Seven Hills south and east.

When you signed up for the banquet you paid for a three course menu plus "amuse bouche". Water will be included. The other drinks will be on the individual participants. If you didn't let us know your choice for the main course, we put you down for "meat".

The banquet will start on the terrace of the restaurant with an apéritif for those who want one towards 19:30 h. Dinner will begin around 8 and should last about two and a half hours.

Travel arrangements: We'll have group tickets for the tram (Lines 16 or 66). We meet at the lobby of the Leoninum at 18:45 h (sharp please) and walk the short distance to the stop *Hauptbahnhof* (underground entrance Thomas-Mann-Straße/Noeggerathstraße). If it's more convenient for you, you might join us there. Just in case you miss us: The trains (Lines 16 or 66) run very frequently (directions Ramersdorf, Bad Honnef or Bad Godesberg). The destination stop is *Heussallee/Museumsmeile*. From there it's a five minute walk to the restaurant.

For the return journey there will be an option to taking the tram: if some of you feel like it, they might spend an hour of a (hopefully) pleasant summer night by walking back along the river. You'll decide then.

Excursion to the *Seven Hills* (Drachenfels and Königswinter)

Date	Sunday, July 16
Start	13:30 h Lobby Hotel Leoninum
	13:50 h station *Hauptbahnhof*, tram 66 to *Königswinter/Bad Honnef*
Return	17.20 h by boat from landing stage of *Bonner Personen Schiffahrt*
Arrival	around 18.15 h

Start

We meet in the lobby of Hotel Leoninum at 13:30 h sharp to walk less than 10 minutes to the underground tram stop *Hauptbahnhof* (Entrance: Thomas-Mann-Straße. Those staying in other hotels may come there directly.).

Departure of line 66 tram (direction Königswinter/Bad Honnef): 13:50 h, track U4a.

Königswinter

Arrival at stop *Königswinter Fähre*: 14:16 h. It is located on the riverside promenade of Königswinter — once a hotspot of weekend tourism. Today things are much quieter, but depending on the weather occasional queuing can't be ruled out. The stop is next to the landing stage of the boat which will take us back to Bonn later in the afternoon.

Cog railway

From here it's a leisurely ten-minute walk to the valley station of the historic cog railway that leads up to the Drachenfels. If the tram arrives on time we might make the one leaving at 14:30 h, but there is no need to hurry. The next ride is at 14:45 h. And even the

one at 15:00 h will leave you plenty of time to explore the Drachenfels.

Since everybody will have their personal (return) ticket there is no need for the group to stay together and you are invited to look around at your own discretion — provided you are back at the riverside at 17:20 h at the latest. The boat won't wait!

Drachenfels

The cog railway will take eight minutes to the summit with a short stop at its middle station. I suggest that you go straight to the top. Here you'll find a stunning viewing platform with, weather permitting, a view as far as Cologne Cathedral. There is a self-service cafeteria (for more attractive options see below). The short but fairly steep climb to the castle ruins starts behind the restaurant. If you find this too strenuous, don't worry. The view from up there isn't any better than the one from the terrace.

What you might not want to miss is the *dragon machine* (also behind the cafeteria). If you feed it a Euro, a little puppet dragon will appear and give you his version of not having been slain by Siegfried. Unfortunately he only speaks German. The machine is about a hundred years old and a quaint reminder of a world without virtual reality and AI. Smaug would be appalled!

Since there are interesting things to be discovered on the way back I suggest that you don't spend too much time on the summit but explore the surroundings of the middle station of the cog railway. You can reach it within three or four minutes with the railway or you can take the footpath through the woods which starts to the left of the cafeteria. It's a pleasant if somewhat steep walk of about 10 to 15 minutes.

At the middle station there are three possibilities. You probably won't have time to go for all of them.

A) is *Schloss Drachenburg* — a 19th century millionaire's dream of the Middle Ages. All fake but quite impressive. It has been restored fairly recently and is full of murals and furniture of a less than modern taste. The outer castle has a museum on the history of the Seven Hills as Germany's first nature reserve. And of course there is yet another view from the gardens. The entrance fee is 9 EUR.

B) is the *Dragon World* a five minute walk below the middle station. One of the mainstays of a long gone age of Drachenfels tourism it consists of three elements. No. 1 is the one you want to see. It's the *Nibelungenhalle*. A kind of temple, it was inaugurated in 1913 commemorating Richard Wagner's 100th birthday. It has murals depicting scenes from the maestro's most famous work "The Ring of the Nibelungs". Together with the somewhat oppressive architecture it uncannily foreshadows the aesthetic which would become state of the art 20 years later under Hitler. Not nice but highly instructive. The (mini) tour then leads to no. 2, the *Dragon's Lair*, which sports a 13 m long stone dragon — created in 1933 on the occasion of the 50th anniversary of Wagner's death. A place of horror in my childhood, it is a most sorry sight today. The dragon's head has fallen off and apparently the lady who owns the place hasn't got the money to stick it on again. Item no. 3 is a tiny reptile zoo. The kids love it, but you might just as well skip it. Entrance fee: 8 EUR.

C) Half way between the middle station and the Dragon World there is a nice shady beer garden and opposite (a little hidden) a small inn (*Felder's*) with a sunny terrace overlooking the river. Just in case you need a rest.

To return to Königswinter it's probably more comfortable to walk back to the middle station and take the train (every fifteen minutes). It's also possible to walk, but the bottom part of the path is not nearly as nice as its upper half. It will get you to the valley station of the cog railway within a quarter of an hour. From there it's 10 more minutes to the riverbank where it all started. You know the way.

Boat

Our boat will leave from the landing stage of *Bonner Personen Schiffahrt* (bps) at 17:20 h. It is usually a few minutes late, but don't bank on it! If you reach the river with time to spare: there are quite a few cafés and ice cream parlours along the promenade. One of them (*Alte Liebe*) is on a boat next to the landing from which we'll be going back.

The trip back (you'll get your ticket from me when you are boarding) with two short intermediate stops will take about 40 minutes and end below *Alter Zoll* in the middle of Bonn.

Help

If you should get lost or experience any other catastrophe:
just give me a ring (0049 1522 5119080) — and to put it in the words of Albus Dumbledore: "Help shall always be given to those who ask for it." Simple questions are allowed too.

What every LaTeX newbie should know

Barbara Beeton

LaTeX has a reputation for producing excellent results, but at the cost of a steep learning curve. That's true, but by understanding a few basic principles, and learning how to avoid some techniques that may seem obvious but often lead one into the weeds, it's possible to avoid some of that pain.

Among the concepts to be covered are these:

- Why is \\ not a good way to end paragraphs?
- Why use `\newcommand` rather than `\def`?
- Why do some spaces in your input cause problems?
- How to use style changes effectively, and limit them to exactly where you want them.

Very few packages will be discussed, but the concepts covered should be compatible with whatever packages you choose to employ. regardless of your field of interest.

This talk is based on years of looking at good and bad document input and output, answering questions from problem-plagued authors, and trying to write documentation that can be understood on first reading.

Automated tagging of LaTeX documents — what is possible today?

Ulrike Fischer

With the summer 2023 release of the LaTeX format it is now possible to create tagged PDF in an automated way from many "Lamport documents": documents using the commands described in the LaTeX manual from Leslie Lamport.

In this talk I will show what is possible and what still needs manual intervention. I will also describe some of the challenges we faced on the technical side and when designing the mapping between LaTeX structures and the set of PDF tags.

Behind the scenes of the Great TikZlings Christmas Extravaganza

Ulrike & Gert Fischer, Carla Maggi, Paulo Cereda, samcarter

The Great TikZlings Christmas Extravaganza is a yearly video series that utilises LaTeX to produce animated films. They usually consist of several short video sequences which feature various characters from the TikZlings ecosystem accompanied by music. An overview of previous videos can be found at `https://github.com/TikZlings`.

In this talk, we will offer a look behind the scenes of the Extravaganza and explain the process of how we turn rough sketches of scenes first into PDFs with LaTeX and subsequently convert those into videos and combine them with music.

News from boxes and glue: How do the TeX algorithms help in developing a new typesetting engine?

Patrick Gundlach

In this presentation I will talk about the experience of the last two years with boxes and glue. The library has not yet reached its final state, but a lot has already been typeset with it. I will show what kind of experiences I have made with the TeX algorithms, which data structures are suitable for text typesetting and how PDF specialties like interaction and accessibility can be integrated.

About boxes and glue: boxes and glue is a library written in the Go programming language that includes many of TeX's algorithms, such as the optimum fit paragraph breaking algorithm, the hyphenation algorithm, and the basic structure with nodes and node lists to assemble boxes. It was originally written as a replacement for LuaTeX to create documents with the speedata Publisher.

Using Asymptote like MetaPost

Jim Hefferon

Asymptote is a descriptive vector graphics language for technical drawing that fits very well with TeX and friends. One appealing thing is that it is in part based on algorithms from Metafont and MetaPost, but it extends those to three dimensions. I'll cover a number of workflow aspects that a beginner to this system who is coming from MetaPost might like to use, notably using a single source file to output many related graphics.

TeX Live and Overleaf revisited

Tom Hejda

Overleaf makes an annual deployment of TeX Live, and we are wondering whether there is an opportunity for both Overleaf and the TeX Live maintainers and LaTeX developers to benefit from our deployments and related testings. We already gave a presentation on this in 2020, and would like to follow up with more recent changes. An open discussion on the topic will follow a brief presentation.

Living in containers — on TeX Live in a docker setting

Island of TEX

Over the course of the last year(s), the Island of TeX has received quite some interest in its Docker containers. This talk gives a brief overview about our container infrastructure for TeX Live and ConTeXt, including some examples on using our containers in production environments. Last but not least, we will elaborate on some interesting (mostly still open) problems connected to containerizing TeX Live.

The Island of TeX 2023 — sailing the smooth seas of ideas

Island of TEX

The Island of TeX always valued community over development pace. This year, we are proud that we could convince our inner sloths to produce a long-awaited new `albatross` release and a new website for our community. On the technical side, we improved our build infrastructure and started welcoming TeX packages. But in the end, this year was primarily about collecting ideas so stay tuned for our talk and call for action.

JabRef as BibTeX-based literature management software

Oliver Kopp

JabRef is a literature management software completely based on the BibTeX format. This talk provides an overview of JabRef by first introducing the basic concept of JabRef. After that, highlights of JabRef will be demonstrated: Integrated web search, grouping of entries, import and export of other formats, and the quality assurance of entries. The integration of PDFs will be demonstrated: Both the linking of PDFs and the integration of BibTeX data into PDFs using XMP metadata.

The LaTeX template generator: How micro-templates reduce template maintenance effort

Oliver Kopp

Scientific findings are published by different publishers. These provide different templates. These differ in the documentation and packages provided. For example, `microtype` or `hyperref` are mostly not included or not configured properly. Furthermore, there is a demand for minimal examples in the body of the paper. For instance, how to typeset a listing with line numbers and hyperlink to that line number. These minimal examples should appear in any paper template. If the minimal example is updated, how can various paper templates be updated automatically? The "LaTeX Template Generator" is one answer to this question. It uses "micro-templates" to create full-fledged paper templates containing the same configurations for popular packages. Thus, it reduces the maintenance effort of LaTeX templates.

Introduction to Typst

Eberhard W. Lisse

typst is a new markup-based typesetting system that is designed to be as powerful as LaTeX while being much easier to learn and use. It flows from a Master's thesis at the Technical Unversity Berlin, is written in Rust, and has a domain-specific language that is much easier to master than TeX or LaTeX. It produces quite reasonable output, and especially for shorter documents it is extremely fast, though it remains a work in progress. It can be obtained from Github at `https://github.com/typst/typst`.

I am a long time user of LaTeX, in particular with LyX and while not a programmer but rather an obstetrician/gynecologist, I'm computer-literate enough to generate and use templates with perl and bash. This will be an introductory presentation, showing the comparison of some simple texts in LaTeX and typst.

Universal syllabic pattern generation

Jakub Máca, Petr Sojka, Ondřej Sojka

Space- and time-effective segmentation (hyphenation) of natural languages remain at the core of every document rendering system, be it TeX, web browser, or mobile operating system. In most languages, segmentation mimicking syllabic pronunciation is a pragmatic preference today.

As language switching is often not marked in rendered texts, the typesetting engine needs *universal* syllabic segmentation. In this article, we show the feasibility of this idea by offering a prototypical solution to two main problems: A) no wide character (`https://en.wikipedia.org/wiki/Wide_character`) support in tools like Patgen or TeX hyphenation, i.e. internal Unicode support is missing; B) A Patgen generation process for multiple languages at once.

For A), we have created a version of Patgen that uses the Judy array (`https://en.wikipedia.org/wiki/Judy_array` data structure and compared its effectiveness with the trie implementation. For B), we have applied it to generating universal syllabic patterns from wordlists of a dozen syllabic, as opposed to etymology-based, languages.

We show that A) bringing wide character support into the hyphenation part of TeX suite of programs is possible by using Judy arrays, and B) developing universal, up-to-date, high-coverage, and highly generalized universal syllabic segmentation patterns is possible, with high impact on virtually all typesetting engines, including web page renderers.

38 years with LaTeX — A personal picture story

Frank Mittelbach

As the title indicates, this is part of the story of LaTeX in pictures, as seen from my eyes. It shows many highlights throughout the years and puts faces to names — some of which are in the audience but many not. It is based on what was available in my photo archive and certainly biased, but I nevertheless hope it is of some interest.

The LaTeX Companion, 3rd edition — Anecdotes and lessons learned

Frank Mittelbach

During the last five years a lot of work went into producing a new edition of *The LaTeX Companion*. In this talk I will talk about some aspects of that work, the unique challenges and some of the lessons learned during that endeavour.

Tagged PDF, derived HTML and aspects of accessibility

Ross Moore

From a well-tagged document, conforming to the PDF/UA standard, an HTML version can be derived. With due care being taken when coding technical information, inherent semantics can be carried through the processing and exhibited within the resulting web page. This can be done in accordance with WCAG and ARIA recommendations for making information more readily available to persons with disabilities.

In a pre-prepared video, we'll see in some detail how this can be done, for sequences of authors/ panelists and their affiliations presented using footnotes in the visual PDF view.

An HTML/CSS schema for TEX whatsits

Dennis Müller

I present a schema for translating TEX whatsits to HTML/CSS. This translation can serve as a basis for (very) low-level TEX-to-HTML converters, and is in fact used by the RusTEX system for that purpose. Notably, the schema is accurate enough to yield surprisingly decent (and surprisingly often, exactly right) results on surprisingly many "high-level" LATEX macros, which makes it adequate to use in lieu of (and often even instead of) dedicated support for macros and packages.

Markdown 3: What's new, what's next?

Vít Novotný

Plain TEX, expl3, and Lua provide a common programming environment across different TEX formats. Similarly, the Markdown package for TEX has provided an extensible and format-agnostic markup language for the past seven years. In this talk, I will present the third major release of the Markdown package and the changes it brings compared to version 2.10.0, which I presented at TUG 2021.

In my talk, I will target the three major stakeholders of the Markdown package:

1. Writers will learn about the new elements, which they can type in their Markdown documents.
2. Coders will learn how they can extend Markdown with new elements and how they can style Markdown documents in different TEX formats.
3. Developers will learn about the implementation details of the Markdown package and will have a chance to discuss plans for the future governance and development of the Markdown package.

Bumpy road towards a good LATEX visual editor

Ben Davies

Overleaf has both a pure Source mode and a Visual ("Rich Text") editor. We recently redesigned the Visual editor, and we will demo the current configuration. Then, benefits, drawbacks, and specific issues this editor duality poses will be presented, together with some takeaways we have learned on the way.

Metafont, MetaPost and a complex-script typeface

Rajeesh KV

Malayalam is an Indic script with abundant shape-shifting characters. We explore a reusable component-based design for Malayalam fonts, and develop them using Metafont/MetaPost to assemble the characters. We discuss the paradigm shift from GUI design tools to 'code-based' design of shapes and glyphs, even by non-coders, and the progress our small team has made. The advantages and challenges of using Metafont/MetaPost to develop a complex-script OpenType font are discussed.

Architectural guides for Bonn — book production with ConTEXt

Henning Hraban Ramm

As one of three associates of Dreiviertelhaus publishers and as a professional typesetter and printing engineer, I'm responsible for all technical and design aspects of our books. One series is about modernist buildings in Bonn, and I'll show you how the page layout is done, including double page floats with inset captions and OpenStreetMap cartography. I'll also show examples from different layouts.

Primo — The new sustainable solution for publishing

Rishi T, Apu V, Hàn Thế Thành, Jan Vaněk

Primo is a cutting-edge, cloud-based authoring, submission, and proofing framework that provides a sustainable solution for academic publishing. It combines the advantages of XML-based workflows that facilitate controlled authoring and/or editing in accordance with specific DTDs and house styles, with the visually appealing and mathematically precise typesetting language of TEX, enabling the creation of high-quality PDFs and mathematical images (offering an alternative to MathML coding).

By speaking the widely accepted communicating lingua of mathematics and science (i.e., TEX), and utilizing the XML/MathML format for archiving, Primo has the potential to revolutionize the publishing industry. This tool caters to both the author and the publisher, bringing their needs together with enhanced participation of authors in the publishing process. The three main modules of Primo include Authoring, Submission/Reviewing, and Proofing, all of which are equipped with usability checks during submission, a collaborative editing feature, a WYSIWYG math editing tool, and publisher/journal-based PDF manuscript rendering. With Primo, authors can be assured that their work will be published with the highest level of precision and quality.

Curvature combs and harmonized paths in MetaPost

Linus Romer

Most font editors offer curvature-related tools. One of these tools is the visualization of curvature via *curvature combs*. Another tool is the so-called *harmonization*, which makes the curvature continuous along paths. An implementation of both tools in MetaPost will be presented. Curvature-optimized paths already play a significant role in METAFONT and MetaPost and therefore some example MetaPost paths will be examined for their curvature behavior.

News from the HINT project

Martin Ruckert

The HINT file format was presented at TUG 2019 and at TUG 2020, the first usable viewer for HINT files was presented. The HiTEX engine became part of TEX Live in 2022. This presentation will explore the changes that have taken place since then and what to expect in the future. The talk will focus on

- demonstrating the more recent versions of the HINT file viewer and their improvements in glyph rendering.
- demonstrating the use of links, labels, and outlines,
- explaining the capabilities of the HINT file format to convert pages to plain text for searching or text-to-speech processing,
- and presenting hints on how to design TEX macros for variable page sizes.

The `tcolorbox` inner beamer theme

samcarter

The `tcolorbox` inner beamer theme is a new theme for the beamer class. It replaces normal beamer blocks with `tcolorbox`es of the same look and feel. This allows users to easily modify the appearance of blocks. In this short talk, I will give a short overview of the theme and show some examples of how one can customise blocks.

A METAFONT for rustic capitals

Victor Sannier

I will present a typeface that I have designed using the METAFONT system, inspired by the rustic capitals used in the early centuries of our era for public inscriptions on walls, such as those found in Pompeii, and in many books and official documents written in Latin.

Producing different forms of output from XML via ConTEXt

Thomas Schmitz

This talk will showcase ConTEXt's capabilities of processing XML. It will demonstrate how one can produce different forms of PDF output (such as slides, lecture notes, handouts, or bibliographies) from a single XML file. We will explore how we can make use of ConTEXt and its integrated Lua interpreter to modify our PDF output.

Extending OpenType math, making choices

Mikael Sundqvist

In the past year and a half, Hans Hagen and I have been reviewing the typesetting of mathematics in ConTEXt LMTX. This system primarily utilizes OpenType math fonts, and during our work, we have encountered inconsistencies both within fonts and across different fonts.

Microsoft was the first to introduce OpenType math with Cambria Math in Office 2007. They have also outlined what comes closest to a standard for OpenType math, although some details are missing or debatable. Subsequently, several OpenType math fonts were created by converting and extending older TEX fonts. These fonts often inherited a more traditional "TEXy" behavior, sometimes differing from the behavior of Cambria. As a result, achieving consistent and visually appealing output with different types of OpenType math fonts has been challenging. Throughout our work, we have had to make choices that, in hindsight, could or should have been made much earlier.

In this brief talk, we will discuss some of these choices, including italic corrections, handling of zero dimension glyphs, extensibles, rules, different kern types, accents (both top and bottom), and font parameters. Our discussion will be illustrated with visual examples, to keep it accessible and less technical.

On generating documented source code by blocks in TEX

Jan Šustek

In this talk I will focus on literate programming in TEX — writing source code and its documentation in a single file. Firstly I will show an easy modification of OpTEX macros to allow literate programming. Then I will modify the macros to build the source code by nested blocks which can be built consecutively in the whole document — quite similar to the file `tex.web`, but implemented completely in TEX. Such documentation is more comprehensible to the reader.

With a few more macros or hooks, one can apply this method in the following real situations.

- Cross references make `goto` jumps easy in programming languages with line numbers.
- The abovementioned blocks can imitate subprograms with arguments in programming languages where they are not allowed.
- TEX macros can define a metalanguage and generate the source code in two different programming languages simultaneously.

Without the TEX methods the solutions would be more complicated.

Interactive and real-time typesetting for demonstration and experimentation

Didier Verna

In general, typesetting experimentation is not a very practical thing to do. WYSIWYG typesetting systems are very reactive but do not offer highly configurable algorithms, and TEX, with its separate development / compilation / visualization phases, is not as interactive as its WYSIWYG competitors. Being able to experiment with typesetting algorithms interactively and in real-time is nevertheless desirable, for instance for demonstration purposes, or for rapid prototyping and debugging of new ideas.

We present ETAP (Experimental Typesetting Algorithms Platform), a tool written to ease typesetting experimentation and demonstration. ETAP currently provides several paragraph justification algorithms, all with many configuration options such as kerning, ligatures, flexible spaces, sloppiness, hyphenation, etc. The resulting paragraph is displayed with many visual hints as well, such as paragraph, character, and line boxes, baselines, over/underfullness hints, hyphenation clues, etc. All these parameters, along with the desired paragraph width, are adjustable interactively through a GUI, and the resulting paragraph is displayed and updated in real-time.

But ETAP can also be used without, or in conjunction with the GUI, as a scriptable application. In particular, it is able to generate all sorts of statistical reports or charts on the behavior of the various algorithms, for instance, the number of over/underfull boxes per paragraph width, the average compression or stretch ratio per line, whatever else you want. This allows you to quickly demonstrate or evaluate the comparative behavior or merits of the provided algorithms, or whichever you may want to add to the pool.

The update of the `nostarch` class

Boris Veytsman

I wrote the `nostarch` class for No Starch Press more than a decade and half ago. It accommodated many specific features of No Starch books: the special formatting for the first paragraph in a chapter, the characteristic artwork at chapter starts, etc. Since then it has been used and modified by the publishing team (with special thanks to Alex Freed for many fixes).

The wide adoption of Overleaf by many authors has increased the number of LATEX submissions, thus creating a demand for a rewrite. The main goal of the update is to minimize the need for manual adjustments, while implementing the current design requirements of the publisher. Some of these requirements are rather unusual: the rules of url breaking, the variable width of captions for figures and tables, etc. In this talk I discuss the challenges and solutions of the redesign.

An updated survey of OpenType math fonts

Ulrik Vieth

OpenType math fonts have been around for 15 years now. In recent years, more and more OpenType math fonts have been added. In this talk, we will review some of the recent additions, such as Libertinus Math, Garamond Math, Erewhon Math, XCharter, KpMath, New Computer Modern, Concrete Math, Euler Math, comparing them to the previous repertoire of OpenType math fonts such as Latin Modern and TEX Gyre.

Supporting backends in expl3

Joseph Wright

The backend in TEX is responsible for the parts of producing output that TEX doesn't know about, for example colour, image inclusion and hyperlink creation. Each backend has its own syntax and range of supported concepts, so at the macro level there needs to be the appropriate code to 'talk' to the backend. In expl3, we have developed a consistent set of backend support files, based on the experience of (LA)TEX developers over 30+ years of working with these backends. Here, I will look at the history of backend abstraction and the model used in expl3.

Beamer news

Joseph Wright, samcarter

The beamer class is used by many users all around the world to create slides for their presentations. This talk will present some changes and new features, which were added over the last few years and which might be interesting to know for beamer users.

Further adventures in Unicode-land: Refining case changing

Joseph Wright

Getting text processing right for Unicode in TEX is a challenge, particularly where one wants to support the full range in pdfTEX. Over the past few years, I have worked on one aspect: case changing. Code to carry out the Unicode case changing algorithm was integrated into the LATEX kernel a couple of years ago. Since then, we have been refining the details, adding more power and discovering new issues. Here, I'll look at what we've done to get the code working smoothly, and look forward to what might still be improved.

What every (LA)TEX newbie should know

Barbara Beeton

Abstract

LaTeX has a reputation for producing excellent results, but at the cost of a steep learning curve. That's true, but by understanding a few basic principles, and learning how to avoid some techniques that may seem obvious but often lead one into the weeds, it's possible to avoid some of that pain.

This presenttion is based on years of looking at good and bad document input and output, answering questions from problem-plagued authors, and trying to write documentation that can be understood on first reading.

Another source of material is the collection of questions and answers provided by the TeX segment of StackExchange. Many newbie questions appear over and over again. Good "duplicate" answers for these have been identified, and links are collected as "Often referenced questions", found at `https://tex.meta.stackexchange.com/q/2419`.

Conventions

In order to avoid overfull lines, error and warning messages shown here will be broken to fit the narrow columns of this article style. Many error messages output by LaTeX will consist of several lines, the first being the message, and the next showing the number of the line on which the error is identified along with the content of that line, up through the error text. A following line, indented so that it, with the numbered line, completes the line as it appears in the input.

Although this presentation will mostly deal with details, please remember that the basic concept of LaTeX is to separate content from structure.

Another applicable concept, one that is often misconstrued in the (LA)TeX community is that of "template". When that term is used here, it means a source file that is an "outline" beginning with `\documentclass` and containing a minimum of basic structural commands into which text and additional definitions can be inserted as appropriate.

Basic structure: Commands, modes and scope

Instructions are communicated to (LA)TeX by means of commands, or "control sequences", which by default begin with a backslash (\). There are two varieties: those which consist of the backslash followed by one character ("control symbol"), and multi-letter commands ("control words") in which only letters (upper- or lowercase) are permitted (no digits or special characters). A control word will be terminated by a space or any other non-letter. But a space after a control symbol will appear as a space in the output.

A user can define new commands, or assign new meanings to existing commands. It's advisable to use `\newcommand` when creating a new definition; this checks to make sure that the command name hasn't been used before, and complains if it has. If it's necessary to redefine a command that already exists, the recommended way is to use `\renewcommand` — but be sure you know what you're doing. For example, redefining `\par` is chancy, as LaTeX uses this "under the covers" for many different formatting adjustments, and it's very easy to mess things up. Single-letter commands are also bad candidates for (re)definition by users, as many of them are predefined as accents or forms of letters not usual in English text; redefining `\i`, for example, can give a nasty surprise if there is the name of a Turkish author in your bibliography. But single-digit commands are not predefined in core LaTeX, so are available for ad hoc use.

TeX, and therefore LaTeX, functions in several distinct modes:

- horizontal — text,
- vertical — beginning of job and between paragraphs,
- math — two varieties: in-text and display.

Starting to input ordinary text is one way to enter horizontal mode. A blank line or explicit `\par` will transition from horizontal to vertical mode. Some operations are limited to a particular mode, or are most effective and predictable within such a mode. For example, it's best to specify `\vspace` and most floats while in vertical mode.

Along with modes, there is the concept of scope, making it possible to localize definitions and operations.

Math mode is one instance of scope; certain characters and operations are valid only within math, and others are invalid there. Within text, math usually begins and ends with `$`, and these must be matched. Display math breaks the flow of text; closing a display returns to text mode unless followed by a blank line or `\par`. More about math later.

Another way of delimiting scope is to wrap it in braces: `{...}`. Within this scope, the meaning of a command may be changed for temporary effect; the definition in effect before the opening brace will be restored as soon as the closing brace is digested. Instead of a brace pair, the commands `\begingroup`...`\endgroup` have the same effect.

In LaTeX, closed environments can be defined, inside which the conditions may be quite different

than in the surrounding material. Such environments begin with `\begin{`⟨*env-name*⟩`}` and end `\end{`⟨*env-name*⟩`}`. One example is the `theorem` environment, inside which text is italic. If the environment name at the `\end` doesn't match the one used at `\begin`, an error will be reported:

```
! LaTeX Error: \begin{...} on input line ...
    ended by \end{...}.
```

How to end a paragraph: Not with \\

`\\` does end a line. It is the designated command to end lines in tables, poetry, multi-line math environments, and some other situations. But it does not end a paragraph. A paragraph is ended by a blank line or an explicit `\par`.

Trying to end a paragraph with `\\` can result is some confusing warnings and error messages. For example, `\\` on a line by itself will result in this warning:

```
Underfull \hbox (badness 10000)
    in paragraph at lines ...
```

Furthermore, if the `\\` is preceded by a (typed) space, in addition to the above warning, there may be an extra, unwanted, blank line in the output.

If extra vertical space *is* wanted after a line broken with `\\`, it can be added by inserting an optional dimension, wrapped in brackets: `\\[`⟨*dimen*⟩`]`. If such a bracketed expression is really meant to be typeset, it must be preceded by `\relax`.

Spaces. spurious and otherwise unwanted

A goal of high-quality typesetting is even spacing in text. This is really possible only with ragged-right setting, but even margins are usually preferred, so TEX is designed to optimize spacing in that context.

By default, multiple consecutive spaces are interpreted as a single space. Also, a slightly wider space is left at the end of a sentence, making it easy to tell where the sentence ends. (In French typography, or in the presence of `\frenchspacing`, all spaces are treated the same.) When other unequal spacing is observed in a line, something is fishy.

A sentence is presumed to end with a period or similar punctuation. But abbreviations also end with periods, and abbreviations occur frequently in academic documents, and the wider space isn't wanted there. To indicate an ordinary space, insert a backslash after the period, as in `e.g.\ this or that`, or, if the line should not break after the abbreviation, insert an unbreakable space, as in `Dr.~Knuth`.

A similar, but reverse, situation can occur when an uppercase letter is followed by a period. This is assumed to be the initial of a name; it usually is, and an ordinary interword space is set. But sometimes the uppercase letter is at the end of an acronym, and that ends a sentence. In such a case, add `\@` before the period, and it will restore the wider end-of-sentence space.

But sometimes wider spaces appear in text where they are not expected. This is often caused by spaces inadvertently included in definitions. The end-of-line (here called EOL) is interpreted as a space. (Different operating systems define an EOL differently, but that is taken care of by the TEX engine.) A neatly laid-out definition may be the culprit:

```
\newcommand{\abc}{
    \emph{abc def}
}
```

will output unwanted spaces *abc def* when used. This can be avoided by placing a `%` sign at the ends of the lines that cause the problem:

```
\newcommand{\ABC}{%
    \emph{abc def}%
}
```

Then using that command *abc def* will not have the unwanted spaces.

It isn't necessary to use the `%` sign after a control word; remember that a space there just ends the command and is then discarded. But there are places where adding a `%` can cause trouble. After defining any numeric value, TEX will keep looking for anything else that can be interpreted as numeric, so if a line ends with `\xyz=123`, no `%` should be added. Or, if setting a dimension, say `\parindent=2pc`, TEX will keep looking for `plus` or `minus`; a better "stopper" is an empty token, `{}`. (If "plus" or "minus" is there and happens to be actual text, a confusing error message will be produced, but that is rare, and beyond the scope of this presentation.)

There are some other, more obscure situations where unwanted spaces can show up. One is when multiple index entries are inserted in the same place in a file. Often, these are placed on separate lines, and the EOL principle takes effect. Since the multiple spaces aren't consecutive, they remain in the output. Add `%` signs judiciously, remembering to keep one intentional space.

I learned just recently of a really obscure and surprising space. It occurs, like this, in the middle of a w or d, and is caused by the application of a small frame around the colored element by the tcolorbox. This must be suppressed explicily, like this:

```
\usepackage{tcolorbox}
\newcommand{\pink}[1]{{\fboxsep=0pt
    \colorbox{red!20}{#1}}}
```

The resulting word is colorized with no unwanted spaces. While this is really beyond the scope of this presentation, it's something that one should be aware of. If it happens, seek expert assistance.

Font changes

Font changes are a time-honored method of communicating shades of meaning or pointing out distinct or particularly important concepts. Many such instances are built into document classes and packages; for example, theorems are set in an *italic* font, section headings in **bold**, and some journals set figure captions in sans serif to distinguish them from the main text.

LaTeX provides two distinct methods for making font changes. Commands of one class take an argument and limit the persistence of the change to the content of that argument; these have the form of `\textbf{...}` for **bold**, `\textit{...}` for *italic*, etc. The other class sets the font style so that it will not change until another explicit change is made, or it is limited by the scope of an environment; some examples are `{\itshape...}`, `{\bfseries...}`, and `{\sffamily...}`. These command names are best looked up in a good user guide.

Several font-changing commands do different things depending on the context. `\emph{...}` will switch to italic if the current text is upright, or to upright if the current text is italic. Within math, `\text{...}` will set a text string in the same style as the surrounding text; thus, within a theorem, `\text{...}` will be set in italic. If this string should always be upright, like function words, `\textup{...}` should be used instead.

Basic TeX defined two-letter names for most font styles. All of these are of the persistent type. They should be avoided with LaTeX, as some of the LaTeX forms provide improvements, such as automatic application of the italic correction, which would otherwise have to be input explicitly.

Math

Math is always a closed environment. If started, it must be ended explicitly and unambiguously. Within text, math begins and ends with `$`; there must therefore be an even number of `$` signs in a document. LaTeX also provides `\(...\)` for in-text math, but most users stick with the `$`. Many different display environments are defined by the packages amsmath and mathtools, and it is worthwhile to learn them by reading the user guides. (mathtools loads amsmath, so it's not necessary to load amsmath separately.)

Within math, all input spaces are meaningless to (La)TeX; they can be entered in the source file as useful to make it readable to a human. Blank lines, however, are considered errors. This was a decision in the design of TeX to make it easy to detect an unterminated math element, because math should not span a paragraph break. In both in-text math and displays, the error message will be

```
! Missing $ inserted.
```

If a blank line occurs in a multi-line display environment from amsmath, the *first* error message will be

```
! Paragraph ended before ⟨env-name⟩
    was complete.
<to be read again>
```

This will be followed by *many* more error messages, all caused by the first. These will be confusing and misleading. Always fix the problem identified by the first error and ignore the rest; they will disappear once the first error is fixed, here, the blank line is removed.

If the appearance of a blank line is wanted for readaility, begin it with a `%` sign. It's also bad form to leave a blank line before display math; a display is usually a continuation of the preceding paragraph, and avoiding a paragraph break also avoids a possible page break before the display.

As in other environments, the `\end` name must exactly match the name specified at `\begin`. A "shorthand" for a single-line, unnumbered display is `\[...\]`. The environments designed for multi-line displays should not be used for a single-line display.

Although LaTeX provided `eqnarray` as a display environment, don't use it. If the display is numbered and the equation is long, the equation can be overprinted by the equation number.

Tables, figures, and other floats

The number of floats, their positions on a page, and the spacing around and between them is defined by the document class. So if something doesn't work as you expect (hope for?), any potential helper will insist on learning what document class is being used.

Input for a float must appear in the source file while there is still enough space on the output page to fit it in. In particular, on two-column pages, a `\begin{figure*}` or `\begin{table*}` must occur in the source *before* anything else is set on the page. The basic float handling does not allow full-width floats to be placed anywhere but at the top of a page; some packages extend this capability, but those won't be discussed here.

Here are the defaults for the basic `article` class.

- Total number of floats allowed on a page with text: 3.

- Number of floats allowed at top of page: 2. Percentage of page allowed for top-of-page floats: 70%/
- Number of floats allowed at bottom of page: 1. Percentage of page allowed for bottom-of-page floats: 30%/
- Minimum height of page required for text: 20%.
- Minimum height of float requiring a page by itself: 50%.

The reference height is `\textheight`. That is, the height of page headers and footers is excluded.

If an insertion is small, must be placed precisely and fits in that location, don't use a float. `\includegraphics` or one of several available table structures should be used directly, often wrapped in `\begin{center} ... \end{center}`/ (Within a float, use `\centering` instead.)

The wrapfig package supports cut-in inserts at the sides of a page or column. Refer to the documentation for details.

By tradition, captions are applied at the top of tables and the bottom of figures. If an insertion is not a float, the usual `\caption` can't be used. Instead, `\usepackage{caption}` and the command `\captionof`.

The document class and preamble

When embarking on a new document, the first thing is to choose the document class. If the goal is publication in a particular journal, check the publisher's instructions to see what is required. Many, but not all, popular journal classes are available from CTAN. If the project is a thesis or dissertation, find out the special requirements, and if your institution provides a tailored class, obtain a copy. Try to determine whether it is actively maintained, and if there is local support. Read the documentation. It is the responsibility of the document class to define the essential structure of the intended document. If the document you are preparing differs in essential ways from what is supported by the document class, the time to get help is *now*.

There will be features not natively supported by the document class; for example, the choice of how to prepare a bibliography may be left to the author. This is why packages have been created.

Most packages are loaded in the preamble; the one exception is `\RequirePackage`, which may be specified before `\documentclass`, and is the place where options should be loaded. Some authors create a preamble that is suitable for one document, then use the same preamble for their next document, adding more packages as they go. And some unwitting newbies "adopt" such second-hand "templates" without understanding how they were created. *Don't do it!*

Start with a suitable document class and add features (packages, options, and definitions) as they become necessary. Organize the loading of packages into logical groups (all fonts together, for example), and be careful not to load a package more than once; if options are needed, any loaded with a non-first `\usepackage` will be ignored. Some packages automatically load other packages; for example, mathtools loads amsmath and amssymb loads amsfonts. And, very important, pay attention to the order of package loading: hyperref must be loaded (almost) last; the few packages that must come after hyperref are all well documented. Read the documentation.

Processing the job

Once the file is created, it's time to produce output. There are several engines to choose from: pdfLaTeX, XeLaTeX, and lusLaTeX. These can be run interactively from the command line, or initiated from an editor. Assuming there are no errors, how many times the file must be processed depends on what features it contains.

(La)TeX is "one-way". If any cross-references or `\cite`s are present, this information is written out to an `.aux` file; information for a table of contents is written to a `toc` file, and other tables are also possible. The bibliography must be processed by a separate program (and its log checked for errors) with the reformatted bib data written ont to yet another file. Then LaTeX must be run (at least) twice more — once to read in the `.aux` and other secondary files and include the bibliography and resolved cross-references, and the second time to resolve the correct page numbers (which will change when the TOC and similar bits are added at the beginning).

All this assumes that there are no errors. Errors will be recorded in the log file. Learn where the log file is located, and make a habit of referring to it. Warnings, such as those for missing characters, will also be recorded there, but not shown online:

`Missing character: There is no` ⟨*char*⟩ `in font` ⟨*font*⟩`!`

In the log, errors may appear with closely grouped line numbers. If so, and the first is one that interrupts the orderly processing of a scoped environment, following errors may be spurious. So fix the first error and try processing before trying to understand the others; often, they may just go away.

Good luck. With practice comes understanding.

Barbara Beeton

News from the HINT Project

Martin Ruckert

Abstract

The HINT file format[5] was presented at TEX Users Group 2019[4] and at TEX Users Group 2020[6], the first usable viewer for HINT files was presented. The HiTEX engine became part of TEX Live in 2022. This presentation will explore the changes that have taken place since then and what to expect in the future. The talk will focus on

- demonstrating the more recent versions of the HINT file viewer and the improvements in glyph rendering;
- demonstrating the use of links, labels, and outlines;
- explaining the capabilities of the HINT file format to convert pages to plain text for searching or text-to-speech processing; and
- presenting hints on how to design TEX macros for variable page sizes.

1 Displaying Glyphs

Initially, the HINT viewer did support only `.pk` fonts. These font files contain METAFONT fonts at a fixed resolution, usually at 600 dpi. Rendering such a font on a computer screen with a typically much lower resolution, was done in three steps:

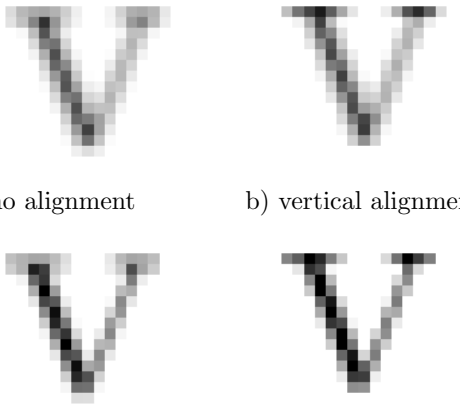
1. Decoding the font file header and caching it for later use.
2. Decoding a glyph into a black and white bitmap and caching it for later use.
3. For each pixel on screen intersecting the glyph's bounding box
   - map the pixel center to a source point in the glyph's bitmap and
   - compute the pixel's gray value by linear interpolating the black and white values of the four pixels surrounding the source point in the bitmap.

Since high dpi values, often above 300 dpi, are common on small mobile devices, the results were more than acceptable on these devices. On ordinary computer screens typically with dpi values below 100, the results were insufficient. Especially the rendering of thin lines would distribute the available amount of black ink over a two pixel wide area and the line would fade away into a blurry light-gray.

Things changed with the use of the FreeType font rendering library[7]. This library can render PostScript Type 1 outline fonts at any resolution desired. After replacing the `.pk` fonts by `.pfb` fonts, the viewer could render the glyphs as gray-value bitmaps for the actual screen resolution[3]. To produce good looking glyphs from an outline font, first the positions of key points of the outline, for example the points where the outline has a horizontal or vertical tangent, will be rounded to the pixel grid. After that, pixels that are only partly covered by the outline will be assigned gray values depending on the amount of coverage. This will result in less blur and consistent stroke widths. Improving the readability especially for small font sizes.

The quality of the font rendering in the HINT viewer was, however, still inferior to a rendering of the same font by other programs. The reason was, that the viewer would not map the glyph bitmap one to one to the screen but instead would map the bitmap to TEX's exact glyph position – usually not aligned to the pixel grid – using step 3 as given above.

To improve the readability at small font sizes, the current viewer will round the glyph position to the pixel grid before rendering the glyph. And it replaces the linear interpolation of pixel values by using the gray value of the nearest source pixel. The rounding will occur only if the font size is below a given threshold. In principle the rounding can be split into rounding horizontal and rounding vertical position. While the first affects character distances, the latter moves entire lines and is less distracting. For a demonstration see [3].



a) no alignment b) vertical alignment

c) horizontal alignment d) full alignment

Figure 1: A cmr 10pt V with different alignment to the pixel grid.

Further improvements are possible, but not yet implemented. One method is oversampling, where a

glyph is rendered at, for example, four different horizontal positions on the pixel grid. Choosing one of these four renderings, the horizontal glyph position must be rounded to 1/4 of the pixel size which is far less distracting. Another method is sub-pixel rendering. This method uses the fact that one white pixel on screen actually consists of three colored dots: red, green, and blue. So by considering them a independent light sources, the horizontal resolution can be tripled. This improves the positioning but leads to colored borders which some people find distracting.

2 Links, Labels, and Outlines

People my age have learned navigating through thick books already in primary school, if not in kindergarten. These skills are more or less obsolete when it comes to navigating through "thick" electronic documents. So good replacements are necessary. The most obvious point to start exploring a book is its table of content where for each section the corresponding page number is listed. The HINT file format supports the concept of a home page: a position in the document identified be the author that can be reached in the viewer with a single key stroke, touch, or click. The HINT document, however, has no fixed page numbers. The pages grow and shrink with the window size (and with the magnification factor). So instead a table of content must use a click-able link that brings you immediately to the section in question. Similar links are used for the table of figures, the index, and for all kinds of cross-references, be it to individual parts of the text, a figure, a table, a citation, or a displayed formula.

As an alternative to the table of content, the HINT file format also supports "outlines": A clickable table of content, hierarchically organized and displayed in a separate window. To allow optimal use of the available space, sub levels of the hierarchy can be hidden or expanded as needed[3].

In the mean time, the LaTeX hyperref package offers support for most of the above features.

In one respect HINT files are radically different from books or pdf files: There are no predefined pages. So following a link is not as simple as displaying a page with a given page number, but it requires finding two good page breaks so that the target is on the page between them. The algorithm used in the current HINT viewer is still under development and there are cases where the choice of page breaks could be better.

3 Designing Macros for Variable Pages

The traditional implementation of centering text is the `\centerline` macro. It expands to `\hbox` to `\hsize` { `\hfill` *text* `\hfill` } which will look nice as long as the *text* is shorter than `\hsize`. If the text is longer, it will produce an overfull box, stick out into the margin, and even goes over the edge of the window. A better solution uses TeX's line breaking procedure[3], which requires a vertical box.

```
\vbox{\rightskip 0pt plus2em
      \leftskip=\rightskip
      \parfillskip=0pt\parindent 0pt
      \spaceskip.3333em
  \xspaceskip.5em\relax
This is Text Centered on the Page
}
```

Letting `\rightskip` and `\leftskip` stretch enough, but not too much, so that the line breaking routine will try to keep the lines filled but still has enough room to produce decent lines. The inter-word-glue, on the other hand, is prevented from stretching. (It could be made to allow for some shrinking to gain additional flexibility.)

The only new feature introduced in HiTeX since 2019 is the support for `\vtop`. This is important because writing for variable page sizes often requires replacing a horizontal box by a vertical box to enable the breaking of paragraphs into lines. `\vtop` is required if multiple vertical boxes need to be aligned on the top baseline[3].

4 Searching

The user input into a search field is just a plain sequence of characters coded in UTF8 or some local encoding like ISO 8859-1. The text, as represented in a TeX document is far more complex and searching requires finding a match between both representations. Even if the input consist only of ASCII characters the HINT viewer must handle some special cases.

If the word the user wants to find uses a ligature, the match is made using the replacement characters, that are retained in TeX's ligature node. If the word on the page is hyphenated and split across two lines, the match must ignore extra characters inserted by the pre and post hyphenation lists as well as the space that is usually separating the word at the end of one line from the word that starts the new line. Indeed the HINT backend provides a function, that converts entire pages into sequences of characters moving from top left to bottom right, eliminating the effects of ligatures and hyphenations and condensing various combinations of glue – inter-word

glue, baseline skips, left skips, right skips, and indentations to name just a few – to a single space. Kerns, on the other hand, are completely ignored. An infelicity here is the definition of the LaTeX macro, which uses a glue instead of a kern between 'A´ and 'T´. So you have to search for "`LA TEX`".

It is planed to use the page to string function also to feed a Text-to-Speech converter.

Currently searching does not work well with non ASCII characters, but it is planed to implement UTF8 as the default encoding used for HiTeX and HINT files.

5 New Viewers for Linux, MacOS, and iOS

Together with the viewers for Windows and Android, the applications for Linux, MacOS, and iOS complete the set of Viewers. The Windows application, being the oldest and my work-horse for conducting experiments, is the most complex. The application for MacOS is the most recent and was presented at Jonathan Fines's TeX hour[1, 3]. The application for Linux is the most simple. It consists beside the backend and the OpenGL renderer (shared between all applications) only of a 600 line main program[2]. This is a good starting point for writing your own viewer.

References

[1] Jonatan Fine, Martin Ruckert, et al. Rethinking TeX in STEM. `https://texhour.github.io/2022/09/29/rethink-tex-in-stem/`, 9 2022.

[2] Martin Ruckert. Hint source repository. `https://github.com/ruckertm/HINT`.

[3] Martin Ruckert. The HiTeX video collection. `http://hint.userweb.mwn.de/hint/video/`.

[4] Martin Ruckert. The design of the HINT file format. *TUGboat*, 40(2):143–146, 2019.

[5] Martin Ruckert. *HINT: The File Format*. 2019. ISBN 1-079-48159-1.

[6] Martin Ruckert and Gudrun Socher. The HINT project: Status and open questions. *TUGboat*, 41(2):208–211, 2020.

[7] David Turner, Werner Lemberg, et al. Freetype. `http://www.freetype.org/`.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
`martin.ruckert@hm.edu`

An HTML/CSS Schema for TEX Primitives – Generating High-Quality Responsive HTML from generic TEX

Dennis Müller

This paper uses STEX3. The semantically annotated XHTML version of this paper is available at `url.mathhub.info/tug23css`

Abstract

I present a schema for translating TEX primitives to HTML/CSS. This translation can serve as a basis for (very) low-level TEX-to-HTML converters, and is in fact used by the RUSTEX system – a (somewhat experimental) implementation of a TEX engine in Rust, used to convert LATEX documents to XHTML– for that purpose.

Notably, the schema is accurate enough to yield surprisingly decent (and surprisingly often "the exactly right") results on surprisingly many "high-level" LATEX macros, which makes it adequate to use in lieu of (and often even instead of) dedicated support for macros and packages.

1 Introduction

Translating LATEX documents (partially or fully) to HTML is a difficult problem, primarily because the two document formats address very different needs: TEX is intended to produce statically layout documents with fixed dimensions, ultimately representing ink on paper. HTML on the other hand assumes a variety of differently sized and scaled screens and consequently prefers to express layouts in more abstract terms, the typesetting of which are ultimately left to the browser to interpret; ideally responsively – i.e. we want the document layout to adapt to different screen sizes, ranging from 8K desktop monitors to cell phone screens.

This means that there is no one "correct" way to convert TEX to HTML– rather there is a plurality of choices to be made; most notably, which aspects of the static layout with fixed dimensions described by TEX code to preserve, or discard in favour of leaving them up to the rendering engine, explaining the plurality of existing converters.

Naturally, many LATEX macros are somewhat aligned with tags in HTML; for example, sectioning macros (like `\chapter`, `\section`, etc.) correspond to `<h1>`, `<h2>`, etc; and the `\begin{itemize}` and `\begin{enumerate}` environments and the `\item` macro correspond to `<ul>`, `<ol>` and `<li>`, respectively. Most converters therefore opt for the reasonable strategy of mapping common LATEX macros directly to their closest HTML relatives, with no or minimal usage of (simple) CSS; effectively focusing on preserving the *document semantics* of the used constructs (e.g. "paragraph", "section heading", "unordered list"). In many situations, this is the natural approach to pursue, especially if we can reasonably assume that the document sources to be converted are sufficiently "uniform", so that we can provide a similarly uniform CSS style sheet to style them, and this is largely the way existent converters work. To name just a few:

LATEXML [6] focuses strongly on the *semantics*, using XML as the primary output format and heuristically determining an author's intended semantics of everything from text paragraphs (definitions, examples, theorems, etc.) down to the meaning of individual symbols in mathematical formulae; achieving great success with `ar5iv.org`, hosting HTML documents generated from TEX sources available on `arxiv.org`. TEX4ht [12] focuses on plain HTML as output with minimal styling, going as far as to replace the `\LaTeX` macro by the plain ASCII string "`LaTeX`". Pandoc [10] largely focuses on the most important macros and environments with analogues in all of its supported document format to convert between any two of them, e.g. TEX, Markdown, HTML, or `docx`. Mathjax [5] focuses exclusively on macros for mathematical formulae and symbols, allowing to use TEX syntax in HTML documents directly, which are subsequently replaced via JavaScript by the intended presentation.

However, the approach described above has notable drawbacks: Firstly, it requires special treatment of LATEX macros that plain TEX would expand into primitives instead, and the amount of LATEX macros is virtually unlimited – CTAN has (currently) a collection of 6399 packages, tendency growing, which get updated regularly, and authors can add their own macros at any point. Supporting only the former is a neverending task, and providing direct HTML translations for the latter is impossible. This is made worse by the very real and ubiquitous practice among LATEX users of copy-pasting and reusing various macro definitions and preambles assembled from stackoverflow, friends and colleagues, and handed down for (by now *literally*) generations, even in situations where (unbeknownst to them) "official" packages with better solutions (possibly supported by HTML converters) exist.

For example, I myself have happily reused the following macro definition for years:

```
\usepackage{amsmath,amssymb}
\def\forkindep{\mathrel{\raise0.2ex\hbox
  {\ooalign{\hidewidth$\vert$\hidewidth
```

```
\cr\raise-0.9ex\hbox{$\smile$}}}}}
```

...neither knowing nor caring what it actually does other than that it allowed me to typeset $A \downarrow_C B$ ("A and B are *forking-independent* (or *non-forking*) over C"; a concept in model theory)[1] – despite there existing a unicode symbol (0x2ADD) and a corresponding LaTeX macro `\forksnot` in the `unicode-math` package. If we want to maximise coverage, we therefore need a reasonable strategy for arbitrarily elaborate unexpected LaTeX macros.

Secondly, by generating rather plain HTML, we guarantee that the resulting presentation is *neutral* and can be easily adapted by users via their own CSS stylesheets – the "morally correct" thing to do. However, it also severely clashes with the expectations of (casual) users that the result looks roughly the same as the PDF does. After all, the way LaTeX documents are written by authors is optimized for a particular layout and arrangement of document elements. Subsequently discarding them in favor of as-plain-as-possible HTML that optimizes more for the "document semantics" of the components than their (precise) optics yields plain looking HTML that is immediately perceived as ugly, "not what I want" and requires lots of massaging to achieve a similar aesthetic level as the PDF generated by pdflatex does. And *aesthetics matter* – that's why TeX was built in the first place.

Thirdly, by focusing on supporting as many LaTeX macros as possible directly, conversion engines tend to neglect support for primitives in multiple senses of "support" – indeed, I found it difficult finding any existing TeX documents of mine that "survive" any of the existing HTML converters for a realistic comparison, typically dying with no output or only initial, badly formatted fragments.

The RusTeX system is a TeX-to-HTML converter born out of our needs in the sTeX project [4, 8]. The `stex` package allows for annotating LaTeX documents (in particular mathematical formulae and statements) with their (flexi-)formal semantics. These documents are subsequently converted to HTML, preserving both the (informal) document layouts as well as the semantic annotations in such a way, that knowledge management services acting on the semantics can be subsequently integrated via JavaScript. Our existing corpora of sTeX documents cover a wide range, from individual fragments (definitions, theorem statements, remarks,...) up to research papers, lecture slides in `beamer`, and book-like lecture notes that usually *include* the slides between text fragments, all of them using a multitude of (typical and untypical, official and custom) packages, preambles and stylings.

We consequently want to translate the sources for all these heterogeneous documents to HTML such that 1. the results look as similar to their PDF counterparts as possible, 2. the semantic annotations are preserved as XML attributes, and 3. (most importantly) conversion succeeds for any error-free document, regardless of packages and macros used, so that at least the semantic annotations can be extracted, even if the presentation is occasionally (somewhat) broken.

Contribution Motivated by the above, this paper describes RusTeX's rather extremal point in the design space of LaTeX-to-HTML converters: The goal is to mimic the core TeX expansion mechanism (i.e. pdflatex) as closely as possible and map the resulting sequence of TeX primitives to (primarily) `<div>`s with CSS attributes, while avoiding the neverending amount of work required for the special treatment for non-primitive TeX macros. Ideally, this allows for achieving full error-free coverage with respect to converting full documents, and yielding HTML that looks reasonably close to what a user would expect.

Of course, if we *only* care about aesthetics, we might as well render the generated PDF in the browser directly. So as an addendum to the above, we should add the desideratum that the HTML remain "reasonably recognizable as HTML": for example, plain text in paragraphs (or horizontal boxes) should actually be represented as plain text in the resulting HTML– in fact, as much as possible we want to leave to the browser what a browser does best: break lines in paragraphs, size boxes based on their contents (where we want them to be), and arrange components based on available (screen) space, according to constraints imposed by our CSS schema.

RusTeX's git repository [11] contains a `.tex`-file with test cases for (and beyond) all the following, and the HTML generated from them for direct comparison. Additionally it contains the PDF and HTML produced from my Ph.D. dissertation [7], which serves as a particularly good test case for several reasons:

1. I was a typical LaTeX user when I wrote it, with no particular knowledge of TeX's internal workings, and hence unbiased by what I would nowadays do to avoid problems.
2. I spent a lot of effort on making it look nice by the usual means – copy-pasting from elsewhere

[1] Possibly sourced from `tex.stackexchange.com/questions/42093/what-is-the-latex-symbol-for-forking-independent-model-theory` – I needed and found it some time around 2013.

and using whatever package google tells me to use to achieve the desired effect.

3. It is a 215 page document using everything from elaborate formulas, syntax-highlighted code listings, various figures and tables, and color-coded environments (using `tcolorbox`es) for remarks, theorems, examples, definitions, etc.

The HTML generated from our STEX corpora can be found at `url.mathhub.info/stex`, including this paper (see link above), which thus additionally serves as a demonstration of the examples below (notably, with two column mode deactivated). They also power our *course portal* at `courses.voll-ki.fau.de`, where students at our university can access semantically annotated course materials and various didactic services generated from them.

The full CSS schema can be found at [2].

Disclaimer Note that I am not arguing to eschew dedicated support for LATEX and package macros entirely – document semantics can be important, for example for accessibility reasons. Additionally, while the translation presented here is surprisingly effective, it has clear limitations, especially on the scale of *individual characters* (see section 9).

Hence, the contents of this paper should be seen as a reasonable *fallback* strategy usable *in conjunction with* dedicated support for macros. Indeed, RUSTEX too currently implements (few) package macros as well, namely `\url`, `\not` and `\cancel`, `\underbrace` and `\overbrace`, `\marginpar`, `\begin{wrapfig}` and (somewhat embarrassingly) `\LaTeX`.

In fact, if this paper has a purpose beyond reporting on what I consider to be an interesting experiment, it should be the following: *Taking TEX primitives seriously pays off aesthetically*, can spare a lot of work and effort, and where possible, I encourage developers of TEX-to-HTML converters to take them seriously *in addition to* dedicated support for macros.

Furthermore, many of the techniques described below are the result of more-or-less informed experimentation; in many cases, better ways to represent TEX primitives in HTML might exist. I appreciate feedback and suggestions for improvements.

2 General Architecture

As mentioned, RUSTEX attempts to mimic the behaviour of `pdflatex` as closely as possible. As such, it implements the behaviour of the primitive commands available in plain TEX, eTEX and pdfTEX, amounting to 293 + 47 commands, excluding primitive "register-like" commands such as `\everyhbox`, `\baselineskip` or `\linepenalty`. Their precise behaviour has been determined from (obviously) the bible [3] and the manuals for eTEX and pdfTEX, but also often reverse engineered via extensive experimentation.

At the start of the program, a user's `pdftexconfig.tex` and `latex.ltx` are located using `kpsewhich` and processed first. This entails that a user needs to have a LATEX distribution set up, but subsequently makes sure that RUSTEX behaves as close to the local LATEX setup as possible.

Tokens are expanded in the expected manner down to the primitives, which cause state changes, impact expansion, or ultimately end up fully processed in RUSTEX's *stomach* waiting to be output as HTML. The latter primitives are the subject of this paper.

`pgf` (and thus `tikz`) is handled via an adapted version of the existing SVG driver and thus omitted here. Images are inserted directly in the HTML in Base64-encoding.

In lieu of a *shipout routine*, box registers for *floats* (as well as `\insert`s such as footnotes) are occasionally heuristically inspected and inserted, but this mechanism is due for a more adequate treatment and hence also omitted.

2.1 Trees and Fonts

Naturally, HTML is a tree structure of nested nodes. So, somewhat counterintuitively, so are TEX's stomach elements, but unfortunately at the cost of attaching information such as the current font, font size, color, etc. directly to the individual "character boxes". If we wanted to introduce a `<span>` node for every individual character, we could mimic this directly in HTML– however, this approach is too extreme even for my taste. Luckily, in almost every situation where colors and fonts are changed, the changes are achieved via LATEX macros that align with TEX's 'stomach tree". For example,

```
\textbf{\textcolor{blue}{some} \emph{text}}
```

clearly entails a tree of font and color changes, which ideally should be represented as a corresponding HTML tree:

```
<span style="font-weight:bold">
  <span style="text-color:blue">some</span>
  <span style="font-style:italic">text</span>
</span>
```

And indeed, all three macros (`\textbf`, `\textcolor`, `\emph`) introduce TEX groups for their arguments, assuring that these changes too reflect a tree structure.

[2] `github.com/slatex/RusTeX/blob/master/rustex/src/resources/html.css`

Consequently, RusTeX can (somewhat) safely add special nodes to the stomach on font changes, changes to the color stack, or *links* (as produced by `\pdfstartlink`). As these are (usually) local to the current TeX group, the stomach consequently also keeps track of when TeX groups are opened and closed. If such changes (i.e. their start and end points) conflict with other stomach element's delimiters (such as boxes or paragraphs), they are appropriately closed and subsequently reopened, e.g.:

```
Some paragraph \begingroup \itshape
this is italic \par
New paragraph, still italic \endgroup not
italic anymore
```

Some paragraph *this is italic*

New paragraph, still italic not italic anymore

would yield HTML similar to:

```
<div class="paragraph">
  Some paragraph
  <span style="font-style:italic">
    this is italic
  </span>
</div>
<div class="paragraph">
  <span style="font-style:italic">
    New paragraph, still italic
  </span>
  not italic anymore
</div>
```

In general, the nodes produced by font changes and similar commands are considered "*annotations*": If these nodes have no children, or a single child that modifies the same CSS property, they are discarded or replaced by their only child. If they have a single child or are the only child of their parent node, the corresponding `style`-attribute is attached to the relevant node directly. Only in the remaining case is an actual `<span>` node produced in the output HTML.

To deal with fonts in general, it should be noted that most TeX fonts are freely available in a web-compatible format (e.g. `otf`) online; we *could* consequently use the actual fonts used by TeX in the output PDF. In practice, we prefer to have adequate Unicode characters in the HTML output, rather than ASCII characters representing a position in a font table. Consequently, RusTeX instead hardcodes fonts as pairs of 1. a map from ASCII codes to Unicode strings and, 2. a sequence of font modifiers (e.g. *bold*, *italic*). The former is used to produce actual characters, the latter to choose appropriate CSS attributes as above.

Currently, RusTeX fixes Latin Modern as the font family used, but somewhat nonsensically obtains font metrics the same way as TeX, by processing the `tfm`-files on demand [2], providing only rough approximations of the actual values (in HTML).

2.2 Global Document Setup

At `\begin{document}`, RusTeX determines 1. the current font and its size, 2. the page width (as determined by `\pdfpagewidth`) and 3. the text width (as determined by `\hsize`), and attaches them as corresponding CSS attributes to the `<body>` node – the page width determining the `max-width` and $(\langle page\ width\rangle - \langle text\ width\rangle)/2$ determining the `padding-left` and `padding-right` properties. The latter is important to accomodate e.g. `\marginpar` and related mechanisms, and is discussed more precisely in section 5.

3 Boxes and Dimensions

Clearly, the most important primitives to get "right" are (horizontal or vertical) *boxes*, produced by `\hbox`, `\vbox` and variants (`\vtop`, `\vcenter`), as they are the primary means that more elaborate macros use to achieve their aims. They also serve as a good example of the complexities involved when translating to HTML.

Boxes have five important numerical values that matter with respect to how they are typeset: `width`, `height`, `depth`, `spread` and `to`, which we will discuss shortly.

Horizontal boxes (as produced by `\hbox`) – as the name suggests – have their contents arranged horizontally, and vertical ones vertically. This is nicely analogous to the CSS *flex model*, so naturally, we can associate boxes with CSS flex display values. An entire document can be thought of as a single top-level vertical box. Hence:

```
.hbox, .vbox, .body {
  display: inline-flex;
}
.vbox, .body { flex-direction: column; }
.hbox { flex-direction: row; }
```

An important distinction that matters here is that between the actual *contents* of the box and its *boundary*. Usually, the dimensions of a box are computed from the dimensions of its children – which, conveniently, is analogous to HTML/CSS, so in the typical case we do not need to bother with them at all and leave those up to the rendering engine:

```
.hbox, .vbox, .body {
```

```
  width: min-content;
  height: min-content;
}
```

Whenever possible, we *avoid* precisely assigning dimensional values in HTML and defer to the ones computed by the rendering engine. This is important to account for discrepancies between HTML and TEX, e.g. regarding the precise heights of characters, lines, paragraphs , etc.

However, the dimensions of a box can be changed after the fact, using the `\wd`, `\ht` and `\dp` commands (corresponding to `width`, `height` and `depth`, respectively). If these dimensions are changed, the *contents* and how they are layed out are not changed at all, but the typesetting algorithm, when putting "ink to paper", will proceed *as if* the box had the provided dimensions. This allows macros to layer boxes *on top* of each other; in the (very common) most extreme case by making boxes take up no space at all. For example:

```
\setbox\myregister\hbox{some content}
\wd\myregister=0pt \ht\myregister=0pt
\dp\myregister=0pt
\box\myregister other content
```

This will produce a horizontal box with the content "some content" with all dimensions being 0 from the point of view of the output algorithm, meaning the "other content" following the box will be put directly *on top* of the box, like so:

other content

Hence, we *do* have to occasionally consider the actual (computed or assigned) dimensions of TEX boxes and other elements.

Regarding boxes, we attach actual values for `width`/`height` to their HTML nodes *if and only if* they have been *assigned* fixed values, and let

```
.hbox, .vbox { overflow: visible; }
```

We can then achieve the same effect in HTML via:

```
<div class="hbox" style="width:0;height:0;">
  some content
</div> other content
```

3.1 `width`/`height` vs. `to`

Things get more interesting if the assigned values for the dimensions of the box are *larger* than the actual box contents – this tells us how we need to align the contents of boxes vertically and horizontally. This, however, is also where the `to`-value of a box comes into play:

Setting (exemplary) `\wd`=⟨*val*⟩ for a horizontal box, as mentioned, does not actually impact the way the box *content* is layed out. `\hbox to`=⟨*to-val*⟩`{...}` however *does*, while also setting the `width` of the box: The `to`-attribute instructs TEX to arrange the contents of the box "in line with" the box being ⟨*to-val*⟩ wide,e.g.:

```
\hbox{some box content}
\hbox to \textwidth{some box content}
```

some box content
some box content

This example is deceptive in that it suggests the box contents were evenly spread out across the ⟨*to-val*⟩ of the box, but this is not so. Consider:

```
\hbox to \textwidth{
  \hbox{some}\hbox{box}\hbox{content}
}
\hbox to \textwidth{%
  \hbox{some}\hbox{box}\hbox{content}%
}
```

someboxcontent
someboxcontent

It's not that the individual content elements in the box are spread out evenly; instead, they are left-aligned and *space characters* (and newline characters, which are treated like spaces) behave (roughly) as if followed by `\hfil` – i.e. they take up as much space as they can in the containing `\hbox`. And while subsequently, the box has a width of ⟨*to-val*⟩, changing that with `\wd` is possible:

```
\setbox\myregister\hbox to \textwidth{%
  \hbox{some}\hbox{box}\hbox{content}%
}\wd\myregister=0pt \box\myregister
\setbox\myregister\hbox to \textwidth{%
  some box content%
}\wd\myregister=0pt \box\myregister
```

someboxcontent box content

This distinction between the three values `width`, `to`, and "total width of the box's children" forces us to actually distinguish between a) the box itself (i.e. its contents) with its (potential) `to` value, and b) its "boundary box", i.e. subsequently assigned `width`s and `height`s. The same holds analogously for the `to` value and `height` of a vertical box:

```
.hbox { text-align: left; }
.vbox { justify-content: flex-start; }
.hbox-container, .vbox-container {}
```

where the `.hbox-container`-class is used for *assigned* `width`s and `height`s, and `to` translates to the width of the `.hbox` itself. Making spaces behave as they

should in an \hbox forces us to style them accordingly:

```
.space-in-hbox {
  display: inline-block;
  margin-left: auto;
  margin-right: auto;
}
```

Using this class for spaces (directly) in \hboxes makes the remaining content stretched across the full width of the box, as in the examples above.

Notably, TEX allows for *negative* values in dimensions, which CSS does not. To capture the resulting behaviour, whenever a dimension (exemplary width) is < 0, we set the width CSS property to 0, and attach (in this case) margin-right:⟨*width*⟩ to the HTML node (analogously margin-top for height).

Finally, the spread parameter can be used *instead* of to and *adds* the provided dimension to the computed width/height of the box; e.g. if \hbox{foo} has width 15pt, then \hbox spread 15pt{foo} has width $15 + 15 = 30$pt:

Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.foo Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.

Annoyingly, the only way to accomodate this seems to be to compute the "original" value, add the spread value, and attach that as the final width/height to the node.

3.2 Depth and Rules

So far, we have only considered width and height, but TEX has an additional dimension for boxes that CSS does not: depth, which measures the extent to which a given box extends *below* the baseline of the parent box. Depth is rarely important, or rather, matters primarily when manipulating individual characters, which CSS is currently not capable of for reasons explained later. However, notable not uncommon exceptions are explicitly *assigned* depth values, in particular for \vtop boxes.

To better understand depth, we should turn our attention the the \vrule primitive, which produces a colored box of the provided dimensions:[3]

```
Lorem ...
\vrule width 10pt height 10pt depth 10pt
Lorem ...
```

[3] \hrule is implemented analogously, except for using display:block instead of inline-block.

Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque. Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.

This creates a black box with 10pt width and a total 20pt height, centered at the *baseline* of the current line: extending 10pt *above* the baseline (the height) and 10pt *below* (the depth).[4]

Such a box with the right dimensions can be easily produced using CSS:

```
.vrule {
  display: inline-block;
}
```

The individual s are then provided background, width and height (=height+depth) properties corresponding to the color and the dimensions of the \vrule – in the above example[5]

```
style="background:#000000;height:20pt"
```

The tricky part is ensuring that the box is correctly positioned with respect to the surrounding text (or other elements). As above, the solution is to wrap the .vrule in a .vrule-container with the same height as the inner , and adding margin-bottom:-⟨*depth*⟩ to the inner .vrule. This not only allows for moving the box the specified amount below the baseline, but also makes sure that the "boundary" that the rendering engine computes for positioning elements has the relevant dimensions as well.

If a rule has no explicitly provided width/height, it is computed by TEX to be 0.6pt wide, and a length fitting the current box:

```
\hbox{ \vrule
 \vbox{ \hbox{some} \hrule \hbox{text}}
\vrule }
```

| some
text |

We can easily set the width of the \hrule with width:100% to achieve the same effect. Unfortunately, the same does not work with \vrule and its height in HTML, as an artifact of when and how the heights of boxes are computed by the rendering engine. In those situations, we have to distinguish between paragraphs and \hboxes: In the former case we heuristically set the height to the current font size, in the latter (since we are in a flex box), we can set

[4] Note the gap between the second and third line of text, caused by the depth ob the \vrule.

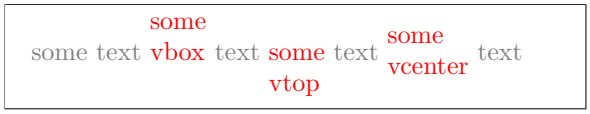
[5] For simplicity's sake, we will use the same dimensions (in pt) in both TEX code and CSS; in actual practice, we scale 1pt in TEX to some value in px units.

`align-self:stretch` to make the rule fit the containing box.

3.3 \vbox vs \vtop vs \vcenter

`\vtop` behaves like `\vbox`, except that where a `\vbox` is vertically aligned at the *bottom* of the parent box's baseline, a `\vtop` is vertically aligned at the top with the surrounding text, extending downwards. `\vcenter` is vertically aligned at the center and is only allowed in math mode:

```
some text \vbox{\hbox{some}\hbox{vbox}}
text \vtop{\hbox{some}\hbox{vtop}}
text $\vcenter{\hbox{some}\hbox{vcenter}}$
text
```

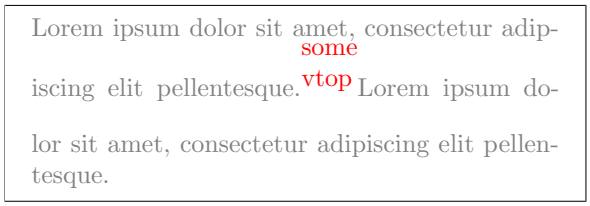


Internally, the three types of vertical boxes differ precisely in their a priori `depth`s and `height`s. As long as these are not subsequently reassigned (using `\ht` and `\dp`), we can achieve the same effect much more accurately by using the `vertical-align` property, that covers the same primary *intent* of the three types of vertical boxes:

```
.vbox{ vertical-align: bottom }
.vtop{ vertical-align: baseline }
.vcenter{ vertical-align: middle }
```

We now need to be careful with changing the *height* of a `\vtop` box, however: Since the primary vertical dimension of a `\vtop` corresponds to its *depth* (below the baseline), *increasing* its height actually corresponds to moving the box contents upwards *without changing the amount of space* it takes up *below* the baseline:[6]

```
Lorem ...
\setbox\myregister\vtop{\hbox{some}\hbox{vtop}}
\ht\myregister=20pt\box\myregister
Lorem ...
```



This can be approximated in HTML by setting both the `margin-top` *and* `bottom` CSS properties of the `.vbox-container` to the value ⟨*height*⟩−⟨*current line height*⟩: The `bottom` property moves the box upwards, while the `margin-top` property makes sure that the boundary box grows acordingly, instead of the moved box overlapping with other elements.[7]

Conversely, if we manipulate the *depth* of a `\vtop`, we can set the `height` of the `.vtop` HTML node itself to ⟨*depth*⟩+⟨*current line height*⟩.

Annoyingly, it now turns out that height/depth manipulations on `\vbox`es and `\vtop`s (respectively) do not play well with `vertical-align` CSS properties within paragraphs – the boxes are not correctly aligned vertically. When explicitly setting these dimensions, it is therefore necessary to, as with `\vrule`, introduce an intermediate HTML node with class `.vbox-height-container` to achieve the effect.

4 Paragraphs

At a first glance, paragraphs in TEX seem largely stright-forward:

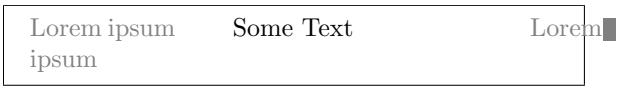
```
.paragraph {
  text-align:justify;
  display: inline-block;
  margin-top: auto;
}
```

The `margin-top:auto` assures that paragraphs are vertically aligned at the bottom of `\vbox`es.

Any horizontal material (text, `\noindent`, `\unhbox`,...) outside of a paragraph or an `\hbox` (and similar constructions) *opens* a new paragraph, and `\par` closes it again.

If we were primarily interested in document semantics without caring about the page layout dictated by TEX, we could be done at this point. However, in TEX, paragraphs have fixed widths dictated by several parameters and commands, including `\hsize`, `\leftskip`, `\rightskip`, `\hangindent` and `\hangafter`, and `\parshape`. This matters when a paragraph is opened inside a `\vbox`. Consider e.g.

```
Lorem ipsum \vbox{Some Text} Lorem ipsum
```



The `Some Text` in the `\vbox` opens a new paragraph, including indentation, and that paragraph has width `\hsize`, regardless of its contents. The `\vbox` itself then inherits the full width of the containing paragraph.[8]

[6] Again, note how the three lines in the paragraph are pushed apart by the unchanged depth and new height of the box

[7] The same idea is used for `\raise/\lower`.

[8] Here, we set `\hsize` to a smaller value to attempt to demonstrate the effect without breaking the layouting of this very document too much.

Approximating this behaviour (in the absence of dedicated macro support) matters, for example to accomodate `\begin{minipage}`s, `tcolorbox` and similar packages. This is also one instance where TEX is significantly more flexible than HTML/CSS: `\hangindent` and `\parshape` do not have CSS equivalents. While in principle in might be possible to "emulate" them using empty `<div>` nodes with `float` attributes, we currently ignore them and proceed as if the whole paragraph were typeset according to the rules applying to the last line; e.g. the last entry in the `\parshape` list.

The relevant parameters can subsequently be condensed into three attributes, in the simplest case computed thusly: 1. the actual width of the text (`\hspace`−(`\leftskip`+`\rightskip`)), and 2. left and right margins (`\leftskip` and `\rightskip`), which we translate to the CSS attributes `min-width`, `margin-left` and `margin-right`, respectively.

Notably, to accomodate macros that make use of computed dimensions of various boxes, we need to approximate TEX's line breaking algorithm to make sure that the computed heights of paragraphs are reasonably accurate.

5 Responsiveness and Relative Widths

The above suggests, that we need to hardcode the absolute widths of both the document as a whole (in the sense of `\textwidth`/`\pagewidth`) as well as the widths or paragraphs and `\hbox`es. This is of course undesirable in that it destroys responsive layout in HTML. Ideally, we would prefer to use *relative* widths in terms of percentages.

Regarding the document width, this is easily resolved: Instead of letting `width`:⟨*text width*⟩, we set `max-width`:⟨*text width*⟩. This way, the page accomodates smaller screen, but if enough screen space is available will default to the size the document was originally designed for.

Relative widths in general however only work as expected if the direct parent of a node has a fixed assigned width, and as previously mentioned, in as far as possible we want to defer the precise dimensions of HTML nodes to the rendering engine. Moreover, once we have a box width `width:0`, no percentage will get us back to a non-zero value. Both problems were solvable if CSS would allow for inheriting attribute values from arbitrary ancestors, but since it does not, we need to be more creative:

Instead of directly inheriting, we can use a *custom* CSS property `--current-width` and initialize it as `--current-width:min(100vw,`⟨*text width*⟩`);` `width:var(--current-width)` in the body. This achieves the same effect as the more naive approach above, but now allows for stating other widths in the body of the HTML node as values relative to the `--current-width` attribute.

Using this approach, all relative widths in a document are now relative to the *current document's* initial `\textwidth`. This is problematic in the context of STEX, where the `\inputref` macro largely replaces TEX's `\input`: Besides allowing for referencing source files relative to a math archive (i.e. a "library" of document snippets), which is important for building modular libraries, when converting to HTML `\inputref` simply inserts a reference to the file, that can subsequently be dynamically inserted into the referencing document. This obviates the need to both reprocess the same file for every context in which it occurs, as well as to rebuild all referencing files every time any of the `\inputref`ed files change. Notably, such `\inputref`s often occur deeply nested, e.g. a file with a short individual definition might be `\inputref`ed in an `\begin{itemize}` environment in a definition block in a framed beamer slide within lecture notes. This entails that we would like to inherit widths from *the closest ancestor with a fixed assigned width* > 0 (e.g. the innermost `\item` in the example above) rather than the `<body>`, and update the value of `--current-width` accordingly, to accomodate any document context in which the HTML node might (dynamically) occur.

Hence, when encountering e.g. a (top-level) `\vbox` with width `0.5\textwidth` (e.g. a `\begin{minipage}`), we would like to do:

```
<div class="vbox" style="--current-width:calc(
  0.5 * var(--current-width));
  width:var(--current-width)">...</div>
```

Unfortunately, CSS does not allow for self-referential attribute updates; so we have to use an intermediary custom attribute `--temp-width` and an inner `<span>` to do the following:

```
<div class="vbox" style="--temp-width:calc(
    0.5 * var(--current-width));
    width:var(--temp-width)">
  <span style="display:contents;
    --current-width:var(--temp-width)
  ">...</span></div>
```

to achieve the desired effect. While this is ugly from an implementation point of view, it allows for variable viewport widths and solves the problem with inheriting widths *through* boxes of size 0.

6 Skips and Text Alignment

In section 4, we acted as if `\leftskip` and `\rightskip` where simple dimensions – i.e. values of unit `pt`.

Skips actually have three components: A base dimension, an (optional) *stretch* factor, and an (optional) *shrink* factor. A skip represents a (horizontal or vertical) space that is *ideally* ⟨*base dimension*⟩ wide/high, but can stretch or shrink according to the other two components to fit the current page layout. Stretch and shrink factors have one of four units `pt`, `fil`, `fill` or `filll`, the latter three representing "increasingly infinite" stretch/shrink factors.

Skips are used to introduce vertical or horizontal space, using the `\hskip` and `\vskip` commands. Focusing solely on their base dimensions for now, both can be represented as empty `<div>` nodes with corresponding `margin-left` or `margin-bottom` values, respectively. Conveniently, this works with both positive and negative base dimensions, and we can use the same mechanism for `\kern`, which for all practical purposes behaves like `\hskip` or `\vskip` with zero stretch/shrink. This allows us to cover both of the following cases:

```
\noindent some text \hskip20pt some text\par
\noindent some text \hskip-20pt some text
```

some text some text
some text

If we add a strech factor, we can e.g. achieve the following:

```
\noindent some text \hskip20pt plus 1filll
some text\par
```

some text Some text

Unfortunately, CSS has no analogue for stretch and shrink factors. For *shrink*, this largely causes no serious issues. *Stretch* factors however are primarily used to achieve (primarily horizontal) *alignment*. Left-aligned, centered, or right-aligned content is achieved in TEX by inserting corresponding skips; so the best we can do is to represent skips as the CSS `text-align` property:

If `\leftskip` or `\rightskip` have stretch factors, we compare them and set the alignment for the paragraph accordingly. For `\hbox`, we need to inspect the contents of the box for initial and terminal occurrences of relevant skips, compare them, and derive the intended alignment depending on which is "bigger".

Additionally, we can add `margin-left:auto` to the `<div>`s corresponding to skips iff they have a stretch factor of (at least) `1fil`; however, this only works in `\hbox`es (not in paragraphs), and does not necessarily behave right in conjunction with other skips. Thankfully, text alignment seems to be the primary regularly occuring situation where skips are noticable and important to represent accurately in the HTML, which this heuristical approach seems to cover reasonably well – while discrepancies between PDF and HTML can be easily found, they are usually not severe.

7 Math Mode

For stomach elements in math mode, we naturally use Presentation MathML. Translating the relevant primitives to MathML is largely straight forward and covered elsewhere [9], with the slight "modernization" that we prefer CSS over MathML attributes. Since the font used for MathML depends on the rendering engine, and some of them are rather unsatisfactory (e.g. vanilla Firefox under Ubuntu), we can explicitly set the font to `Latin Modern Math` for a more unifying look. Skips and kerns are implemented as above, but using `<mspace>` nodes instead of `<div>`.

Regarding font sizes, we can either defer to the rendering engine or leave that up to TEX– in which case we need to make sure that we override the CSS rules imposed by the rendering engine via:

```
msub > :nth-child(2), msup > :nth-child(2),
mfrac > * , mover > :not(:first-child),
munder > :not(:first-child) {font-size:inherit}
```

More pressingly however, occurrences of `\hbox` or `\vbox` in math mode require us to "escape" back to HTML in `<math>` elements. While not officially supported, using `<mtext>` nodes for that works well in both Firefox and Chromium (and with some hacking with MathJax). However, when doing so, various CSS properties are inherited from those set by the default stylesheet for MathML. Hence, whenever we escape back to horizontal or vertical mode, we explicitly insert the parameters of the current text font, and set:

```
mtext {
  letter-spacing: initial;
  word-spacing: initial;
  display: inline-flex;
}
```

As mentioned in [9], spacing around operators (i.e. `<mo>` nodes) is governed by an operator dictionary. The spacing rules are in principle well-chosen and best left to the rendering engine. TEX can change these however, using the commands `\mathop`, `\mathbin`, etc.

To accomodate this functionality, we can explicitly set left and right padding based on TEX's math character class, and set:

```
mo {padding-left: 0;padding-right: 0}
```

Notably, this works (as of May 2023) in Firefox, but not in Chromium-based browsers[9], where the spacing determined by the operator dictionaries is effectively a *minimum* that can not be reduced further.

Changing these spacing factors can occasionally be important when composing symbols from more primitive ones. For example, the `\Longrightarrow` macro ⟹ concatenates the symbols = and ⇒ with a negative `\kern` between them - in which case unintended spacing between the two symbols can break the intended result.

8 \halign

The `\halign` command is the primary means LaTeX packages use to layout *tables*, and not surprisingly, its closest correspondants in HTML are `<table>` nodes. However, as with text alignment, effects that in HTML are achieved via attributes of the parent node (`<table>`, `<tr>` or `<td>`) are achieved in TeX via content elements *in* the individual cells – or between them: Where a table in HTML is exactly a sequence of rows consisting of cells, in TeX, the `\noalign` command allows for inserting vertical material *between* rows, which is used to insert horizontal lines (e.g. `\hline`) or determine the spacing between rows. Borders and spacing between cells are achieved via `\vrule`s and skips.

Hence, we have to face two major problems when translating `\halign`s to `<table>`s:

1. If we want to accomodate spacing, text alignments and borders, we need to "parse" the contents of cells and `\noalign` blocks to determine which CSS attributes to attach to the `<table>`, `<tr>` and `<td>` nodes. This is worsened by the fact that the margin attributes on `<td>` and `<tr>` nodes have no actual effect.
2. The *height* of a `<tr>` is computed from the *actual* height of its children, and even enclosing a whole cell in a `<div>` with `height:0` does not change the actual height of the relevant `<tr>`.

While the former problem is inconvenient but solvable, the latter becomes severe if we consider less obvious situations that `\halign` is used for: For example, the `\forkindep` macro mentioned above (i.e. ⫛) uses `\ooalign` to combine the two characters | and ⌣, which uses an `\halign` to superimpose them, forcing us to make the rows narrower than `<tr>`s allow for.

Therefore we use the CSS grid model for `\halign` rather than the (seemingly more adequate) `<table>`:

```
.halign {
  display:inline-grid;
  width: fit-content;
  grid-auto-rows: auto;
}
```

with cells being styled like `.hbox` with the additional attributes `height:100%;width:100%`, and any `\halign` with n columns being given the additional CSS attribute `grid-template-columns:repeat(`n`,1fr)`. This aligns the individual cells almost exactly like `<table>` would, but gives us the more control over their intended heights. `\noalign` vertical material can now be inserted in a `.vbox` `<div>` with `grid-column:span` n. Notably, this entirely obviates the need to implement special rules for visible borders or spacing between rows/columns: The existing treatment for `\vrule`/`\hrule` and skips produces (almost universally) the desired output out of the box.

Notably, empty cells in `\halign` are not actually empty. Consider:

```
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ab
cd

ef

Note that the third row really is entirely empty, with no spacing involved. Instead, we get a row that has roughly the same height as the other three. We can remedy this effect via:

```
\baselineskip=0pt\relax
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ab
cd
ef

or do even more ridiculous things:

```
\baselineskip=0pt\relax
\lineskiplimit=-100pt\relax
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ab

This entails, that we now need to take `\baselineskip` and `\lineskiplimit` into account and use them to compute `min-height` (for `\baselineskip`) or `height` values (in case of sufficiently negative `\lineskiplimit` values) for the cell's HTML node.

9 Limitations

This brings us to the first insurmountable difference between TeX and CSS: *lines*. A line of text in

[9] conversely, scaling brackets properly with `stretchy="true"` seems to not work in Firefox as of yet.

TEX consists of individual character boxes with individual heights, widths and depths, and the spacing between lines is governed by the three parameters `\baselineskip` (the "default" distance between two baselines), `\lineskiplimit` (the minimally allowed distnce between the bottom of a line and the top of the subsequent one), and `\lineskip` (the minimal skip to insert between two lines, if their distance is below the `\lineskiplimit`). In particular, the height of a horizontal box containing e.g. a single character is entirely determined by the height of that particular character.

In contrast, a line of text in HTML/CSS has a *fixed* height of the current `line-height` value regardless of the occurring characters – and every single character counts as a "line": for every character, a *leading* space is inserted on top of it to make the containing box adhere to the `line-height`. This makes box manipulation on the level of individual characters currently (almost) impossible.[10]

One striking example for this is the `\LaTeX` macro, where the `A` is enclosed in a `\vbox`. RUSTEX replaces its expansion by a simple `\raise\hbox` to achieve the (almost) same effect.

Situations where layouting critically depends on very precise positioning and sizing of boxes remains tricky. This is the case for example with the `tikzcd` package, where the nodes are layout as tables, with `pgf` arrows between the individual cells.

Various macros make use of LATEX floats in non-trivial ways, such as `\marginpar` and the `\begin{wrapfig}` environment, making special treatment for them (as of yet) unavoidable.

The `xy` package is a clear example of where, due to its usage of custom fonts, there is currently no feasible way to achieve support in terms of TEX primitives alone; anecdotally, I have been told that a `pgf` driver for `xy` is in the works, which, if completed, would likely immediately work for RUSTEX as well.

10 Conclusion

Despite the limitations mentioned above, the schema presented here works surprisingly well in a variety of cases. For example, list environments (`\begin{itemize}`, `\begin{enumerate}`, etc.), `\begin{lstlisting}`, figures, `\begin{algorithmic}`, `tcolorbox`, various environments for definitions, theorems and examples, `bibtex` and `biblatex`, and many other macros, environments and packages with often intricate options and configurations, work out of the box without special treatment and with the expected presentation in the HTML.

Indeed, it is certainly surprising how much can be achieved without providing dedicated implementations for non-primitive macros, to the point where I am nowadays more surprised if the schema *fails* than when it *succeeds*.

To mention one particular highlight: A tongue-in-cheek paper was published in May 2023 on `arxiv.org` that argued for solving the order-of-authors problem in scientific publishing by *overlaying all the author names on top of each other*, including instructions how to achieve that in both TEX and HTML [1].

Running RUSTEX over the LATEX sources for the paper produced the right layout directly (see Figure 1).

PDF:

To compensate for alphabetical discrimination, several specific papers have explored alternate mechanisms for deciding authorship order, as documented in a footnote. These mechanisms include competition via 25-game croquet series (Russell 1974), 2-day backgammon contest (Hadley 1977), tennis match (Griffiths 1978), basketball free throws (Reichartis 1991), arm wrestling (Birmingham 1995), brownie bake-off (Young 1992), a game of chicken (Reichenstein 1983), or rock paper scissors (Wolfling 2004); by coin toss (Millard 1992), dice roll (Mayfield 2011), the outcome of famous cricket games (O'Hara 2010), currency exchange rate fluctuation (Mitchell-Olds 2003), or dog treat consumption order (Pudlitz 2019); or by authors' height (Woodward 2005), fertility (Babcock 1992), proximity to tenure (Gillespie 1998), reverse alphabetical order (Messenger 2006), or degree of belief in the paper's thesis (Chalkmers 1998). Others have proposed games such as Russian roulette ("publish and perish") (Purvis 2016). See the excellent surveys (Duffy 2016; Deville 2014; Obscura 2014) and their comments.

HTML:

To compensate for alphabetical discrimination, several specific papers have explored alternate mechanisms for deciding authorship order, as documented in a footnote. These mechanisms include competition via 25-game croquet series (Russell 1974), 2-day backgammon contest (Hadley 1977), tennis match (Griffiths 1978), basketball free throws (Reichartis 1991), arm wrestling (Birmingham 1995), brownie bake-off (Young 1992), a game of chicken (Reichenstein 1983), or rock paper scissors (Wolfling 2004); by coin toss (Millard 1992), dice roll (Mayfield 2011), the outcome of famous cricket games (O'Hara 2010), currency exchange rate fluctuation (Mitchell-Olds 2003), or dog treat consumption order (Pudlitz 2019); or by authors' height (Woodward 2005), fertility (Babcock 1992), proximity to tenure (Gillespie 1998), reverse alphabetical order (Messenger 2006), or degree of belief in the paper's thesis (Chalkmers 1998). Others have proposed games such as Russian roulette ("publish and perish") (Purvis 2016). See the excellent surveys (Duffy 2016; Deville 2014; Obscura 2014) and their comments.

Figure 1: Screenshots from [1] in PDF and RUSTEX generated HTML

[10] A proposal to the W3C CSS WG regarding leading space, which would presumably help here, has been open since 2018: `github.com/w3c/csswg-drafts/issues/3240`

The most important aspect for generating adequate (and often great) HTML seems to be the "proper" treatment of `\hbox`/`\vbox`, `\hrule`/`\vrule` and skips/kerns, which RusTeX implements as described here. Their treatment should be relatively easy adaptable to, and usable by, other HTML converters as well, where "PDF-like" HTML output is desirable.

The most dire *limitations* are often related to intrinsic limitations of CSS– presumably, any extension of CSS that allows for more fine-grained control, especially on the character level, would allow for even better translations from TeX.

Future Work Naturally, some of the techniques described here have been slightly simplified and are augmented in practice via various heuristics that are still subject to experimentation and improvements. Other discrepancies or problems are usually addressed (if possible) as we become aware of them (which still happens regularly).

Acknowledgements The presented research is part of the VoLL-KI project, supported by the Bundesministerium für Bildung und Forschung (BMBF) under grant 16DHBKI089.

References

[1] Erik D. Demaine and Martin L. Demaine. *Every Author as First Author*. 2023. arXiv: 2304.01393 [cs.DL].

[2] David Fuchs. "TeX Font Metric files". In: *Communications of the TeX Users Group (TUGboat)*. Vol. 2. 1. 1981, pp. 53–61. URL: https://www.tug.org/TUGboat/tb02-1/tb02fuchstfm.pdf.

[3] Donald E. Knuth. *The TeXbook*. Addison Wesley, 1984.

[4] Michael Kohlhase and Dennis Müller. *The sTeX3 Package Collection*. Tech. rep. URL: https://github.com/slatex/sTeX/blob/main/doc/stex-doc.pdf (visited on 04/09/2023).

[5] *MathJax: Beautiful Math in all Browsers*. http://mathjax.org. URL: http://mathjax.com.

[6] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/ (visited on 03/12/2021).

[7] Dennis Müller. "Mathematical Knowledge Management Across Formal Libraries". PhD thesis. Informatics, FAU Erlangen-Nürnberg, Dec. 2019. URL: https://opus4.kobv.de/opus4-fau/files/12359/thesis.pdf.

[8] Dennis Müller and Michael Kohlhase. "sTeX3 – A LaTeX-based Ecosystem for Semantic/Active Mathematical Documents". In: *TUGboat; TUG 2022 Conference Proceedings*. Ed. by Karl Berry. Vol. 43. 2. 2022, pp. 197–201. URL: https://kwarc.info/people/dmueller/pubs/tug22.pdf.

[9] Luca Padovani. "MathML Formatting with TeX Rules, TeX Fonts, and TeX Quality". In: *Communications of the TeX Users Group (TUGboat)*. Vol. 24. 1. 2003, pp. 53–61. URL: https://tug.org/tugboat/tb24-1/padovani.pdf.

[10] *Pandoc – a universal document converter*. https://pandoc.org/. 2023.

[11] *sLaTeX/RusTeX*. URL: https://github.com/sLaTeX/RusTeX (visited on 04/22/2022).

[12] *TeX4ht*. https://tug.org/tex4ht/. URL: https://tug.org/tex4ht/.

Primo — The new sustainable solution for publishing

Rishikesan Nair T, Apu V and Hàn Thế Thành, Jan Vaněk

Abstract

Primo is a cutting-edge, cloud-based authoring, submission, and proofing framework that provides a sustainable solution for academic publishing. It combines the advantages of XML-based workflows that facilitate controlled authoring and/or editing in accordance with specific DTDs and house styles, with the visually appealing and mathematically precise typesetting language of TEX, enabling the creation of high-quality PDFs and mathematical images (offering an alternative to MathML coding).

By speaking the widely accepted communicating lingua of mathematics and science (i.e., TEX), and utilizing the XML/MathML format for archiving, Primo has the potential to revolutionize the publishing industry. This tool caters to both the author and the publisher, bringing their needs together with enhanced participation of authors in the publishing process. The three main modules of Primo include Authoring, Submission/Reviewing, and Proofing, all of which are equipped with usability checks during submission, a collaborative editing feature, a WYSIWYG math editing tool, and publisher/journal-based PDF manuscript rendering. With Primo, authors can be assured that their work will be published with the highest level of precision and quality.

1 Introduction

Primo, the latest addition to the lineup of TEX-based tools, is developed by STM Software Engineering Pvt Ltd. who is a specialized TEX typesetting house renowned for its top-notch typesetting and pre-press services, catering to the needs of STM publishing giants specialized in the complex article typesetting. With its state-of-the-art technologies, STM Software Engineering Pvt Ltd. developed a range of cloud-based typesetting frameworks, including TEXFolio [1] and Ithal [2], primarily designed for in-house typesetting and format conversion purposes within publishing houses. On the other hand, Neptune [3] and Primo target authors directly, providing them with efficient and user-friendly TEX-based tools.

2 Primo

Primo's modular structure and well-designed tools enable authors to navigate the entire publication journey with ease, from initial authoring to final proofing. By integrating these three modules, Primo optimizes the authoring, submission, and proofing processes, making it a comprehensive and efficient platform for scholarly publishing.

2.1 Authoring tool

An intriguing offshoot of the aforementioned processes is a stand alone authoring tool, codenamed Primo Editor. This tool encompasses all the necessary elements to effortlessly compose an article that meets all technical requirements for seamless uploading to a publisher.

2.1.1 Salient features

Please note that while the below list covers the main features, there are undoubtedly additional features yet to be mentioned that further enhance the functionality and user experience of the tool.

1. **Collaborative Editing:** Multiple authors can contribute simultaneously.
2. **Operating System Independence:** The tool shall be operating system independent and will have a WYSIWYG interface.
3. **Cloud-Based with TeX Installation:** A completely cloud-based version with a TEX installation comprising of essential packages, fonts, compiler and utilities. The user need not worry about installation or setting up of a TEX distro in their local system.
4. **Proper template:** With the Primo authoring tool, authors no longer need to spend time searching for the appropriate template for their submissions. The tool offers a comprehensive collection of templates, and by simply providing the name of the journal and publisher, it automatically selects the most suitable template. This eliminates the hassle of manually locating the correct template and ensures that authors can focus on their content without the added burden of formatting.
5. **Math Input:** Users can enter mathematical equations using LATEX syntax or utilize built-in math tools from the menu.
6. **Form-Like Interface:** A user-friendly form-like interface is available to capture front matter data such as author information, affiliations, abstract, keywords, graphical or stereo-chemical abstracts, and more.
7. **WYSIWYG Interfaces for Tables and Figures:** WYSIWYG interfaces for entry of table and figure data.

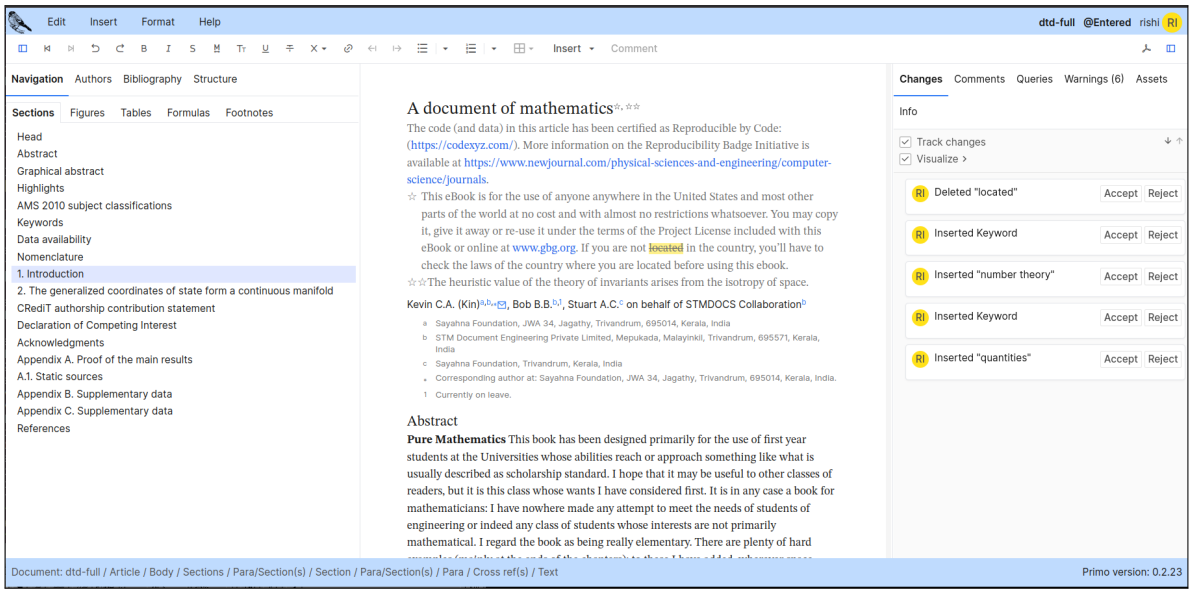


Figure 1: Primo: The main page.

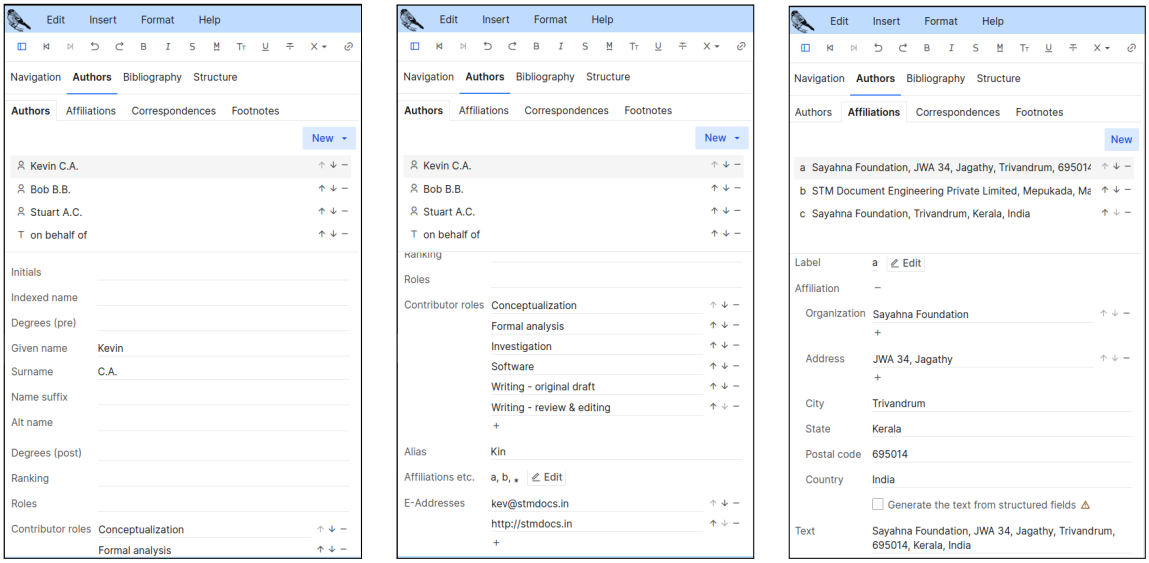


Figure 2: The author and affiliation field.

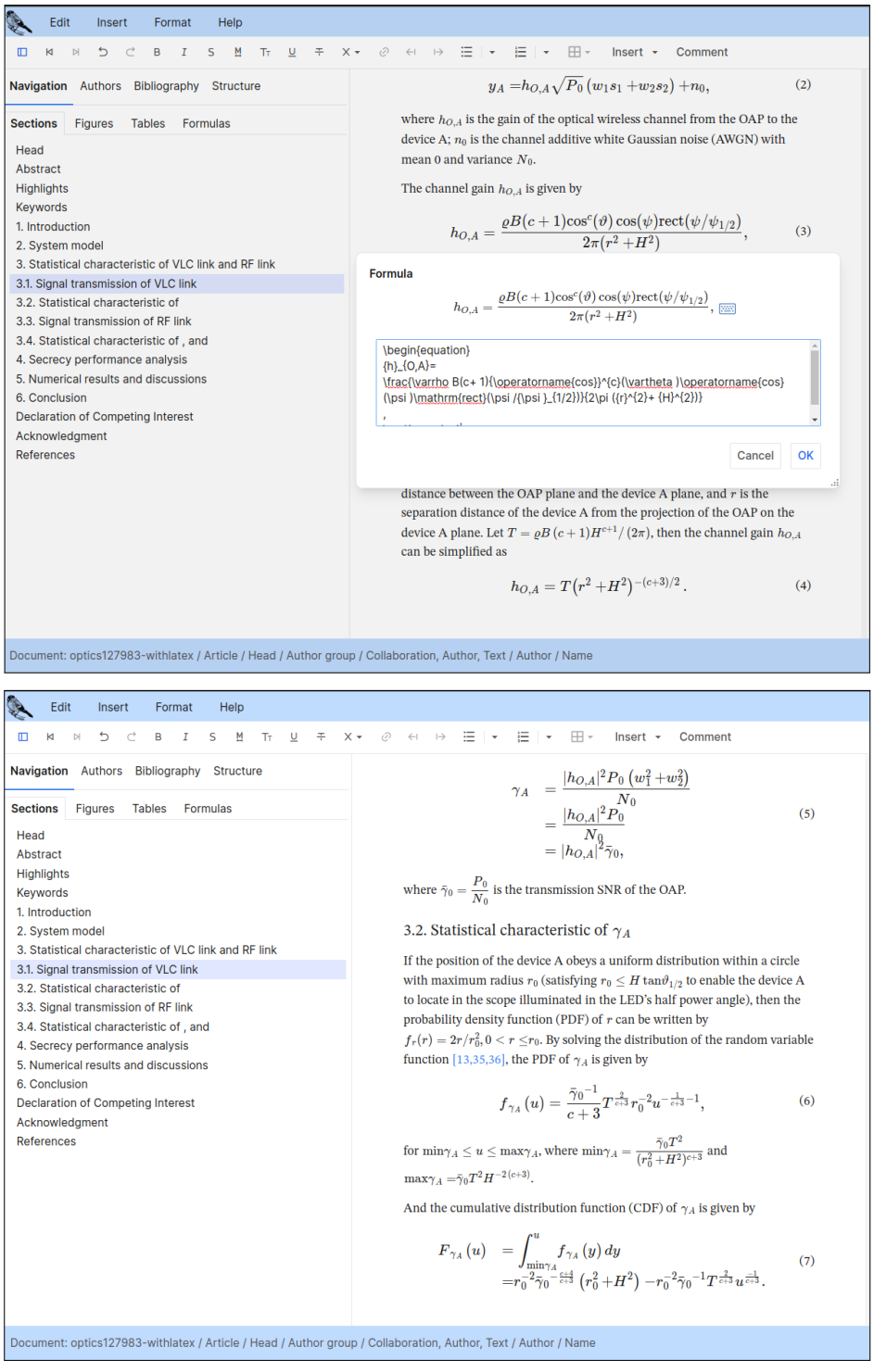


Figure 3: Math content rendering.

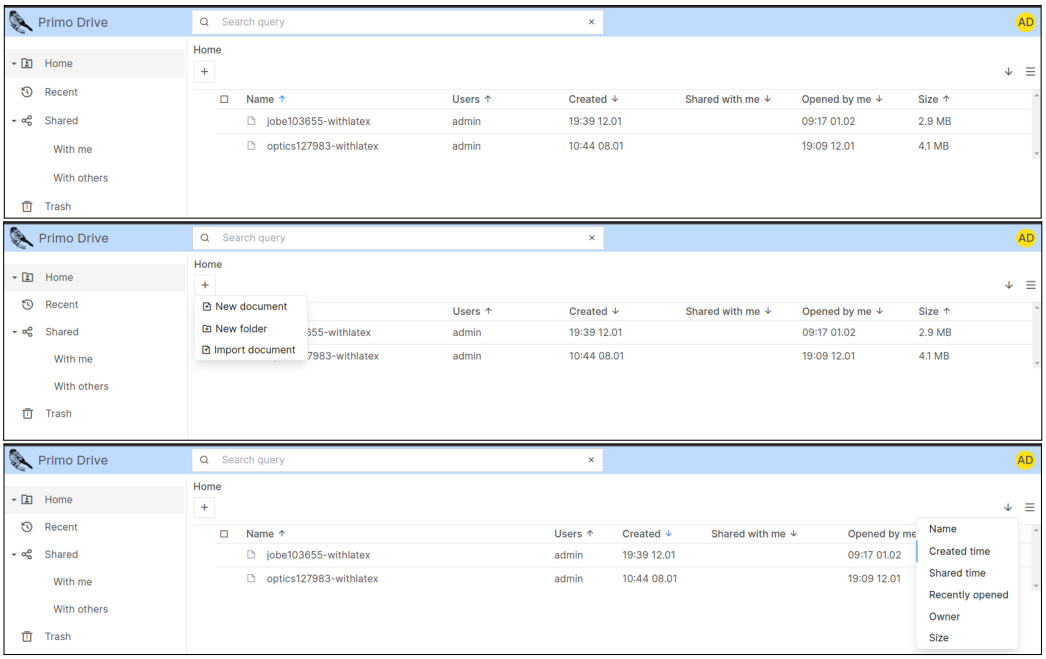


Figure 4: The primo drive.

8. **BibTeX Support:** BibTEX data is always welcome if the user prefers to use the same.
9. **Bibliography Data Checking:** Checking bibliography data with Cross-ref is an added benefit.
10. **Bibliography Import:** With the help of Primo, users can easily import bibliography data using identifiers such as DOIs (Digital Object Identifiers) or PMID (PubMed IDs), streamlining the referencing process.
11. **Enhanced Author Participation:** The tool promotes active author involvement in the publishing process, minimizing errors and semantic inconsistencies introduced by typesetters. This enhances the overall quality and reduces the time gap between submission and publication.
12. **Compliance Checking:** The tool automatically checks the manuscript's adherence to the specific style guidelines of the publisher or journal, ensuring compliance with formatting requirements.

These features collectively provide a comprehensive and user-friendly platform for collaborative manuscript preparation, improving the efficiency and quality of the publishing process.

2.2 Submission tool

The submission process for authors can often be arduous and time-consuming. With strict timelines and numerous procedures to navigate, authors often find it challenging and frustrating. Primo seeks to alleviate these difficulties faced by the author community. In its initial stage, this authoring tool assists authors in crafting their manuscripts in compliance with the specific style requirements of publishers and journals. The subsequent step involves a seamless transfer of the source files to the submission system employed by publishers.

2.2.1 Salient features

1. **Source files:** Since the source files were already prepared as per the specification using the authoring tool, there will not be any surprises in the submission system.
2. **Proper submission:** Transferring files directly from author tool to submission tool and finally to the publishers' submission system helps to eliminate chances for any missing files or materials.
3. **File category:** In the popular submission systems, we have to select the file type of each source file that we upload. For example, "Manuscript", "Revised Manuscript", "Figure", etc. Primo tool helps to sort this out easily and helps authors to identify which is which.
4. **Usability check:** Depending on the compliance of the submission system to which finally the source files are uploaded for the publisher,

the Primo submission tool runs a custom-made usability check on the source files and reports problems any problems with the source files. This will help those authors who prepare manuscript in their system or any other interface and directly upload source files to the Primo submission tool.

2.3 Proofing tool

The proofing tool plays a crucial role in Primo. Once the typesetting process is complete, the typesetter uploads or imports a dataset that includes the article's XML/MathML, figures, supplementary materials such as audio, video, program codes, and alternative images for MathML, among others.

The format of the dataset is simple and it looks like:

```
Archive:  ENDEND_99996.zip
  Name
  ----
  ENDEND_99996.pdf
  main.assets/
  main.assets/fx1006.jpg
  main.assets/gr1.jpg
  main.assets/fx999.jpg
  main.assets/gr2b.jpg
  main.assets/mmc1.pdf
  main.assets/fx1001.jpg
  main.assets/fx1004.jpg
  main.assets/fx1.jpg
  main.assets/fx1005.jpg
  main.assets/fx1002.jpg
  main.assets/gr2a.jpg
  main.assets/fx1003.jpg
  main.assets/fx1007.jpg
  main.pdf
  main.stripin/
  main.stripin/si33.svg
  main.stripin/si121.svg
  main.stripin/si165.svg
 ..................
 ..................
 ..................
```

2.3.1 Salient features

1. **Track changes:** This facility helps the authors to understand the changes made by the typesetters in their document. The accept and reject buttons can be used to accept or reject the changes made by the typesetters. The visualize mode in the track changes further has two features. They are to visualize the changes made by the authors and to visualize the changes made by the copy-editors or co-authors.
2. **Comment:** If authors are unsure about how a change is to be done, then the comment facility can be used to put a comment.
3. **Queries:** Queries raised by the typesetters or copy-editors are available in the query panel. Authors can provide reply to the queries just below the queries.
4. **DTD compliant:** All the features allowed by the DTD can be used. For example, they cannot insert a figure in the author field since the DTD does not support that.
5. **Changing the order:** Order of author names or position of given and surname or just the content can be swapped.
6. **List:** Changing the formats of the list counters or when inserting a new list, selecting the format of the counters is just easy by using the drop-down menu facility.
7. **Intelligent insertions:** Since the tool follows the DTD faithfully, the insert menu will show the items according to the content model only. So if you are in a paragraph, the drop-down list will show sections, math (both inline and display), list, display quote, etc. However, if you are in the reference section, the insert menu will show bib-entry, bibliography section, etc.

3 Technologies behind Primo

Primo is written mostly in Scala, both server-side and client-side. The client-side is compiled using Scala-JS to JavaScript. On the server-side, Scala is compiled to Java byte-code and runs in the JVM. It can seamlessly inter-operate with existing Java libraries. The development environment is Intellij IDEA, the build-tool is SBT. Primo uses its own widget library called VDL, part of the Primo code base.

Primo doesn't have many external dependencies. We use following "major" libraries:

- JDK obviously
- undertow - the web-server, like tomcat, but smaller
- sqlite - for the DB
- lucene - full-text index of the documents

And some "minor" libraries:

- xpp3 - XML parser
- scala-js DOM, java-time, java-logging
- boopickle, scala-css, and some others

References

[1] Rishikesan Nair T., Rajagopal C.V., and Radhakrishnan C.V. TEXFolio — a framework to typeset XML documents using TEX. TUGBoat

40(2), 147–149. 2019.
https://tug.org/TUGboat/tb40-2/tb125rishi-texfolio.pdf

[2] https://ithal.io/main.html

[3] Aravind Rajendran, Rishikesan Nair T., and Rajagopal C.V. NEPTUNE — a proofing framework for LaTeX authors. TUGBoat 40(2), 150–152, 2019.
https://tug.org/TUGboat/tb40-2/tb125rajendran-neptune.pdf

◇ Rishikesan Nair T, Apu V
STM Document Engineering Pvt Ltd.,
River Valley Campus,
Mepukada, Malayinkizh,
Trivandrum 695571,
India
rishi (at) stmdocs dot in, apu.v (at) stmdocs dot in
https://stmdocs.org

◇ Hàn Thế Thành, Jan Vaněk
Trivic s.ro., Družstevni 161, 763 15 Slušovice, Czech Republic
thanh (at) trivic dot io, jan (at) trivic dot io
https://trivic.io

Curvature combs and harmonized paths in METAPOST

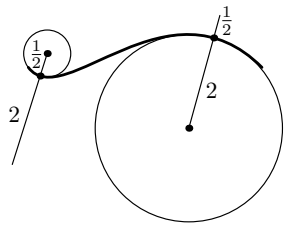
Linus Romer

Abstract

Most font editors offer curvature related tools. One of these tools is the visualization of curvature via *curvature combs*. Another tool is the so-called *harmonization*, which makes the curvature continuous along paths. An implementation of both tools in METAPOST will be presented. Curvature optimized paths already play a significant role in METAFONT and METAPOST and therefore some exemplary METAPOST paths will be examined for their curvature behavior.

1 Curvature

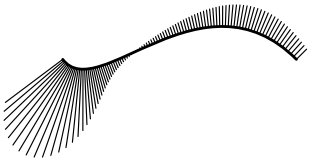
The curvature in a point of a curve is the inverse of the radius of the osculating circle at this point (depicted here as a "curvature vector" on the opposite side of the radius):



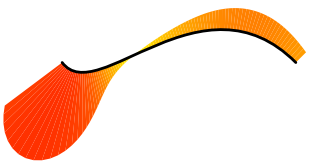
For straight segments, the curvature is constantly zero, since the radius of the osculating circle is infinitely large. Vice versa, the curvature becomes infinitely large when the radius of the osculating circle tends to zero.

2 Curvature combs in METAPOST

We can assemble these curvature vectors into a curvature comb:



The curvature may be additionally mapped to a color and the gaps may be filled:



A figure as above can be achieved by

`path p; p =` ⟨*path*⟩ `; comb(p,300); draw p;`

using the `comb` macro that will be presented here (the `300` scales the curvature comb). We start by defining the macro `crossprod` that returns the cross product between two given vectors $\vec{w}$ and $\vec{z}$:

```
primarydef w crossprod z =
 (xpart w * ypart z - ypart w * xpart z)
enddef;
```

Then the macro `curv` shall be applied to a path `p` in order to return a "curvature vector" that is orthogonal to the path in its initial point. The length of the vector is proportional to the initial curvature of the path:

```
vardef curv expr p =
 save v,w,l; pair v,w;
 v = direction 0 of p;
 l = length v;
 v := v/l;
 w = (point 0 of p - 2*postcontrol 0 of p
                   + precontrol 1 of p)/l;
 2/3*(v crossprod w)/l*(v rotated -90)
enddef;
```

Here is the math behind this macro. A cubic Bézier segment can be described by:

$$\begin{pmatrix}x(t)\\y(t)\end{pmatrix} = t^3(3\vec{Q}-\vec{P}+\vec{S}-3\vec{R}) + 3t^2(\vec{P}-2\vec{Q}+\vec{R}) + 3t(\vec{Q}-\vec{P}) + \vec{P}$$

The initial derivatives are then:

$$\begin{pmatrix}\dot{x}(0)\\\dot{y}(0)\end{pmatrix} = 3\underbrace{(\vec{Q}-\vec{P})}_{=:\vec{v}} \qquad \begin{pmatrix}\ddot{x}(0)\\\ddot{y}(0)\end{pmatrix} = 6\underbrace{(\vec{P}-2\vec{Q}+\vec{R})}_{=:\vec{w}}$$

The signed curvature is calculated by $\dfrac{\begin{pmatrix}\dot{x}\\\dot{y}\end{pmatrix} \times \begin{pmatrix}\ddot{x}\\\ddot{y}\end{pmatrix}}{\left|\begin{pmatrix}\dot{x}\\\dot{y}\end{pmatrix}\right|^3}$.

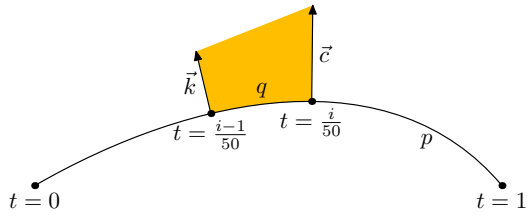
Using $l := |\vec{v}|$ we finally have the formula used in the macro for the initial curvature:

$$\frac{3\vec{v}\times 6\vec{w}}{(3l)^3} = \frac{2}{3}\frac{\vec{v}\times\vec{w}}{l^3} = \frac{2}{3l}\cdot\left(\frac{1}{l}\vec{v}\times\frac{1}{l}\vec{w}\right)$$

The divisions by l are necessary to prevent arithmetic overflows. The special case $\left|\begin{pmatrix}\dot{x}(0)\\\dot{y}(0)\end{pmatrix}\right| = 0$ is not handled here. The curvature then would diverge to $\pm\infty$ (or be 0 if the cubic Bézier segment is a line segment).

Now we define the curvature comb macro of a path p by subdividing each segment in 50 parts and filling an area over each part. Each part of the comb is made of two subsequent "curvature" vectors $\vec{k}$, $\vec{c}$ that are scaled by a constant factor s given by the user. The color depends on the average length of them.

xnotdoi.org/10.47397/tb/0-0/curvaturecombharmonize




```
vardef comb(expr p,s) =
 save q,c,k; path q; pair c,k;
 for n = 0 upto length(p)-1:
  c := s * curv subpath(n,n+1) of p;
  for i = 1 upto 50:
   k := c;
   c := s * curv subpath(n+i/50,
              n if i<25: +1 fi) of p;
   q := subpath(n+(i-1)/50,n+i/50) of p;
   fill q -- point 1 of q + c
          -- point 0 of q + k
          -- cycle withcolor
      (1,1/(1+.1*.5[length c,length k]),0);
  endfor
 endfor
enddef;
```

The condition `if i<25: +1 fi` makes the subpath as large as possible to get better accuracy.

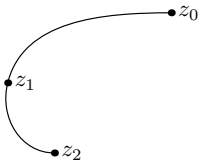
A curvature of 0 is mapped to yellow and an infinitely large curvature is mapped to red. This is done by changing the green value between 1 and 0. If you increase the `.1`, the green value tends faster to 0.

3 Harmonize paths in METAPOST

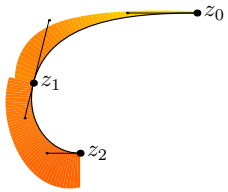
In METAPOST, the code

```
z0 = (70,60); z1 = (0,30); z2 = (20,0);
draw z0{left} .. z1 .. z2{right};
```

produces the following curve:

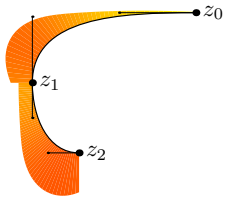


While the directions at the start and the end of the path were set by the user, METAPOST has chosen the angle of the path in z_1 to equalize the so-called *mock curvature* on both sides. The mock curvature is a Taylor approximation of the real curvature (Hobby, 1986). After that, two cubic Bézier segments that nearly minimize the curve energy have been drawn between the given points.



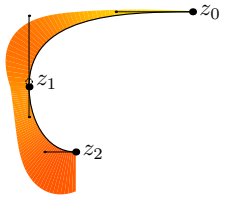

```
z0{left}
.. z1
.. z2{right}
```

The curvature comb in the preceding picture shows that the curvature in z_1 indeed is not continuous but only near continuous. When a user sets the direction in the joining knot, METAPOST has no possibility to optimize the curvature in the joint and the curvature often changes more abruptly in the joint (see the following picture). However, this case is frequent in type design because knots at horizontal and vertical extrema are preferred over knots with arbitrary direction.




```
z0{left}
.. z1{down}
.. z2{right}
```

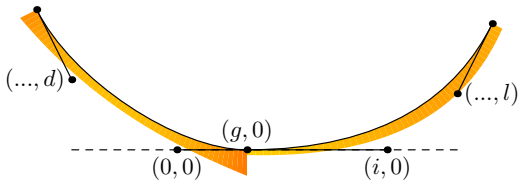
Fortunately, a METAPOST path can be modified to a curvature continuous curve by the later defined `harmonize` macro that moves the joining knot along its tangent:




```
harmonize z0{left}
        .. z1{down}
        .. z2{right}
```

4 The math of harmonization

Assume two adjoint cubic Bézier segments that have the same direction in their joint and do not have zero-handles. Furthermore, assume the joining knot to not be a point of inflection. By translation and rotation we can force one control point next to the joining knot to lie on the origin of the coordinate system and the joining knot tangent to lie on the x-axis:



We want to choose g such that the curvature is continuous. So the curvature on both sides of $(g, 0)$ must

Linus Romer

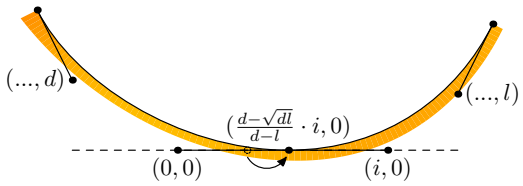
be equal:

$$\frac{2d}{3g^2} = \frac{2l}{3(i-g)^2}$$

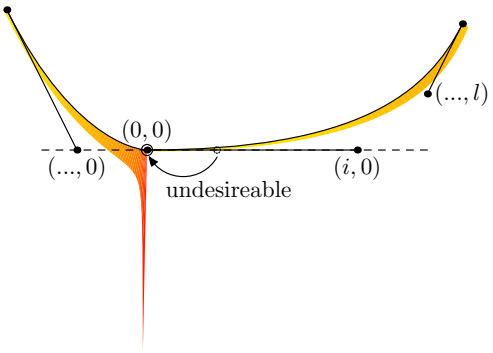
In the special case $d = l$, we get $g = i - g$. Solving for g we get

$$g = \begin{cases} \frac{d \pm \sqrt{dl}}{d-l} \cdot i & \text{if } d \neq l, \\ \frac{i}{2} & \text{else.} \end{cases}$$

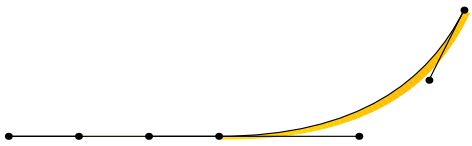
Since $\sqrt{dl}$ is the geometric mean between d and l, the solution $\frac{d-\sqrt{dl}}{d-l} \cdot i$ guarantees the joining knot to lie between its control points.



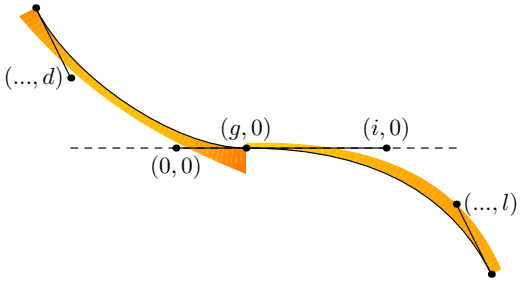
If either d or l is zero, $g = \frac{d-\sqrt{dl}}{d-l} \cdot i$ becomes either 0 or i. That means, that the joining knot will become collocated with one of its control points, which generally should be avoided. One reason for this avoidance is that the curvature might become infinitely large:



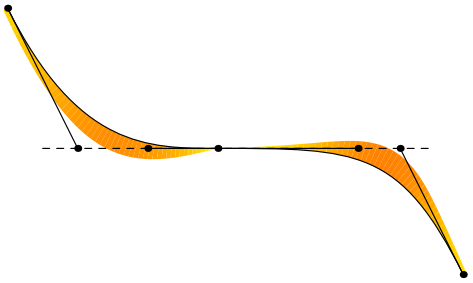
So, we will not alter the paths at all in the case of either d or l being zero. This case occurs also when a straight line goes over to a curve, which is quite frequent in type design:



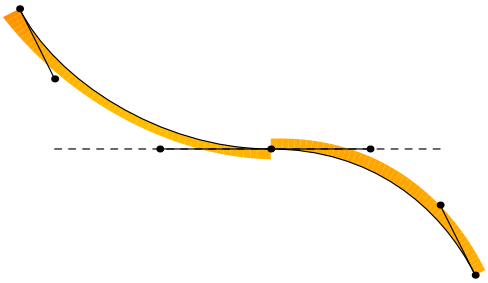
When the joining knot is a point of inflection, the curvatures $\frac{2d}{3g^2}$ and $\frac{2l}{3(i-g)^2}$ must have different signs.



Hence, a curvature-continuous solution forces $d = l = 0$. A curvature continuous solution then only has to satisfy that all control points must lie on one line e.g.:



In this situation, one could also satisfy further conditions like the preservation of area. On the other hand, having all four control points on the same line of the two affected cubic Bézier segments is critical. Due to rounding errors, such a conversion is likely to add new points of inflection. So, instead of this, we will only guarantee the *absolute value* of the curvature to be continuous in the case of points of inflection by moving the joining knot in the same manner as before in between its control points:



Finally, the solution of setting

$$g_{\text{new}} = \begin{cases} g_{\text{old}} & \text{if } d = 0 \text{ or } l = 0, \\ \frac{i}{2} & \text{else if } |d| = |l|, \\ \frac{|d|-\sqrt{|dl|}}{|d|-|l|} \cdot i & \text{else} \end{cases}$$

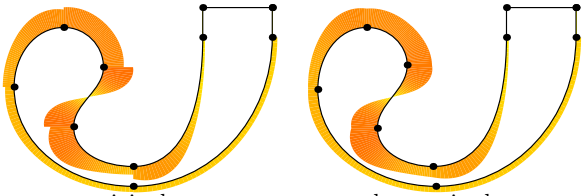
is chosen here and shall be the definition of *harmonization*. We define a corresponding macro `harmonize` that returns a harmonized version of a given path p. In the generic case, the tangent in the joining knot is not the x-axis (as depicted in the preceding figures), so we calculate d and l as the height to the tangent by cross products.

```
vardef harmonize expr p =
 save t,u,d,l,n,q; pair t,u,q[];
 n = length p;
 for j = if cycle p: 0 else: 1 fi upto n-1:
  q[j] = point j of p;
  t := unitvector(direction j of p);
  u := unitvector(point j of p
           - precontrol j of p);
  if eps > abs((u dotprod t) - 1): % smooth
   l := abs(t crossprod
       (precontrol j+1 of p - point j of p) );
   d := abs(t crossprod
       (postcontrol j-1 of p - point j of p) );
   if not ( (l = 0) or (d = 0) ):
    q[j] := if (d = l): .5 else:
        ((d-sqrt(d*l))/(d-l)) fi
        [precontrol j of p,postcontrol j of p];
   fi
  fi
 endfor
 if not cycle p:
  q[0] = point 0 of p;
  q[n] = point n of p;
 fi
 q[0] % start returned path
 for j = 0 upto n-1: % define new path
  .. controls postcontrol j of p
  and precontrol j+1 of p .. if (j = n-1)
  and (cycle p): cycle else: q[j+1] fi
 endfor
enddef;
```

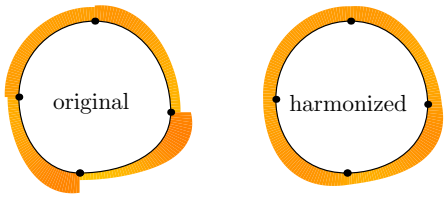
A mostly equivalent algorithm has been published in (Roach, 1990).

5 Examples of harmonization

Should you use harmonization? At least it does not harm to consider it. Most of the time, the changes are subtle as in the following bulb terminal:



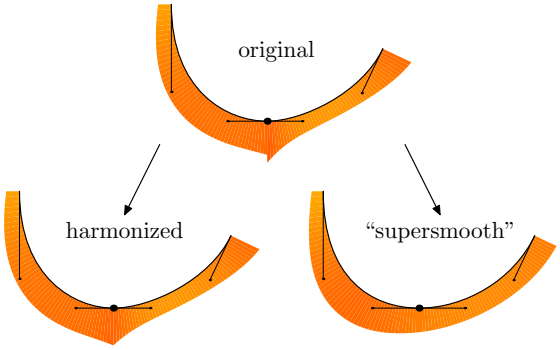
And sometimes they are less subtle as in the following calligraphic dots:



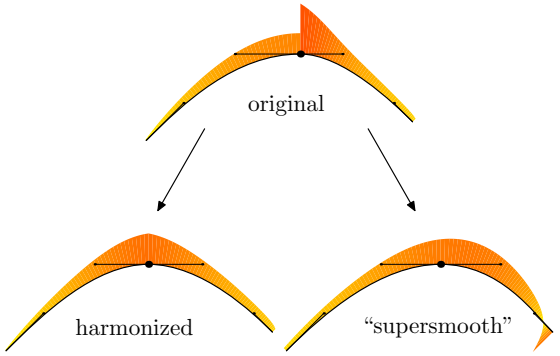
After harmonization, the dot has become more rounded and may have lost its "personality".

6 Smoothing out paths even more

Harmonization does not affect control points nor the curvatures at other joining knots. Hence, it can be easily used over several cubic Bézier segments. Nonetheless, harmonized paths normally no longer interpolate the knots they were originally meant to. The author once thought it might therefore be a good idea to leave the joining knots and move the control points instead. Then we not only can make the curvature continuous but also the the change of curvature. The curve then becomes some kind of "supersmooth".



However, there are some problems that come with this "supersmoothness": It might introduce additional points of inflection (see below). Furthermore, this will normally change the curvature at other knots and break curvature continuity there.



References

Hobby, John D. "Smooth, easy to compute interpolating splines". *Discrete & computational geometry* **1**(2), 123–140, 1986.

Roach, Robert L. "Curvature continuity of cubic Bezier curves in the solid modeling aerospace research tools design software". interim report, NASA Langley Research Center, 1990.

◇ Linus Romer
Ahornstrasse 8
Uznach, 8730
Switzerland

Linus Romer

An updated survey of OpenType math fonts

Ulrik Vieth

Abstract

OpenType math fonts have been introduced more than 15 years ago. Over the years, more and more math fonts have been developed and added to the font collection. In this paper, we review some of the more recent additions, comparing them to previous choices of OpenType math fonts such as Cambria, Lucida, Latin Modern, or TEX Gyre.

In our analysis, we focus on aspects such as completeness of symbols and alphabets, design choices of alphabets, and available font features. However, a detailed study of glyph and font metrics is beyond the scope of this paper, but some aspects of this have been recently addressed by other contributions.

1 Introduction

OpenType math fonts have been introduced more than 15 years ago. It started when Microsoft added support for math typesetting in Office 2007 [1] and proposed an extension of the OpenType font format, adding a MATH table that eventually became part of the OpenType standard [2].

It didin't take long until the TEX community recognized the potential of OpenType math fonts [3, 4] and started adopting the font technology for their own purposes.

XETEX started in 2008 to introduce limited support for OpenType math in the scope of an extended TEX math engine [5]. LuaTEX followed in 2009 with a more complete implementation, aiming to provide a full-featured OpenType math engine [6, 7].

Since 2010 both engines and supporting macro packages and font loaders have been available in the mainstream TEX Live distribution. At this point, the technology for OpenType math typesetting was essentially ready for use, except that there weren't many OpenType math fonts available yet.

2 Overview of available math fonts

When OpenType math was introduced, only a single math font was available: Cambria Math [8] by Tiro Typeworks, which was commissioned by Microsoft and distributed as a system font with Office 2007. Cambria Math was also intended as a reference implementation show-casing the features of OpenType math, illustrated in a promotional booklet.

This was followed in 2008 by Asana Math [9] by Apostolos Syropoulos as the first independently developed OpenType math font, which was based on `pxfonts` by Young Ryu.

When the STIX fonts 1.0 were released in 2010, they were quickly assembled as an OpenType math font and released as the XITS fonts [10]. It was only years later that OpenType versions of STIX fonts were released with STIX fonts 1.1.1 in 2013 and the much revised STIX2 fonts in 2016 [11, 12].

In the meantime, the earlier XITS and STIX fonts are considered obsolete, and only the STIX2 fonts are maintained.

Perhaps the most significant contribution to the collection of math fonts came in 2011–2014 with the development of the Latin Modern and TEX Gyre math fonts by the GUST font team with support from various TEX user groups [13, 14, 15, 16].

Another contribution by the GUST team was the development of a math font for DejaVu in 2015, which was added to the TEX Gyre collection.

During the same time came the development of Lucida OpenType text and math fonts in 2011–2012, which was initiated as a TUG project with support from Bigelow & Holmes and a group of volunteers. While the Lucida fonts aren't free, they are available at a very reasonable price from TUG [17].

With these developments, there were already more than 10 choices of OpenType math fonts in 2015, when there were just a few in 2010.

But there was more to come: In the following years, more math fonts were added, complementing various freely available OpenType text fonts.

Starting in 2016, Khaled Hosny developed the Libertinus OTF fonts [18], derived from Libertine and Biolinum, and added a Libertinus Math font, borrowing some symbols and alphabets from other existing fonts such as the STIX fonts.

A Garamond Math font [19] followed in 2018, developed by Yuansheng Zhao, using alphabets from EB Garamond and borrowing a sans-serif alphabet from Libertinus Math.

Daniel Flipo provided the Erewhon Math and XCharter Math fonts [20, 21], using alphabets from Michael Sharpe's Erewhon[1] and XCharter text fonts, which, in turn, are derived from extended versions of Adobe Utopia and Bitstream Charter. The math symbols for Utopia and Charter are based on the Fourier-GUT and MathDesign packages by Michel Bovani and Paul Pichaureau.

Another recent contribution by Daniel Flipo is the KpFonts OTF collection [22], based on KpFonts by Christophe Caignaert, which, in turn, is derived from a version of URW Palladio (not Kepler!) and complemented by a sans-serif and a monospace to make a complete font family.

[1] *erewhon* backwards is *nowhere*, which alludes to Utopia.

font name	first	latest	version	release	sources	developer, maintainer	ref.
Cambria Math	2007	2019	6.99	MS	—	Microsoft, Tiro Typeworks	[8]
Asana Math	2008	2019	0.958	CTAN	—	Apostolos Syropoulos	[9]
XITS Math	2010	2020	1.302	CTAN	Github	Khaled Hosny	[10]
STIX Math (obsolete)	2010	2014	1.1.1	CTAN	Github	David Jones, STIpub	[11]
STIX Two Math	2016	2021	2.13	CTAN	Github	David Jones, STIpub	[12]
Latin Modern Math	2011	2014	1.959	CTAN	GUST	GUST font team	[13]
TEX Gyre Pagella Math	2012	2016	1.632	CTAN	GUST	GUST font team	[14]
TEX Gyre Termes Math	2012	2016	1.543	CTAN	GUST	GUST font team	[14]
TEX Gyre Bonum Math	2013	2016	1.005	CTAN	GUST	GUST font team	[14]
TEX Gyre Schola Math	2014	2016	1.533	CTAN	GUST	GUST font team	[14]
TEX Gyre DejaVu Math	2015	2016	1.106	CTAN	GUST	GUST font team	[14]
Lucida Bright Math	2012	2023	1.901	TUG	—	Bigelow & Holmes, TUG	[17]
Libertinus Math	2016	2021	7.040	CTAN	Github	Khaled Hosny	[18]
Garamond Math	2018	2022	2022-01	CTAN	Github	Yuansheng Zhao	[19]
Erewhon Math	2019	2023	0.60	CTAN	—	Daniel Flipo	[20]
XCharter Math	2022	2023	0.40	CTAN	—	Daniel Flipo	[21]
KpFonts (Roman, Sans)	2020	2023	0.52	CTAN	—	Daniel Flipo	[22]
GFS Neohellenic Math	2016	2022	1.02	CTAN	—	Antonis Tsolomitis, GFS	[23]
Fira Math	2018	2020	0.3.4	CTAN	Github	Xiangdong Zeng	[24]
Lato Math	2020	2020	??	—	Github	Chenjing Bu	[25]
Noto Math	2020	2023	2.539	—	Github	Noto Fonts Project	[27]
New CM Math	2019	2023	4.5	CTAN	—	Antonis Tsolomitis	[29]
Concrete Math	2022	2023	0.40	CTAN	—	Daniel Flipo	[30]
Euler Math	2022	2023	0.40	CTAN	—	Daniel Flipo, Khaled Hosny	[31]

Table 1: List of available OpenType math font packages with dates of first and latest releases, latest versions, availability of releases and sources, developer or maintainer, as well as links to resources.

While KpFonts also includes a sans-serif design, it is not the only sans-serif math font available.

There is GFS Neohellenic Math [23] maintained by Antonis Tsolomitis, which is based on a sans-serif font in neo-hellenic style that was developed by the Greek Font Society (GFS).

Another example is Fira Math [24] developed by Xiangdong Zeng in 2018, using alphabets from Fira Sans and corresponding math symbols.

There also exists a project for Lato Math [25], using alphabets from Lato [26] by Łukasz Dziedzic combined with symbols borrowed from Fira Math. Unfortunately, the project seems unfinished and is unsuitable for distribution in the current state.

Another very recent project, started in 2023, aims to provide OpenType math functionality for Noto Math [27]. While the font already exists for some years, it only provided the glyphs, but it didn't come with a MATH table, so it was lacking the math typesetting functionality. When the project is done, it will provide another important addition to the collection of sans-serif math fonts.

Finally, besides all the developments to provide math support for various existing OpenType fonts, there has also been renewed interest in extending and reviving some traditional TEX fonts.

A significant extension is the New Computer Modern font family [28, 29] by Antonis Tsolomitis, which extends Latin Modern fonts in many ways. Besides numerous additions to the text fonts, it also adds additional Unicode blocks of mathematical and technical symbols to the math fonts. As a result, these fonts are now the most complete math fonts, even more complete than STIX fonts.

Another recent contribution by Daniel Flipo has revived some traditional TEX fonts, providing OTF versions of Concrete Math and Euler Math [30, 31]. While Concrete Math was generated from sources, Euler Math is based on Neo Euler [32] by Khaled Hosny, started in 2009, which originated from a collaboration with Hermann Zapf more than a decade ago and was long since abandoned [33].

With these developments, we now have more than 20 choices of OpenType math fonts in 2023 (not counting variants). This is a significant increase compared to the numbers of 2015 or 2010.

A summary of available OpenType math font packages is provided in table 1.

font name	weights
XITS Math	Regular, Bold
Lucida Bright Math	Regular, Demi
Erewhon Math	Regular, Bold (minimal)
XCharter Math	Regular, Bold (minimal)
KpRoman Math	Light, Semibold
	Regular, Bold
KpSans Math	Regular, Bold
New CM Math	Regular, Book

Table 2: List of available OpenType math fonts with bold versions or additional weights.

Some OpenType math font packages come with multiple weights, so the total number of individual font shapes is actually more than 30 now.

In some cases, there is a fairly complete bold math font, in other cases, only a bare minimum is provided, suitable for inline math only.

Besides bold math fonts, there are also some font packages which provide multiple weights of the base fonts, such as light or book variants.

A summary of OpenType math fonts with bold or additional weights is provided in table 2.

Nearly all OpenType math fonts discussed in this paper are free and readily available from CTAN or TEX Live. However, some unfinished projects are currently only available from Github.

The only non-free fonts discussed in this paper are Cambria Math, which comes as a system font on Windows, and the Lucida fonts, which are sold via TUG. We have excluded other non-free fonts since we don't have any up-to-date information.

In this paper, we want to analyze how the available math fonts compare with regards to coverage of symbols and alphabets, and with regards to design choices of alphabets.

Some of these topics have also been considered in an earlier review [34], which reflected the state of math fonts in 2012, when just a few OpenType math fonts were available, such as Cambria, Lucida, Latin Modern, and some TEX Gyre fonts.

In this review, we provide an update on the state of OpenType math fonts in 2023 with many updated and many additional fonts available.

Given the number of available fonts, a detailed technical study of font parameters and glyph metrics is beyond the scope of this paper.

Fortunately, some recent studies by LuaMetaTEX developers[2] have covered this topic in detail and have also resulted in improvements or repairs of several OpenType math fonts [35, 36, 37].

[2] LuaMetaTEX (`LMTX`) is a follow-up of LuaTEX.

3 Completeness of available math fonts

In the following sections, we want to analyze how the available math fonts compare with regards to completeness of symbols and alphabets.

In order to determine the range of coverage, we are essentially counting the number of Unicode slots provided in a given OpenType font.

This could be done using a test script such as Frank Mittelbach's `unicodefonttable` package [38, 39], which generates a Unicode font table for a given font and counts the available glyphs.

A similar approach, more specific to math fonts, would be to adapt the `unimath-symbols,ltx` table from the documentation of `unicode-math` package [40], which typesets a font table of Unicode math symbols encoded in `unicode-math-table.tex` and counts the available glyphs.

In our case, we have used a modified version of this, which provides separate counts for symbols and alphabetic characters. We have also used a modified version of the symbol table.

The numbers determined this way represent a lower estimate for the available glyphs, since we are only counting the base glyphs in Unicode slots and only the known symbols.

In most cases, OpenType math fonts provide more than just the base glyphs. For big operators, big delimiters, wide accents, or similar objects, there are multiple sizes and an extensible versions.

Besides additional sizes, many OpenType math fonts also provide additional glyph variants that can be accessed via stylistic sets.

It is difficult to determine an exact number of glyphs that should be provided to make a math font complete. The boundary between mathematical and technical symbols is a little vague and the decision which symbols to include or exclude in the encoding table could be somewhat subjective.

Furthermore, Unicode comes with new releases every year, so there could be additional symbols added from time to time, which could be overlooked if they are missing in the symbol table.

Some of the most complete OpenType math fonts amount to 1270 symbols and 1170 alphabetic characters, so there would be 2440 glyphs in total, not counting any sizes or variants. If we include the additional sizes and variants, there will be even more glyphs needed for a complete math font.

While the glyph variants are usually hidden and excluded from the count, some font designers make them available in the private-use area, which could add them to the total count of Unicode slots.

3.1 Completeness of math symbols

When analyzing the counts regarding completeness of math symbols, we find that there are essentially two groups of OpenType math fonts.

The first group aims for completeness, covering more or less the complete range of Unicode math, providing some 1150–1270 math symbols:

New CM Math	1270 symbols
STIX Two Math	1256 symbols
XITS Math	1253 symbols
Lato Math	1221 symbols
Asana Math	1211 symbols
GFS Neohellenic Math	1175 symbols
Noto Math	1162 symbols
Cambria Math	1157 symbols
Lucida Bright Math	951 symbols

In this group we find fonts that were designed for completeness such as STIX/XITS, Noto, or Lato, but also some new entries such as New CM Math, which is currently the most complete math font. Cambria is also fairly complete by now, but was much less complete in earlier versions. Lucida is somewhere in between: It is a little behind the first group, but way ahead of the second group.

The second group does not aim for completeness and covers only a subset of symbols, providing some 500–600 math symbols:

Garamond Math	604 symbols
Erewhon Math	599 symbols
Euler Math	591 symbols
KpFonts (Roman, Sans)	589 symbols
XCharter Math	577 symbols
Libertinus Math	560 symbols
TEX Gyre Math (5×)	556 symbols
Latin Modern Math	554 symbols
Fira Math	508 symbols
Concreate Math	499 symbols

Among this group, the Latin Modern and TEX Gyre math fonts by the GUST font team provide a a consistent subset across all fonts, which could be taken as a starting point for a common subset encoding. Unfortunately, there is not much agreement among other fonts, so the details of symbol coverage will be slightly different for each font.

While a subset of 500–600 math symbols may seem small compared to the full Unicode symbol range, it is actually not that small. If we consider that a traditional TEX with AMS fonts had no more than 5 fonts of 128 slots to encode all the math symbols and alphabets, any OpenType font with 500–600 symbols (not including alphabets) will be as good as any traditional TEX font.

Finally, it is interesting to note how bold math fonts compare, if they are provided at all.

Since the regular math fonts already include bold math alphabets for semantic markup, separate bold math fonts are only needed in the context of headings, when formulas are switched to bold as a whole, and it may be reasonable to assume that only inline math will be used in this context.

As shown in table 2, only a few font packages provide a separate bold math font, and these bold versions come with a smaller range of math symbols compared to the regular versions:

XITS Math Bold	499 symbols
KpFonts (Roman, Sans)	495 symbols
Lucida Bright Math Demi	478 symbols
Erewhon Math Bold	114 symbols
XCharter Math Bold	107 symbols

In the case of Erewhon Math and XCharter Math, the idea of only providing support for inline math was taken to the extreme, omitting most of the big operators and big delimiters, and only including the basic sizes of the most common symbols.

3.2 Completeness of math alphabets

When analyzing the counts regarding completeness of math alphabets, we find that there are again several groups of OpenType math fonts.

The first group aims for completeness, covering all of the math alphabets, providing some 1150–1170 alphabetic symbols:

New CM Math	1170 alphabetic
STIX Two Math	1170 alphabetic
XITS Math	1170 alphabetic
Cambria Math	1170 alphabetic
Asana Math	1167 alphabetic
Noto Math	1164 alphabetic
TEX Gyre Math (5×)	1163 alphabetic

The second group is a little less complete, covering most of the math alphabets with some limitiations, providing some 1050–1150 alphabetic symbols:

Libertinus Math	1145 alphabetic
Erewhon Math	1117 alphabetic
Latin Modern Math	1111 alphabetic
Garamond Math	1100 alphabetic
XCharter Math	1073 alphabetic
KpRoman Math	1068 alphabetic
Lucida Bright Math	1038 alphabetic

Among the most common omissions are lowercase Script and BBold, which are missing in several fonts. Lucida Math is missing only lower bold Script and bold Fraktur. Garamond Math is missing lowercase Greek in sans serif bold italic.

font name	regular up it bf bi	sans-serif up it bf bi	Script scr bscr	Fraktur frak bfrak	BBold bb	Mono tt
Cambria Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
Asana Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
XITS Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
STIX Two Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
Latin Modern Math	● ● ● ●	● ● ● ●	● – ● –	● ● ● ●	● ●	● ●
TEX Gyre Math (5×)	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
Lucida Bright Math	● ● ● ●	● ● ● ●	● ● ● –	● ● – –	● –	● ●
Libertinus Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
Garamond Math	● ● ● ●	● ● ● ○	● ● ● ●	● ● ● ●	● ●	● ●
Erewhon Math	● ● ● ●	● ● ● ●	● – ● –	● ● ● ●	● ●	● ●
XCharter Math	● ● ● ●	● ● ● ●	● – ● –	● ● ● ●	● –	● ●
KpRoman Math	● ● ● ●	● ● ● ●	● – ● –	● ● ● ●	● –	● ●
KpSans Math	● ● ● ●	– – – –	● – ● –	● ● ● ●	● –	● ●
GFS Neohellenic Math	● ● ● ●	– – – –	● – – –	● – – –	● –	– –
Fira Math	● ● ● ●	– – – –	– – – –	– – – –	● ●	● ●
Lato Math	● ● ● ●	– – – –	– – – –	– – – –	● ●	● ●
Noto Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
New CM Math	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	● ●
Concrete Math	● ● ● ●	– – – –	● – – –	● ● ● ●	● –	– –
Euler Math	● – ● –	– – – –	● – ● –	● ● ● ●	● ●	– –

Table 3: List of available OpenType math fonts with coverage of math alphabets. For regular and sans-serif the columns indicate upright, italic, bold and bold italic. For Script, Fraktur, BBold the columns indicate upper- and lowercase.

The third group consists of sans-serif or special designs, which usually leave out the sans-serif slots, resulting in much lower numbers:

KpSans Math	720 alphabetic
Concrete Math	634 alphabetic
Lato Math	606 alphabetic
Fira Math	584 alphabetic
GFS Neohellenic Math	568 alphabetic
Euler Math	480 alphabetic

Again, the most common omissions are lowercase Script and BBold, which are missing in several fonts. GFS Neohellenic is missing lower and bold Script and Fraktur, as well as lower BBold. Lato and Fira are missing all of Script and Fraktur, but they do provide a full set of BBold.

Euler uses a special setup, which only provides an upright version of the base font, so besides the omission of sans-serif and typewriter slots, it also leaves out the italic and bold italic slots.

While most sans-serif fonts provide a reduced set of math alphabets, Noto is an exception that provides the complete range of alphabets, but uses an unusual approach. While the upright uses a sans-serif font, the italic, bold, and bold italic happen to use a serif font. Then again, a full set of sans-serif alphabets are also provided.

Finally, it is interesting to note how bold math fonts compare in terms of math alphabets.

When formulas are switched to bold as a whole in the context of headings, regular alphabets will be replaced by bold alphabets, and bold alphabets should ideally become heavier, but usually they just remain bold, if they are included at all.

Depending on what is included or omitted, the numbers of alphabetic symbols vary a lot:

XITS Math Bold	1093 alphabetic
Erewhon Math Bold	970 alphabetic
Lucida Bright Math Demi	961 alphabetic
KpFonts (Roman, Sans)	362 alphabetic
XCharter Math Bold	317 alphabetic

Gaps in the regular fonts are usually reflected in the bold fonts: Lucida is already missing lower bold Script and bold Fraktur in the regular font, so the bold font is also missing Script and Fraktur.

Finally, some bold fonts have chosen to provide only a minimum set, so besides the omission of sans-serif and typewriter slots, they also leave out bold alphabets when the regular alphabets are switched to bold, resulting in even lower numbers.

A summary of available or missing alphabets in the various math fonts and bold math fonts is provided in tables 3 and 4.

font name	regular up it bf bi	sans-serif up it bf bi	Script scr bscr	Fraktur frak bfrak	BBold bb	Mono tt
XITS Math Bold	● ● ● ●	● ● ● ●	● ● ● ●	● ● ● ●	● ●	– –
Lucida Bright Math Demi	● ● ● ●	● ● ● ●	● – ● –	– – – –	● –	● ●
Erewhon Math Bold	● ● ● ●	● ● ● ●	● – ● –	● ● ● ●	– –	– –
XCharter Math Bold	● ● – –	– – – –	● – ● –	● ● – –	– –	– –
KpRoman Math Bold	● ● – –	– – – –	● – ● –	● ● – –	● –	– –
KpSans Math Bold	● ● – –	– – – –	● – ● –	● ● – –	● –	– –

Table 4: List of available OpenType bold fonts with coverage of math alphabets. For regular and sans-serif the columns indicate upright, italic, bold and bold italic. For Script, Fraktur and BBold the columns indicate upper- and lowercase.

While it may be difficult to keep track of the details, users of OpenType math fonts shouldn't be too concerned about missing alphabets, unless they have special requirements.

In general, OpenType math fonts provide more math alphabets than traditional TEX math fonts, and most of the gaps only affect specific alphabets, which may not be used much.

It should be safe to assume that nearly all OpenType math fonts provide at least the main alphabet in 4 shapes, including Latin and Greek, as well as a basic set of Script, Fraktur, and BBold.

There may be gaps when it comes to lowercase Script, lowercase BBold, bold Scrip,t or bold Fraktur, but these are much less used. There may also be gaps in the sans-serif or typewriter alphabets.

4 Design choices of math alphabets

For a full-featured OpenType math font, a number of math alphabets are required:

- 4 shapes of the main font (upright, italic, bold, bold italic), each including Latin and Greek,
- 4 shapes of a sans-serif (upright, italic, bold, bold italic), some including Latin and Greek,
- 2 shapes of Script/Calligraphic (regular, bold), each including upper- and lowercase,
- 2 shapes of Fraktur/Blackletter (regular, bold), each including upper- and lowercase,
- 1 shape of Blackboard bold or BBold (regular), also including upper- and lowercase,
- 1 shape of a monospace/typewriter (regular), also including upper- and lowercase.

To provide all these alphabets, it will be necessary to assemble glyphs from multiple sources and to adjust them to match the main font.

When dealing with a comprehensive font family, some choices may be obvious, such as choosing a sans-serif or a typewriter font, but in most cases some design decisions will be needed.

In the following sections, we want to consider how the available OpenType math fonts compare with regards to design choices of math alphabets for Script, Fraktur, and Blackboard Bold.

While some design choices in existing fonts may be unfortunate, it is hard to change anything, once a font has been released and put into use for some time. It will usually be necessary to create a new variant, when you want to revise some design choices.

This is what happened to the STIX fonts, which were renamed to STIX Two after a major revision of the glyph shapes and some math alphabets.

Similarly, the New Computer Modern fonts can be considered a new variant of Latin Modern. While New Computer Modern can choose to disagree with Latin Modern and use different choices, any future revisions of Latin Modern will likely have to respect previous choices for compatibility.

4.1 Design choices of sans-serif

When choosing a sans-serif font for use in math font, it is important to keep in mind that math alphabets are not meant for generic font switches, but for semantic markup of symbols in a formula. In physics, bold sans-serif italic might be used to for tensors, while bold italic might be used for vectors.

Besides providing a suitable range of Latin and Greek, the sans-serif glyphs also need to be clearly distinguishable from the corresponding serif glyphs based on their font properties, such as weight, width, contrast or stroke thickness.

While having some contrast between serif and sans-serif can be helpful, the sans-serif design should not be too incompatible with the main font, since the symbols from different alphabets should work together in a formula.

In general, it is better to combine serif and sans-serif fonts of similar weight and width, having just enough contrast in between to make them clearly distinguishable. It is also a good idea to use familiar shapes and to avoid any unusual shapes.

4.2 Design choices of Script/Calligrahic

When it comes to choices for Script or Calligraphic, there are two different styles how users expect a mathematical Script to look like.

The first group uses a restrained style of Script or Calligraphic. This includes the traditional styles used in Computer Modern, Euler Script, and Lucida Calligraphic:[3]

Neohellenic	$\mathcal{ABCXYZ}$
Concrete	$\mathcal{ABCXYZ}$
Garamond	$\mathcal{ABCXYZ}$ (StylisticSet=3)
KpFonts	$\mathcal{ABCXYZ}$ (StylisticSet=1)
XITS	$\mathcal{ABCXYZ}$ (StylisticSet=3)
Lucida	$\mathcal{ABCXYZ}$ (StylisticSet=4)
Euler	$\mathcal{ABCXYZ}$
LM	$\mathcal{ABCXYZ}$
New CM	$\mathcal{ABCXYZabcxyz}$
STIX Two	$\mathcal{ABCXYZabcxyz}$
Cambria	$\mathcal{ABCXYZabcxyz}$
TG DejaVu	$\mathcal{ABCXYZabcxyz}$

The second group uses a more fancy and elaborate style of formal Script. This includes the new design of Lucida Script:

Erewhon	$\mathscr{ABCXYZ}$
XCharter	$\mathscr{ABCXYZ}$
KpFonts	$\mathscr{ABCXYZ}$
STIX Two	$\mathscr{ABCXYZ}$ (StylisticSet=1)
XITS	$\mathscr{ABCXYZabcxyz}$
Libertinus	$\mathscr{ABCXYZabcxyz}$
Garamond	$\mathscr{ABCXYZabcxyz}$
TG Termes	$\mathscr{ABCXYZabcxyz}$
TG Schola	$\mathscr{ABCXYZabcxyz}$
Lucida	$\mathscr{ABCXYZabcxyz}$

TEX Gyre Pagella uses a very unique style, which could make this font less usable in general:

TG Pagella	$\mathcal{ABCXYZabcxyz}$

Several OpenType math fonts also provide an alternate style of Script or Calligraphic, which can be accessed using stylistic sets. These variants have also been included in the overview.

It is interesting to note that the STIX Two fonts have reversed a design decision of the XITS fonts regarding the choice of Script, and the designs have also been modified. New Computer Modern extends the Script from Latin Modern using the same style, while Concrete Math has adopted the original style of Calligraphic from Computer Modern.

[3] Some fonts have been scaled to match the size of other fonts: Lucida Calligraphic to 90% and Lucida Script to 85%, DejaVu to 90%, Termes, Pagella, and Schola to 95%.

4.3 Design choices of Fraktur/Blackletter

When it comes to choices for Fraktur or Blackletter, there is only one preferred style how users expect a mathematical Fraktur to look like.

The first group includes a majority of math font packages which use a very typical style of Fraktur. Many fonts make use of Euler Fraktur, such as Latin Modern, New Computer Modern, or Pagella:[4]

LM	$\mathfrak{ABCXYZabcxyz}$
New CM	$\mathfrak{ABCXYZabcxyz}$
Concrete	$\mathfrak{ABCXYZabcxyz}$
Euler	$\mathfrak{ABCXYZabcxyz}$
Erewhon	$\mathfrak{ABCXYZabcxyz}$
XCharter	$\mathfrak{ABCXYZabcxyz}$
TG Pagella	$\mathfrak{ABCXYZabcxyz}$
TG Termes	$\mathfrak{ABCXYZabcxyz}$
Garamond	$\mathfrak{ABCXYZabcxyz}$
Cambria	$\mathfrak{ABCXYZabcxyz}$
Libertinus	$\mathfrak{ABCXYZabcxyz}$
STIX Two	$\mathfrak{ABCXYZabcxyz}$
XITS	$\mathfrak{ABCXYZabcxyz}$
TG Schola	$\mathfrak{ABCXYZabcxyz}$
TG DejaVu	$\mathfrak{ABCXYZabcxyz}$

The second group uses a Blackletter style instead of Fraktur, which is fairly unusual and could make these fonts less usable in general:

Neohellenic	$\mathfrak{ABCXYZ}$
Lucida	$\mathfrak{ABCXYZabcxyz}$
KpFonts	$\mathfrak{ABCXYZabcxyz}$

These designs could be just a fallback option when no suitable design of Fraktur was available.

4.4 Design choices of Blackboard Bold

When it comes to choices for Blackboard Bold, there are again two styles using a sans-serif or serif style of the BBold letters.

The first group uses a sans-serif style of BBold:

LM	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
Euler	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
Erewhon	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
STIX Two	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
XITS	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
Lucida	$\mathbb{ABCNOPQRXYZ}$
KpSans	$\mathbb{ABCNOPQRXYZ}$
Neohellenic	$\mathbb{ABCNOPQRXYZ}$
Fira	$\mathbb{ABCNOPQRXYZ\,abc}$
Noto	$\mathbb{ABCNOPQRXYZ\,abc\,012}$
Lato	$\mathbb{ABCNOPQRXYZ\,abc\,012}$

[4] Some fonts have been scaled to match the size of other fonts: DejaVu to 90%, Schola to 95%. Termes, Pagella, and Lucida Blackletter are not scaled and shown at 100%.

The second group uses a serif style of BBold:[5]

New CM	$\mathbb{ABCNOPQRXYZ}$ abc 012
Concrete	$\mathbb{ABCNOPQRXYZ}$
XCharter	$\mathbb{ABCNOPQRXYZ}$
KpRoman	$\mathbb{ABCNOPQRXYZ}$
Garamond	$\mathbb{ABCNOPQRXYZ}$ abc 012
Libertinus	$\mathbb{ABCNOPQRXYZ}$ abc 012
Cambria	$\mathbb{ABCNOPQRXYZ}$ abc 012
TG Schola	$\mathbb{ABCNOPQRXYZ}$ abc 012
TG Termes	$\mathbb{ABCNOPQRXYZ}$ abc 012
TG Pagella	$\mathbb{ABCNOPQRXYZ}$ abc 012
TG DejaVu	$\mathbb{ABCNOPQRXYZ}$ abc 012

While Latin Modern has adopted a sans-serif BBold, which also includes lowercase and numerals, New Computer Modern and Concrete Math have reverted to the traditional style of BBold from AMS fonts, at least for the uppercase. Many other fonts have chosen a scaled or adjusted variant of the sans-serif BBold from STIX/XITSfonts.

5 Summary and Conclusions

OpenType math fonts have been introduced more than 15 years ago. Over the years, more and more math fonts have been developed and added to the font collection. As of this year, we have more than 20 choices of OpenType math fonts available (not counting variants) and more than 30 individual fonts (including variants and additional weights).

Nearly all OpenType math fonts discussed in this paper are free and readily available from CTAN or TEX Live, except for some non-free fonts and some unfinished projects from Github.

The available choices of OpenType math fonts cover most of what was previously available in other formats, including traditional TEX fonts (Computer Modern, Concrete, Euler), standard PostScript fonts (Times, Palatino, etc), and other free PostScript fonts (Garamond, Utopia, Charter, DejaVu).

In our analysis, we have analyzed the coverage of math symbols and alphabets, as well as design choices and available font features.

While the range of symbols and alphabets may vary for each font, most available fonts will be good enough for general use, providing at least as much as traditional TEX fonts or even more.

Regarding design choices, most available font packages follow some typical styles how users expect mathematical Script, Fraktur, or Blackboard Bold to look like. There are only few exceptions which use a unique or unusual style.

[5] Some fonts have been scaled to match the size of other fonts: DejaVu to 85%, Termes, Pagella, and Schola to 90%. Lucida is not scaled and shown at 100%.

In general, OpenType math fonts are not expected to provide the same level of stability and compatibility as traditional TEX fonts. While it should always be possible to reprocess existing documents, you cannot expect the exact same line breaks, unless you archive the specific versions of fonts.

In some cases, OpenType math fonts happen to be stable simply because they haven't been updated for years, but they may still exhibit the same bugs or limitations. Over time, it becomes more and more difficult to change anything, the longer a font has been left unchanged, and it may be necessary to introduce new variants for major revisions.

While font development is ongoing, OpenType math fonts are readily available for use.

References

[1] Murray Sargent: High-quality editing and display of mathematical text in Office 2007. https://learn.microsoft.com/en-us/archive/blogs/murrays

[2] Microsoft Typography: OpenType specification, version 1.9, December 2021. https://learn.microsoft.com/en-us/typography/opentype/spec

[3] Ulrik Vieth: Do we need a Cork math font encoding? *TUGboat*, 29(3), 426–434, 2008. https://tug.org/TUGboat/tb29-3/tb93vieth.pdf
Reprinted in *MAPS*, 38, 3–11, 2009. https://ntg.nl/maps/38/02.pdf

[4] Ulrik Vieth: OpenType Math Illuminated. *TUGboat*, 30(1), 22-31, 2009. https://tug.org/TUGboat/tb30-1/tb94vieth.pdf
Reprinted in *MAPS*, 38, 12–21, 2009. https://ntg.nl/maps/38/03.pdf

[5] Jonathan Kew: XETEX Live. *TUGboat*, 29(1), 151–156, 2008. https://tug.org/TUGboat/tb29-1/tb91kew.pdf

[6] Taco Hoekwater: Math in LuaTEX 0.40. *MAPS*, 38, 22–31, 2009. https://ntg.nl/maps/38/04.pdf

[7] Hans Hagen: LuaTEX math enhancements. *TUGboat*, 37(3), 269–274, 2016. https://tug.org/TUGboat/tb37-3/tb117hagen-otmath.pdf

[8] Tiro Typeworks: Cambria Math. https://tiro.com/projects.html

[9] Apostolos Syropoulos: Asana Math. https://ctan.org/pkg/asana-math

[10] Khaled Hosny: XITS font package.
https://ctan.org/pkg/xits
https://github.com/alif-type/xits

[11] STIX Consortium: STIX font package.
https://ctan.org/pkg/stix

[12] STIX Consortium: STIX2 font package.
https://ctan.org/pkg/stix2-otf
https://github.com/stipub/stixfonts

[13] GUST e-foundry: Latin Modern Math.
https://ctan.org/pkg/lm-math
https://gust.org.pl/projects/e-foundry

[14] GUST e-foundry: TEX Gyre Math.
https://ctan.org/pkg/tex-gyre-math
https://gust.org.pl/projects/e-foundry

[15] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski: GUST e-foundry font projects.
TUGboat, 37(3), 269–274, 2016.
https://tug.org/TUGboat/tb37-3
/tb117jackowski.pdf

[16] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski: Parametric math symbol fonts.
TUGboat, 38(2), 208–211, 2017.
https://tug.org/TUGboat/tb38-2
/tb119jackowski.pdf

[17] TEX Users Group: Lucida fonts from TUG.
https://tug.org/store/lucida/

[18] Khaled Hosny: Libertinus Fonts.
https://ctan.org/pkg/libertinus-fonts
https://github.com/alerque/libertinus

[19] Yuansheng Zhao, Xiangdong Zeng: Garamond Math.
https://ctan.org/pkg/garamond-math
https://github.com/YuanshengZhao
/Garamond-Math

[20] Daniel Flipo: Erewhon Math.
https://ctan.org/pkg/erewhon-math

[21] Daniel Flipo: XCharter Math.
https://ctan.org/pkg/xcharter-math

[22] Daniel Flipo: KpFonts OTF package.
https://ctan.org/pkg/kpfonts-otf

[23] Antonis Tsolomitis: GFS Neohellenic Math.
https://ctan.org/pkg/gfsneohellenicmath

[24] Xiangdong Zeng: Fira Math.
https://ctan.org/pkg/firamath
https://github.com/firamath/firamath

[25] Chenjing Bu: Lato Math.
https://github.com/abccsss/LatoMath

[26] Łukasz Dziedzic: Lato Fonts.
https://github.com/latofonts/lato-source

[27] Noto Fonts Project: Noto Math.
https://github.com/notofonts/math

[28] Antonis Tsolomitis: New Computer Modern font family. *TUGboat*, 42(1), 52–55, 2021.
https://tug.org/TUGboat/tb42-1
/tb130tsolomitis-newcm.pdf

[29] Antonis Tsolomitis: New Computer Modern.
https://ctan.org/pkg/newcomputermodern

[30] Daniel Flipo: Concrete Math.
https://ctan.org/pkg/concmath-otf

[31] Daniel Flipo: Euler Math.
https://ctan.org/pkg/euler-math

[32] Khaled Hosny: Neo Euler — An abandoned OpenType port of Euler math font.
https://github.com/aliftype/euler-otf

[33] Hans Hagen, Taco Hoekwater, Volker Schaa: Reshaping Euler: A collaboration with Hermann Zapf.
TUGboat, 29(3), 283–287, 2008.
https://tug.org/TUGboat/tb29-2
/tb92hagen-euler.pdf

[34] Ulrik Vieth: OpenType math font development: Progress and challenges.
TUGboat, 33(3), 302–308, 2012.
https://tug.org/TUGboat/tb33-3
/tb105vieth.pdf

[35] Hans Hagen, Mikael P. Sundqvist: Pushing math forward with ConTEXt LMTX.
TUGboat, 43(2), 202–206, 2022.
https://tug.org/TUGboat/tb43-2
/tb134hagen-math.pdf

[36] Hans Hagen, Mikael P. Sundqvist: New directions in math fonts.
TUGboat, 43(3), 300–310, 2022.
https://tug.org/TUGboat/tb43-3
/tb135hagen-mathchange.pdf

[37] Hans Hagen, Mikael P. Sundqvist: Patching Lucida Bright Math.
TUGboat, 43(3), 311–316, 2022.
https://tug.org/TUGboat/tb43-3
/tb135hagen-lucida.pdf

[38] Frank Mittelbach: The unicodefonttable package.
TUGboat, 42(3),287–304, 2021.
https://tug.org/TUGboat/tb42-3
/tb132mitt-unicodefonttable.pdf

[39] Frank Mittelbach: unicodefonttable.
https://ctan.org/pkg/unicodefonttable

[40] Will Robertson: unicode-math package.
https://ctan.org/pkg/unicode-math

◇ Ulrik Vieth
Stuttgart, Germany
ulrik dot vieth (at) arcor dot de

Google