

TUGBOAT

Volume 34, Number 2 / 2013

General Delivery	111	Ab epistulis / Steve Peter
	111	Editorial comments / Barbara Beeton Barry Smith, 1953–2012; Yet another DEK interview; TeXdoc on line; Fonts, typography, and printing—on the web and in print
	112	In memoriam: Barry Smith (1953–2012) / Doug Henderson
	113	Hyphenation exception log / Barbara Beeton
	114	Running TeX under Windows PowerShell / Adeline Wilcox
Dreamboat	115	Does TeX have a future? / Hans Hagen
Software & Tools	120	TeX Collection 2013 DVD / TeX Collection editors
	121	What is new in XeTeX 0.9999? / Khaled Hosny
	123	MetaPost: PNG output / Taco Hoekwater
	124	Converting Wikipedia articles to LATEX / Dirk Hüniger
Fonts	125	A survey of text font families / Michael Sharpe
	132	Glisternings: A font of fleurons; Fonts, GNU/Linux, and XeTeX; Mixing traditional and system fonts / Peter Wilson
	136	Interview with Charles Bigelow / Yue Wang
Typography	168	Oh, oh, zero! / Charles Bigelow
Hints & Tricks	181	Production notes / Karl Berry
	182	The treasure chest / Karl Berry
LATEX	184	Preparing for scientific conferences with LATEX: A short practical how-to / Paweł Łukowski and Mariusz Urbański
Literate Programming	190	LiPPGen: A presentation generator for literate-programming-based teaching / Hans-Georg Eßer
Graphics	196	Entry-level MetaPost 2: Move it! / Mari Voipio
	200	Creating Tufte-style bar charts and scatterplots using PGFPlots / Juernjakob Dugge
	205	Typographers, programmers and mathematicians, or the case of an aesthetically pleasing interpolation / Bogusław Jackowski
Philology	223	Representing linguistic pitch in (Xe)LATEX / Kevin Donnelly
Book Reviews	228	Book review: <i>Learning LATEX</i> / Boris Veytsman
	229	Book review: <i>Zapf Exhibition: The Calligraphy of Hermann & Gudrun Zapf</i> / Boris Veytsman
	230	Book review: <i>What Our Lettering Needs</i> / Will Adams
	231	Book review: <i>LATEX Quick Reference</i> / Boris Veytsman
	232	Book review: <i>Presentations with LATEX</i> / Boris Veytsman
Advertisements	233	TeX consulting and production services
Abstracts	235	Die <i>TEXnische Komödie</i> : Contents of issue 2/2013
	235	<i>ArsTEXnica</i> : Contents of issue 15 (April 2013)
	236	GUST: EuroBachoTeX 2013 proceedings
TUG Business	110	TUGboat editorial information
	110	TUG institutional members
	240	TUG 2013 election
News	243	Calendar
	244	TUG 2013 announcement

T_EX Users Group

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

Memberships and Subscriptions

2013 dues for individual members are as follows:

- Ordinary members: \$95.
- Students/Seniors: \$65.

The discounted rate of \$65 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the T_EX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

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

Institutional Membership

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

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which commonly appear in *TUGboat* should not be considered complete.

T_EX is a trademark of American Mathematical Society.

METAFONT is a trademark of Addison-Wesley Inc.

PostScript is a trademark of Adobe Systems, Inc.

[printing date: August 2013]

Printed in U.S.A.

Board of Directors

Donald Knuth, *Grand Wizard of T_EX-arcana*[†]

Steve Peter, *President**

Jim Hefferon*, *Vice President*

Karl Berry*, *Treasurer*

Susan DeMeritt*, *Secretary*

Barbara Beeton

Kaja Christiansen

Michael Doob

Jonathan Fine

Steve Grathwohl

Taco Hoekwater

Klaus Höppner

Ross Moore

Cheryl Ponchin

Philip Taylor

David Walden

Raymond Goucher, *Founding Executive Director*[†]

Hermann Zapf, *Wizard of Fonts*[†]

*member of executive committee

[†]honorary

See <http://tug.org/board.html> for a roster of all past and present board members, and other official positions.

Addresses

T_EX Users Group

P. O. Box 2311

Portland, OR 97208-2311

U.S.A.

Electronic Mail

(Internet)

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

Telephone

+1 503 223-9994

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

Fax

+1 815 301-3568

Technical support for
T_EX users:
support@tug.org

Web

<http://tug.org/>

<http://tug.org/TUGboat/>

Contact the Board
of Directors:
board@tug.org

Copyright © 2013 T_EX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the T_EX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

“The Internet is completely decentr-
alized,” Mr. Rimovsky said.

[anonymous]
International Herald Tribune
(14 September 1998)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 34, NUMBER 2 • 2013
PORTLAND • OREGON • U.S.A.

TUGboat editorial information

This regular issue (Vol. 34, No. 2) is the second issue of the 2013 volume year.

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

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

Submitting items for publication

The third issue for this year will be the TUG 2013 proceedings. The deadline for receipt of final papers for that issue is November 4.

The *TUGboat* style files, for use with plain TeX and L^AT_EX, are available from CTAN and the *TUGboat* web site. We also accept submissions using ConTeXt. Deadlines, tips for authors, and other information: <http://tug.org/TUGboat/location.html>

Suggestions and proposals for *TUGboat* articles are gratefully accepted. Please submit contributions by electronic mail to TUGboat@tug.org.

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

TUGboat editorial board

Barbara Beeton, *Editor-in-Chief*

Karl Berry, *Production Manager*

Boris Veytsman, *Associate Editor, Book Reviews*

Production team

William Adams, Barbara Beeton, Karl Berry, Kaja Christiansen, Robin Fairbairns, Robin Laakso, Steve Peter, Michael Sofka, Christina Thiele

TUGboat advertising

For advertising rates and information, including consultant listings, contact the TUG office, or see:

<http://tug.org/TUGboat/advertising.html>

TUG Institutional Members

American Mathematical Society,
Providence, Rhode Island

Aware Software, Inc.,
Midland Park, New Jersey

Center for Computing Sciences,
Bowie, Maryland

CSTUG, *Praha, Czech Republic*

Florida State University,
School of Computational Science
and Information Technology,
Tallahassee, Florida

IBM Corporation,
T J Watson Research Center,
Yorktown, New York

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

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, Wisconsin

Masaryk University,
Faculty of Informatics,
Brno, Czech Republic

MOSEK ApS,
Copenhagen, Denmark

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

Springer-Verlag Heidelberg,
Heidelberg, Germany

StackExchange,
New York City, New York

Stanford University,
Computer Science Department,
Stanford, California

Stockholm University,
Department of Mathematics,
Stockholm, Sweden

University College, Cork,
Computer Centre,
Cork, Ireland

Université Laval,
Ste-Foy, Québec, Canada

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

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

VT_EX UAB,
Vilnius, Lithuania

Ab Epistulis

Steve Peter

As I sit here writing this in the gentle hum of my overworked air conditioner, I can attest that summer has definitely reached the northern hemisphere. As the weather here has been heating up, so too has the planning for TUG 2013 in Tokyo.

Proposals are coming right now (the deadline is here, which focuses the mind), and from what I've seen so far, it looks to be an amazing conference, with a focus (as you would expect) on East Asian usage of TeX, but with plenty of content for TeX users of every level and linguistic requirement. Even if you aren't giving a presentation, consider joining us. The conference runs from October 23–26 at the University of Tokyo, Komaba, Tokyo, Japan. For full details, see the conference website at <http://tug.org/tug2013>.

Since I wrote for the last issue of *TUGboat*, we have had an official TUG election. I'm very pleased to be able to continue as president for another term (even if I am thinking of re-styling this as the vice president's column). On the board of directors, we have one director retiring, Jonathan Fine, and I'd like to thank him for his service on the board. Joining the board this year will be Arthur Reutenauer, whom many of you will know from various TeX meetings in Europe and abroad. I first met Arthur a few years ago in Cork, and was immediately impressed both with how smart he is, and how approachable. (And of course, anyone who knows him knows that he wears many hats, most of them quite stylish!) Returning to the board are Kaja Christiansen, Steve Grathwohl (a.k.a. *Steve₂*), Jim Hefferon, Klaus Hoeppner, and David Walden. Candidate statements and photos are online at <http://tug.org/election>.

The TeX Collection 2013 DVD has shipped to TUG members. I received my disc last week, and several friends have already installed the latest and greatest, and are playing around (re)discovering TeX and friends.

Board member Boris Veytsman continues to write prolifically. Now online at the TUG website are new book reviews covering *Presentations with LATEX* and *LATEX Quick Reference* by Herbert Voß, and *Learning LATEX* by David Griffiths and Desmond Higham. For these and other reviews, as well as discounts and more, see <http://tug.org/books>.

Until next time. Happy TEXing!

◊ Steve Peter
 president (at) tug dot org
<http://tug.org/TUGboat/Pres>

Editorial comments

Barbara Beeton

Barry Smith, 1953–2012

My acquaintance with Barry was bounded by two telephone calls. The second, last December from Doug Henderson, informed me of Barry's death in October 2012.

The first call was much more upbeat. Early one morning in 1982, I picked up the phone to hear an unfamiliar voice asking if it would be possible to output the camera copy for a book on the Math Society's Alphatype. It was Barry. The book in question was the manual for Oregon Pascal, as prepared in TeX80 by Barry and his partner at Oregon Software, Dave Kellerman. Barry reported the successful creation in his report in *TUGboat* 2:2 (page 34) as part of his report on TeX for VAX/VMS:

Well, it works—TeX for the VAX running VMS is alive, available, and in production use. (Production use is defined by example—we've just finished a 192 page manual for our optimizing PDP-11 Pascal compiler that is entirely typeset by TeX, including charts and diagrams.)

In the event, arrangements were made, and the output that emerged from the Alphatype was classy—much more attractive than other contemporaneous compiler manuals, which were often “typeset” on a line printer. The highlight, for me, was the section containing the “railroad diagrams”, syntax diagrams that presented in graphic form the Pascal grammar. These pages really tested the alignment of the Alphatype, to ensure that no rules (the drawn kind) were broken, and that they joined with the boxes at appropriate locations. Sadly, I've been unable to find an image of these pages on the web; they really looked spectacular.

Barry and David went on to form Kellerman & Smith, providing TeX for the VAX. Then the Mac arrived on the scene. Before Barry and David went their separate ways (Barry to form BlueSky, devoted to the Mac, and David, Northlake Software, continuing with VMS), they cooperated on one more notable project for TUG: guest editing an issue of *TUGboat* (7:1, 1986), including a new design by a professional designer and a special cover drawing by Duane Bibby, which was also featured on that year's meeting T-shirt. The issue is on the TUG web site—take a look.

It's hard to accept that Barry isn't here any more. His contributions were always pushing the boundaries that other people took for granted. He will be missed.

Yet another DEK interview

In Vienna on May 16, Don presented the first annual “Vienna Gödel Lecture” of the Faculty of Informatics at Vienna University of Technology.

The lecture was videoed, you can watch it at <http://www.informatik.tuwien.ac.at/english/vienna-goedel-lectures/2013>. After the lecture, Don submitted to an “All Questions Answered” session, linked from the same page.

Don also took part in an interview, entitled “I was born a geek”. The transcript (in German) can be found at <http://futurezone.at/digitallife/15926-donald-knuth-ich-wurde-als-geek-geboren.php>. The interview is linked from the TUG interviews web page.

TeXdoc on line

For those who have excluded most documentation from their TeX installation, but still need to look at package manuals from time to time, there is an online version of `texdoc` developed and maintained by “frequent contributors” to the site <http://www.texdoc.net/>. The server is maintained by Stefan Kottwitz, and the scripts by Paulo Cereda, both active contributors to `TeX.stackexchange`. The instructions are simple and the material well organized. Thanks, guys!

Fonts, typography, and printing— on the web and in print

“TypeRider” (<http://www.youtube.com/watch?v=3diZa7pmSys>) is a short video, the introduction to a game that “aims to revive interest in the history of typography through a multimedia world”. More details at <http://www.typerider.fr/>; release of the game is expected in September 2013.

Tibet: An Inner Journey, by Matthieu Ricard, has recently been reissued in paperback. One chapter is about a centuries-old print shop that uses engraved wooden blocks. Astonishingly, it survived the Chinese desecration of Tibet. It is still very active, printing both sacred Buddhist texts (in red) and secular material (in black). They also make some of their own paper using a native Tibetan plant. The text occupies only a few pages, but the story is illuminated by about a dozen pages of color photos. (Thanks to Elizabeth Tachikawa, the Unix TeX office person at the University of Washington in the 1980s.)

◊ Barbara Beeton
<http://tug.org/TUGboat>
 tugboat (at) tug dot org

In memoriam: Barry Smith (1953–2012)

Doug Henderson

Barry Gordon Smith

1 September 1953–8 October 2012

A sad day has come to all that knew Barry Smith. He passed away after a long battle with cancer. The doctors gave him six months, and, as was his way, he lived his life developing `Textures`, the Mac version of TeX, as if he would never do anything else—for almost another 2 years.

So optimistic he was in the last week that he didn’t even let on to those most dear to him how close he was to the end. He dearly loved Apple technologies and was very close to finishing an iOS version of `Textures` for the iPad.

I knew Barry as a coder like none other, and his energy for coding perfection burned like few that I have known in my life.

So it is with much regret that I inform everyone of Barry’s death. He was actively working on `Textures` for the Mac and iOS when he left us. He was strongly committed to making the best TeX system possible and chose the Macintosh platform before I came to work with him in 1989. His passing is a loss for us all.

His work is at an end.

Hyphenation exception log

Barbara Beeton

This is the periodic update of the list of words that *TEX* fails to hyphenate properly. The full list last appeared in *TUGboat* 16:1, starting on page 12, with updates in *TUGboat* 22:1/2, pp. 31–32; 23:3/4, pp. 247–248; 26:1, pp. 5–6; 29:2, p. 239; 31:3, p. 160; and 33:1, pp. 5–6.

In the list below, the first column gives results from plain *TEX*'s `\showhyphens{...}`. The entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document. The full list of exceptions, as a *TEX*-readable file, appears at <http://mirror.ctan.org/info/digests/tugboat/ushyphex.tex>. (It's created by Werner Lemberg's scripts, available in the subdirectory `hyphenex`.)

Like the full list, this update is in two parts: English words, and names and non-English words (including transliterations from Cyrillic and other non-Latin scripts) that occur in English texts.

Thanks to all who have submitted entries to the list. Here is a short reminder of the relevant idiosyncrasies of *TEX*'s hyphenation. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The T_EXbook*, page 454.) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by *TEX* except for “missing” hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, ‘analyses’ can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated in the same way regardless of usage.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, Unabridged.

Hyphenation for languages other than U.S. English

Patterns now exist for many languages other than U.S. English, including languages using accented and non-Latin alphabets. CTAN holds an extensive collection of patterns: see `language/hyphenation` and its subdirectories.

A group of volunteers led by Mojca Miklavec and Manuel Pégourié-Gonnard have created a comprehensive package of hyphenation patterns, called `hyph-utf8`; see <http://tug.org/tex-hyphen>.

The list — English words

ape-ri-odic	a-peri-odic
as-pheric	a-spher-ic
as-pher-i-cal	a-spher-i-cal
backscratch(er,ing)	back-scratcher(-ing)
bedrid-den	bed-rid-den
bigshot	big-shot
cacheabil-ity	cache-abil-ity
chancery	chan-cery
dou-bletalk	dou-ble-talk
droplet	drop-let
dystopia	dys-topia
elec-trophore-sis	elec-tro-pho-re-sis
elec-trophoretic	elec-tro-pho-ret-ic
ex-plana-tory	ex-plan-a-tory
facelift(s,ing)	face-lifts(-ing)
gazetteer	gaz-et-ter
ge-ome-ter	ge-om-eter
grandiose	gran-di-ose
halftone	half-tone
heinous	hei-nous
hy-phen-a-tion	hy-phen-a-tion
ir-ra-tional	ir-ra-tio-nal
leaflet	leaf-let
lift-off	lift-off
metaphor	meta-phor
metaphor-i-cal(ly)	meta-phor-i-cal(-ly)
metempsy-chosis	metem-psychosis
midafter-noon	mid-after-noon
quadri-lat-eral	quad-ri-lat-er-al
quadruped	quad-ru-ped
quadrupole	quad-ru-pole
reim-ple-men(t)s,ed	re-im-ple-men(t)s,ed
reim-ple-men-ta-tion	re-im-ple-men-ta-tion
re-nais-sance	ren-ais-sance
rooftop	roof-top
sce-neshift(er,ing)	scene-shift-er(-ing)
shoplift(er,ing)	shop-lift-er(-ing)

<code>sig-nage</code>	sign-age
<code>subn-ode(s)</code>	sub-node(s)
<code>triplet</code>	trip-let
<code>we-blog(s)</code>	web-log(s)
<code>weightlift(er,ing)</code>	weight-lift-er(-ing)

Names and non-English words used in English text

<code>Al-go-nquian</code>	Al-gon-quian
<code>Al-go-nquin</code>	Al-gon-quin
<code>Au-flage</code>	Auf-lage
<code>Got-tfried</code>	Gott-fried
<code>Hoe-fler</code>	Hoef-ler
<code>Ni-et-zsche</code>	Nietz-sche
<code>Py-ongyang</code>	Pyong-yang
<code>Werkzeuge</code>	Werk-zeuge

Some musings on misplaced hyphens

It should be obvious that text in a given language should be processed with the proper patterns for that language (and `\left,righthyphenmin`). Odd things will surely ensue when the patterns for some other language are in effect.

While not wishing to embarrass anyone, it is nonetheless a fact that the following hyphenations were found in the printed version of a recent collection. The format in which they are presented here is, for convenience, the same as that used for the “real” list. Do *not* use these for creating English patterns!

“Good bad examples”

<code>these</code>	the-se
<code>Lua-\TeX</code>	Lu-aTeX
<code>per-spec-tive</code>	perspecti-ve
<code>peo-ple</code>	pe-ople
<code>about</code>	abo-ut
<code>soft-ware</code>	so-ftware
<code>slightly</code>	sli-ghtly
<code>pack-age</code>	packa-ge
<code>make</code>	ma-ke
<code>pub-li-ca-tion</code>	pu-blication
<code>ap-pears</code>	appe-ars

◊ Barbara Beeton
<http://tug.org/TUGboat>
 tugboat (at) tug dot org

Running TeX under Windows PowerShell

Adeline Wilcox

When I need to use TeX with Microsoft Windows, I prefer compiling my (L)TeX code from the command-line shell called PowerShell to using cmd.exe.

After running, say, pdflatex mypaper.tex (file extension optional), correcting LATEX code with gVim (my editor of choice) and writing the file, typing r at the PowerShell prompt reruns the last command given to PowerShell.

Further, if a command has already been used in the current PowerShell session, executing PowerShell’s Get-History cmdlet gives its Id number. For example, if bibtex mypaper was previously run and the command Id is 8, BIBTeX can be rerun by typing the short command r 8.

At the PowerShell prompt, typing ‘Get-Help (cmdlet-name)’ works rather like the Unix man pages.

PowerShell also knows some Unix aliases. Instead of typing the PowerShell cmdlet

`Move-Item oldfile.tex newfile.tex`

the same thing can be done with

`mv oldfile.tex newfile.tex`

An object-based shell, PowerShell can be maddening to experienced Unix shell programmers. But as long as one does not attempt too much with PowerShell, it can still be a handy way to run LATEX.

◊ Adeline Wilcox
 Department of Veterans Affairs
 810 Vermont Ave, NW
 Washington, DC, USA
 adeline dot wilcox (at) va dot gov

Does TeX have a future?

Hans Hagen

1 Introduction

Making the transition from ConTEXt MkII to MkIV took a lot of time. In the process all kinds of code was evaluated, improved and, occasionally, removed. To some extent, the frozen state of MkII reflects the requirements of automated typesetting of the past two decades. Today, LuaTeX is advancing automated typesetting beyond what was previously possible. But do we really need it? In this article I will describe several issues we faced while rewriting the code, the choices, and compromises, we made. I will not attempt to answer the question whether TeX has a future, but merely offer you my own observations and thoughts.¹

2 Media

It is not hard to extrapolate the advance of e-books, and the demise, in some countries, of paper books. Less demand for printed books means less need for typesetting. Of course, real-time rendering is also typesetting. But since there is no one format compatible with all e-book readers, publishers are unlikely to produce multiple device-specific versions. To what extent is a shift in the way documents are encoded important for TeX development? And as publishers cut quality and costs in an attempt to stay alive, who will want high quality output? Personally, I think more and more authors will turn to self-publishing. In this respect we might see a revival of TeX and more complex typesetting. It all depends on how important a particular look and feel is, and what price you are willing to pay to achieve it. Nevertheless, we cannot deny the fact that times are changing, and that technological developments will influence how TeX-like systems evolve.

From the start ConTEXt could produce very complex interactive documents. But apart from its inclusion in several projects, this functionality has never been in any serious demand by the publishing world. One reason for this is that compared to the printed product, interactivity is seen as an additional ‘free’ feature. As we enter the age of electronic books, we see that the features commonly used are only a portion of those available. Nevertheless, all this accumulated functionality is available in MkIV. When a typesetter has an eye for quality, interesting typographic and navigational details will appear.

Originally presented at EuroBachoTeX 2013.

¹ This text was copy-edited for MAPS by Michael Gavravage, whom I gratefully thank for helping me express my thoughts.

3 Application

It is quite usual to find ConTEXt hidden in a larger publication workflow. In such cases the input comes from a database or some online editing environment. The layout, and therefore the typesetting, are often relatively simple. A predefined style tells ConTEXt how to transform input to output. The input may be predictable, but the user still has significant influence on the workflow. In this situation, what sets MkIV apart is its ability to analyze and manipulate data sets. MkII can also deal with data, but with Lua on board, MkIV solutions seem more natural. MkII is sufficient for traditional typesetting situations, but MkII is a dead end compared to MkIV.

We often talk of TeX users and user groups, but the more abstract term *usage* might be a better indicator of how much TeX is used. The number of TeX users is not growing, but TeX usage might be on the rise. Perhaps counting the number of pages produced with TeX is a better indicator of how prevalent TeX is than counting the number of installed TeX systems.

4 Coding

Another observation is that ConTEXt users often produce more advanced and demanding documents than I do as part of my work. For me, the biggest advantage of MkIV is its support for OpenType. Fonts are easier to install, and all those encodings disappear. Another advantage is that MkIV has a flexible XML processor built in, which can save you time in solving problems. Of course we continue to use and improve basic rendering capabilities, but we often have to simplify solutions when designers fail to see the possibilities of automated typesetting. Stability is often cited as the reason to use older combinations of TeX engines and macro packages. Ease of use and improved maintenance might be sufficient reasons to move on.

In the early days of ConTEXt my colleagues and I were its main users. One of the nice things with TeX compared to a word processor—never in my life have I had to use one—is that you can automate things. Imagine that you attend a series of meetings where several hundred learning objectives are identified, described, ordered and grouped. If you are in charge of such a task, it really helps to have a system where numbering and breaking pages comes for free. We were often able to get the adapted documents in the post within a few hours of leaving the meeting. The authors were impressed when, in the next stage of the project, we presented them with multiple professional looking documents derived from the same source. No other application could easily

handle 500 floating images on 300 pages without crashing. This was the time that using `\TeX` paid off for us. That was more than 15 years ago.

Such a `\TeX`-based workflow is a sequence of edit, run and preview cycles; steps recognizable to any old-time computer user. However, it is not something that newcomers, like our children, might deem usable. It's not 'what you see is what you get', but the more abstract process of 'what you key is what gets done'. Wrapping `\TeX` with a simpler interface would only hide its power, flexibility and charm. Then, you might as well use a word processor. Let's face it, using `\TeX` directly only pays off when the user can separate coding from rendering, wants to have full control and desires to be independent of hard coded solutions. Try explaining that to a twittering facebooking kid. Regardless of how we move from MkII to MkIV, the route from source to result remains the same, and so does the intended audience. Updating `\TeX` engines and macro packages will not increase `\TeX` usage.

For some of our ConTeXt projects, traditional paper-based books are complemented by content intended for the web. Consequently, the document source is often XML. We could encode documents using a `\TeX`-based coding, which, if they had the freedom to choose, would likely be more comfortable for authors to use. I wager that many authors who have used `\TeX` directly still prefer it as an input language. Though XML is a widely accepted input and storage format, it is not ideal for typesetting. XML is geared toward publishing on the web and is not as expressive as `\TeX`. However, reusing content is rare, so we needn't worry too much about encodings.

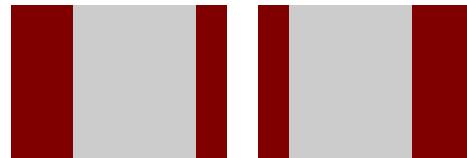
Coding in XML has some advantages for processing by `\TeX`. There are no `\TeX` commands for authors to misuse or redefine, and valid XML documents produce no errors. Another advantage is that styling and coding are completely separate. Of course, relieving the author of the responsibility of rendering complex documents can lead to sub-optimal output, unless the author is willing to adapt his content. The advent of XML has made people aware of the benefits of structure. ConTeXt tries to enforce structure, so `\TeX` can fit nicely into modern publication workflows. However, for the quick and dirty one-time documents, the overhead of adding structure might not be worth the effort. So, even if in MkIV we promote using `\startchapter` over `\chapter` and `\startitem` over `\item`, we keep supporting the less coding demanding variants.

The styles I write today are a mixture of `\TeX`, MetaPost and Lua. Solving the same problems with MkII, if possible, would require considerably more

effort. Just as the faster Internet has become natural, so has the MkIV mix.

5 Double-sided

An electronic medium is single-sided. A book is always double-sided, and in the case of magazines and newspapers also multi-column. ConTeXt has quite some code to deal with double-sided layout. Sometimes `\TeX` collects more content than can fit on one page. When this happens we have to keep track of where content should end up. For instance, dimensions and alignment conditions for margin notes must be swapped for odd and even pages.



In a single-sided universe, all the ConTeXt code that deals with `inner` and `outer` positioning and alignment could go away. Headers and footers could also be simplified. By removing the distinction between left and right pages, we could also drop some page synchronization code. Backgrounds wouldn't have to keep track of state either.

Related to this is page imposition. Page imposition is built into ConTeXt and is rather advanced. New imposition schemes occasionally appear through the effort of Willi Egger, who not only typesets but also prints and binds books. The advent of a new folded paper gadget can be the impetus for adding yet another variable to control the position of pages.

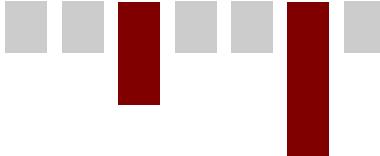
Some of our projects require that we produce imposed products as part of an automated workflow. Cover pages, combined with back pages, are on the agenda for future integrated support. Since these features are applied to finalized pages, implementing them is relatively easy, and they do not interfere much with existing code.

To separate content within electronic documents, we might end up with all sorts of cover-like pages. After all, additional e-pages are cheap, and color comes for free. This means that we might see more advanced page clustering and numbering schemes in ConTeXt MkIV. For instance, it might be nice if chapters had alternating or unique background colors. It would be even nicer if this property could be implemented without introducing new user commands in the source document.

6 Paper size

Paper books have standard page sizes; electronic books do not. Splitting tables with spans or large

cells is somewhat painful. So why should we split a large table in an e-document when we could just as well scroll?

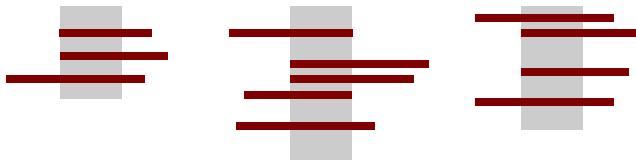


In a way we're going back in time. Long ago scrolls were used as a continuous medium. Thus, scrolling on a display is not as new as it may seem.

The concept of a page is derived from the medium—but what if we ignore this? For instance, if each chapter of a book were a separate entity, we could have one long page per chapter. This is problematic since \TeX sets a limit on how high a page can be. But imagine that instead of thinking vertically we go horizontal. Headers and footers go away or get a new meaning, and the edges would give some indication of where we were. Perhaps we need a floating indicator; we've seen stranger things. Would this require a programmable viewer that we could control from our document, or could we anticipate standard features in viewers and viewing devices? Luckily for us we can adapt the \TeX backend for either eventuality; at least we have done so for over three decades.

7 Floats

Floats are nice for paper. It is interesting to notice that in ConTeXt's early years floats were very prevalent in the documents we produced. In fact, they were a selling point. In educational documents especially, graphics need to appear near to where they are mentioned in the text. In a purely electronic document we needn't struggle with fitting graphics on a page. Relaxing this requirement would simplify designs. Removing the corresponding ConTeXt code would definitely make the codebase leaner and meaner. But don't worry, we have no plans to delete anything.



What if we combine the previously mentioned vertical layout with horizontal extensions? Again with a finger we swipe our way down the page, where we run into an indicator denoting a larger image. Swiping our finger to the left displays the image; which might be accompanied by texts, images or

animations. Another swipe and we're back in the main thread. It is amazing that we can do this with \TeX . In fact we can proceed to multi-dimensional or even parallel documents. I remember turning the Metafun manual into a QuickTime 360 movie. I must have a ConTeXt presentation style somewhere that implements this one page presentation where clicking on areas exposes different parts of the page. \TeX is and will always be a fine playground for such concepts. MkIV with Lua and MetaPost makes it even finer.

8 Margins

The first step from a paper document to, e.g., an e-book device, is to get rid of margins. Due to technical limitations all devices shipped around 2012 have rather hard-coded physical margins. Perhaps one day we will have devices that have matte displays running from edge to edge. Imagine a device without buttons, logos or stickers proudly mentioning the internal chip sets or operating system.

The current tendency is to remove margins. In the near future we might see them coming back. Margins provide structure, and also room for various indicators and navigation aids. This is a good thing. Support for putting things in margins is quite important. In MkIV we already go further than in MkII and more will come.

9 Accessibility

A table of contents still makes sense in an electronic document, but what about an index? An index's usefulness is proportional to how carefully it was prepared. In many cases a search option works just as well. The concept of a table of contents can be expanded to include local tables and navigation aids that help the reader find what he wants. Similarly, we can collect information in multiple indexes. We added multiple interactive indexes to ConTeXt while involved in a project that produced quality assurance manuals. In another project we needed index entries arranged in a linked list, which is why this functionality exists in MkII. This cross-linked variant is not yet available in MkIV only because I don't know anybody who needs it. Interestingly, implementing it in MkIV is far easier than in MkII.

A great deal of functionality, some of it even undocumented, is there because we once needed it. Take, for instance, flow charts. We can make really big ones. Selected cells can become hyperlinks—allowing us to jump through the document. Again, this functionality was a side effect of making those interactive QA manuals.

Mechanisms like these have always been part of ConTEXt, even when they make no sense for paper documents. They are more coding issues than demanding typographical challenges. They do not interfere with other typographical components, so simplifying or removing this functionality has no benefits. We can do much more in MkIV, but sometimes I get the feeling that less is more.

A lot of code in ConTEXt deals with structure. It makes sense to think about ways to improve how we gain access to it: linked lists, pop ups, summaries, reading routes, etc. MkII has several mechanisms that make controlled reading possible, but they never took off. In MkIV most mechanisms that structure data also retain part of it for re-use. Because we store data for use in a second or subsequent typesetting pass, information can be used multiple times.

Some mechanisms also support user data. For instance, when starting a chapter, besides setting its title, you can also name a variable that stores the name of an image—a sort of visual title. As this name is carried around, the image can appear as an icon in the table of contents and on the first page of the chapter. We needed this a long time ago in MkII. This is one reason why in MkIV we can now set user variables in commands that start chapters and sections.

In one project we participate in, a free math method, the content is first published on the web. Given the nature of electronic documents, it went unnoticed that, when typeset for the printed page, the document was quite large. Selective use of content, multiple products, and efficient typesetting are solutions to this. The e-book version is not constrained by the number of pages. Information can be repeated when needed; complemented with the necessary navigational aids. I'm confident that ConTEXt can deal with both variants.

There has been a time, probably due to the fact that I gave presentations showing PDF on a projector, that ConTEXt was promoted as a system for creating electronic documents in PDF format. This is just one feature, but interaction has always been integrated in the core—never an add-on. However, there is a fundamental difference between interaction in MkII and MkIV. Using different techniques in MkIV, we no longer have interfering status nodes. This makes the whole mechanism more robust, although internally it has become pretty complex.

10 Columns

Columns make sense in broadsheet newspapers and journals where one wants to put as much as possible on a page. But I wonder if columns make sense

in electronic documents. After all, electronic pages are cheap, and getting rid of multi-columns makes typesetting much easier. In TeX the mechanisms that deal with columns, e.g., page builder, floats and notes, are often complex. The code can be pretty messy. It would be nice to get rid of this legacy.

A good application of columns can be found in parallel bible translations. Not only must the text be synchronized in multiple columns, it also has to be broken across pages in a reasonable way. Footnotes are another complication.

Will such products be made in the future? The production of printed encyclopedias has already stopped, and concordances might soon follow. On the other hand, the fact that Thomas Schmitz typesets sophisticated documents for tablets, notebooks, projectors, and paper indicates that, for critical editions, the future is not yet determined. And I know several TeXies who typeset catalogs for conferences and festivals where a proper paper version is the only way to provide an effective overview. All these documents share a mixture of one column, multi-column and specially composed pages.

ConTEXt currently has two mechanisms that deal with columns. The first mixes well with single column mode, the second is more powerful and encapsulated. In MkIV the pluggability of the output routine has been improved; so if needed we can support yet unforeseen page building schemes. Parallel streams are first on the agenda.

11 Move on

If we consider only paper documents, do we anticipate needing more typesetting functionality than we already have? Does it make sense to develop macro packages any further? Of course, it is not difficult to make a wish list including more support for complex critical editions and parallel typesetting of translations. For those who use a simple input format such as Markdown, existing ConTEXt functionality is more than sufficient. In fact, as long as we can deal with the concepts found in HTML we're okay. Most of these documents consist only of running text, tables, images, a bit of sectioning, itemized lists and maybe descriptions.

TeX is over thirty years old. It is still maintained and kept up-to-date. It provides users with a lot of freedom. It has an active user community. It is often chosen for long term use. It is boringly stable. With these attributes, we can safely assume that TeX will be around for a while. The same is true for macro packages. They will stay and evolve. But how will TeX change along the inexorable path from paper to

electronic media? Typesetting habits change slowly, so we still have some time to ponder these questions.

On the other hand, look at how quickly the web is evolving, and how quickly younger generations adapt to new electronic devices. When using TeX it is natural to think in book-related categories. But just as a computer desktop is not a real desktop, an e-book is not a real book. Real books have a physical presence; we can hold them in our hands and turn each page as we read. E-books try to mimic these physical characteristics with ridiculous results. For example, you can choose an e-book from an e-bookshelf, and turning pages is simulated by showing the binding and a moving cut edge. But will we want or need these visual clues in the future when we have instant access to everything from anywhere on any device we choose? Why carry around a book when we can have its contents projected on our retina, or hear it spoken in our ear? Regardless of how TeX and its attendant macro packages evolve, we'd best refrain from predicting the future, let alone promote TeX as the ultimate and last word on typography. We can only hope that future hardware and software will allow us to TeX like we allow printers to use printing presses.

◊ Hans Hagen
<http://pragma-ade.com>

• <http://www.tug.org/texcollection> • 2013

TeXt: an easy to install TeX system for MS Windows: based on TeX, with the TeXstudio editor front-end.

Live: a rich TeX system to be installed on hard disk or a portable e such as a USB stick. Comes with support for most modern ms, including GNU/Linux, Mac OSX, and Windows.

TeX: an easy to install TeX system for Mac OSX: the full TeX Live button, with the TeXShop front-end and other Mac tools.

N: a snapshot of the Comprehensive TeX Archive Network, a set of rs worldwide making TeX software publically available.

TeXt ist ein einfache zu installierendes TeX-System für MS Windows, rend auf MiKTeX und TeXstudio als Editor.

Live ist ein umfangreiches TeX-System, zur Installation auf platte oder einem portablen Medium, z. B. USB-Stick. Binaries für Plattformen sind enthalten.

TeX ist ein einfache zu installierendes TeX-System für Mac OSX, mit n vollständigen TeX Live, sowie TeXShop als Frontend und weitere ramme.

N ist ein weltweites Netzwerk von ftp-Servern für TeX-Software. Auf NVD befindet sich ein kompletter Abzug des deutschen J-Knotens dante.ctan.org.

TeXt : un système TeX pour Windows facile à installer, basé sur TeX avec l'éditeur TeXstudio.

Live : un système TeX complet qui peut être installé sur disque dur mode portable sur une clé USB. Fonctionne sur la plupart des mes modernes, dont GNU/Linux, Mac OSX et Windows.

TeX : un système TeX facile à installer pour Mac OSX. Il comporte distribution TeX Live complète ainsi que l'éditeur TeXShop et res outils pour Mac.

N : une copie du *Comprehensive TeX Archive Network*, le réseau de ours assurant la distribution publique de TeX et ses amis dans le entier.

TeX Collection 2013

DVD

June 2013

dante e.V.
www.dante.de

GUTenberg
gutenberg.eu.org



www.tug.org



proTeXt

TeX for MS Windows
 based on MiKTeX

MacTeX

TeX for Mac OSX
 including full TeX Live

TeX Live

TeX for GNU/Linux, Unix,
 and MS Windows

CTAN

Comprehensive TeX
 Archive Network

Editors: Thomas Feuerstack (proTeXt) • Karl Berry (TeX Live)
 Richard Koch (MacTeX) • Manfred Lotz (CTAN)

TeX Collection 2013 DVD

TeX Collection editors

The TeX Collection is the name for the overall collection of software distributed by the TeX user groups each year. Please consider joining TUG or the user group best for you (<http://tug.org/usergroups.html>), or making a donation (<https://www.tug.org/donate.html>), to support the effort.

All of these projects are done entirely by volunteers. If you'd like to help with development, testing, documentation, etc., please visit the project pages for more information on how to contribute.

Thanks to everyone involved.

1 proTeXt (<http://tug.org/protext>)

proTeXt is a TeX system for Windows, based on MiKTeX (<http://miktex.org>) and TeXStudio (formerly known as TeXMakerX) (<http://texstudio.sf.net>) as corresponding editor.

For 2013, proTeXt now has a standard application program (`Setup.exe`) which is accompanied by a detailed document to guide your installation (and which, due to the simplicity of the installation process, you'll probably never need).

proTeXt currently has English, German and French as possible installation languages. Volunteers to make additional translations are most welcome.

2 MacTeX (<http://tug.org/mactex>)

MacTeX is a TeX system for Mac OS X, installing TeX Live and additional Mac-specific tools. MacTeX 2013 runs on both Intel and PowerPC machines and requires at least Mac OS X 10.5 (Leopard). It runs on Mac OS X Leopard, Snow Leopard, Lion, and Mountain Lion. The package installs the full TeX Live 2013, Ghostscript 9.07, the `convert` utility from ImageMagick 6.8.3-3, and the current versions of BibDesk, IATEXiT, TeX Live Utility, TeXShop, and TeXworks, as well as the TeX Dist Preference Pane, which allows users to switch easily between different TeX distributions. After installation, PATH is correctly set for shells, and all applications are configured and ready to run.

The MacTeX 2013 install package as obtained over the Internet works exactly as it has in the past, but the DVD installation has been changed to be more robust and provide more feedback. First, users open Terminal (from `/Applications/Utilities`) and copy two lines of text to this program from a short *MacTeX Install Part 1* document. This step installs a complete TeX Live from the DVD without asking any questions. Second, users run a program which installs everything else and configures the sys-

tem. The third part is only required of users running the older Leopard or Snow Leopard, and adds a GUI front end which runs on these systems.

The Collection also includes MacTeXtras (<http://tug.org/mactex/mactextas.html>), containing many additional items that can be separately installed. On the 2013 DVD, software that runs exclusively on Tiger (Mac OS X 10.4) has been removed, along with software that is installed by the MacTeX DVD installer. The main categories are: bibliography programs; alternative editors, typesetters, and previewers; equation editors; DVI and PDF previewers; and spell checkers.

3 TeX Live (<http://tug.org/texlive>)

TeX Live is a comprehensive cross-platform TeX system. It includes support for most Unix-like systems, including GNU/Linux and Mac OS X, and for Windows. Major user-visible changes in 2013:

- The `texmf` tree merged into `texmf-dist`; language collections merged.
- XeTeX: see following article by Khaled Hosny.
- LuaTeX: updated to Lua 5.2; new library to process external PDF page content.
- MetaPost: native support for PNG output and floating-point (IEEE double) added.
- xdvi: now uses FreeType instead of t1lib for rendering.
- tlmgr: new pinning action to ease configuration of multiple repositories.
- Platforms: added or revived armhf-linux, mips-irix, amd64-netbsd, i386-netbsd; removed powerpc-aix. Also, some platforms are now omitted from the DVD (to save space), but can be installed normally over the net.

More details are available in the TeX Live manual and web pages.

4 CTAN (<http://www.ctan.org>)

CTAN is the Comprehensive TeX Archive Network, a set of servers worldwide making TeX software publicly available.

As usual, the CTAN snapshot was made from the German node (<http://dante.ctan.org>) and omits the other components of the Collection. It is available to TUG members (and joint members) from the TUG members area, <https://www.tug.org/members>.

- ◊ TeX Collection editors
Thomas Feuerstack (proTeXt),
Dick Koch (MacTeX),
Herb Schulz (MacTeXtras),
Karl Berry (TeX Live),
Manfred Lotz (CTAN)
<http://tug.org/texcollection>

What is new in X_ET_EX 0.9999?

Khaled Hosny

One of the strengths of X_ET_EX is the use of external libraries from the underlying system as well as from third parties to do the heavy lifting of supporting modern font technologies and text layout requirements, as well as various aspects of Unicode support. Unicode and modern fonts support can be hard to get right and requires a great deal of time and effort; by using those external libraries we can build upon already existing work in these areas. For OpenType layout we were using the ICU Layout Engine and SilGraphite for Graphite layout. On Mac OS X we were using Apple's Font Manager API for locating fonts, ATSUI for AAT layout and QuickTime for loading images. On other systems we were using FreeType and FontConfig for loading and locating fonts, and Poppler for PDF images.

But all this is not without a cost; depending on external libraries requires continuous maintenance effort to catch up with changes in these external dependencies, if we want to remain relevant. So version 0.9999 of X_ET_EX saw a long overdue update to the underlying libraries.

When OpenType support in X_ET_EX was first introduced in 2006, the ICU Layout Engine was the best, cross-platform, free software choice at the time, though it had a limited API and was also missing several features that are vital for X_ET_EX, so we were using (and maintaining) a locally patched version for our own use.

However, over the past few years the ICU Layout Engine has become unmaintained and many bugs have crept into it, while in the meantime the new and more widely-supported HarfBuzz library has emerged and reached a mature stage. So, for 0.9999 I worked on porting X_ET_EX to HarfBuzz, which gives us a maintained, more complete and more widely supported layout engine¹ with less maintenance burden and simpler code on our side.

The switch to HarfBuzz also fixed some long-standing OpenType-related bugs in X_ET_EX, such as support for version 2 Indic OpenType specifications, or the ability to activate and deactivate any OpenType feature for any script (previously limited to scripts that did not require any specific shaping behaviour, such as Latin or CJK), and many other smaller issues, thanks to the first tier OpenType support provided by HarfBuzz and its versatile API.

¹ Coincidentally, after we finished the HarfBuzz port, ICU developers issued a statement that “*users of ICU Layout are strongly encouraged to consider the HarfBuzz project as a replacement for the ICU Layout Engine.*”

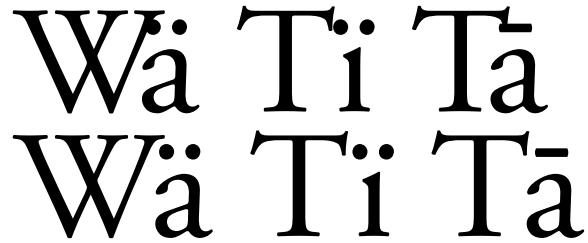


Figure 1: Some kerning bugs with Adobe fonts that were fixed after the switch to HarfBuzz.

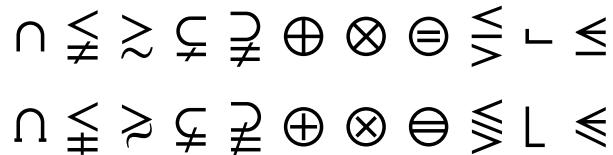


Figure 2: Some variants of mathematical symbols in the XITS Math font (in text mode).

For example, some kerning improvements are shown in fig. 1.

Another benefit we get from using HarfBuzz is support for Unicode Variation Selectors. Variation selectors are a way to represent variant glyphs for certain Unicode code points without encoding them separately or relying only on higher level protocols, like OpenType alternates, to represent them; unlike font alternates, valid variants are defined by Unicode. Variation selectors are used to encode CJK glyph variants (called Ideographic Variation Selectors), certain Mongolian contextual forms that can't be inferred from surrounding characters alone, and even stylistic variants of some mathematical symbols (however, X_ET_EX does not currently support variation selectors in math mode), as shown in fig. 2.

ICU is, however, a general library for Unicode support, so we still use it for other features like encoding conversion and the optional locale-aware line breaking.

On the Graphite front, the old SilGraphite engine has been rewritten as Graphite2 to provide a more robust implementation that is optimized for the actual use cases of Graphite than what was envisioned when the original engine was written. So, in 0.9999 we moved away from the old SilGraphite engine; the layout is now done by HarfBuzz (which in turn uses Graphite2), so we have a more unified interface at the code level, but we still call Graphite2 directly for Graphite-specific line breaking support, as well as for the primitives that query Graphite features. We now also support 4-character feature tags in Graphite fonts.

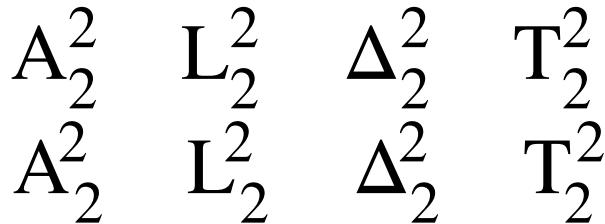


Figure 3: Before and after support for OpenType math cut-ins. (This example was artificially created to show the extremes of cut-ins.)

The situation on Mac OS X was even worse. All the Mac-specific APIs that we were using have been deprecated for several OS releases; furthermore, when Apple moved to 64-bit (`x86_64`) architecture, they didn't port the deprecated frameworks to the new architecture, forcing \XeTeX to always be built as a 32-bit application, causing all sorts of build problems and complications, and raising uncertainty about \XeTeX 's future on Mac (the platform it was originally written for!) in the event of Apple dropping support for 32-bit architecture—which is not unexpected.

Fortunately, Jiang Jiang kindly offered to port \XeTeX to Core Text and other new and supported frameworks, and did all the essential work for this port. Thanks to him we no longer depend on any deprecated Mac OS X libraries and \XeTeX future on Mac OS X has been secured.

One regrettable side effect of the Mac OS X updates is that the new \XeTeX is no longer compatible with `xdv2pdf` output driver and thus that driver has been dropped from \TeX Live. Remaining \XeTeX features that require `xdv2pdf` will be dropped in the next version.

Another small but important update in this release is the support for OpenType math cut-ins (see fig. 3) which provide for finer control of the horizontal placement of sub- and super-scripts than can be achieved with the (ab)use of italic corrections in traditional \TeX math fonts. With this implemented, \XeTeX now supports all major features of OpenType's MATH table, putting it on par with the Microsoft Office and \LaTeX implementations.

One more notable change in this release is that the Unicode math primitives have been renamed to use the \U-prefix instead of the \XeTeX-prefix, for better compatibility with \LaTeX that has the same primitives with the \U-prefix. Old names are still allowed, but might be removed in the future.

Other miscellaneous changes in this release include: preferring OpenType or TrueType over Type 1 fonts with the same name when `FontConfig` is used; proper printing of multi-byte characters to the log and terminal; and proper handling for characters outside Unicode's basic multilingual plane (BMP) in `\show`, `\meaning` and `\showlists` primitives as well as in tracing output.

One last important bug that was fixed (actually in version 0.9998) is the occasional mismatch between the font used by \XeTeX and the output driver `xdvipdfmx` when multiple versions of the same font are installed, which would often result in garbage output or prevent the driver from producing any output at all.

My work on \XeTeX has been supported by the general TUG and Mac \TeX development funds.

◊ Khaled Hosny
<http://xetex.sourceforge.net>

MetaPost: PNG output

Taco Hoekwater

Abstract

The latest version of MetaPost (1.80x) has a third output backend: it is now possible to generate PNG bitmaps from directly within MetaPost.

1 Introduction

For one of my presentations at EuroTEX 2012 in Breskens, I wanted to create an animation in order to demonstrate a MetaPost macro that uses timer variables to progress through a scene.

While working on that presentation, it quickly became obvious that the ‘traditional’ method of creating an animation with MetaPost by using ImageMagick’s `convert` to turn EPS images into PNG images was very time-consuming. So much so that I managed to write a new backend for MetaPost while waiting for ImageMagick to complete the conversion.

2 Simple usage

MetaPost will create a PNG image (instead of EPS or SVG) by setting `outputformat` to the string `png`:

```
outputformat := "png";
outputtemplate := "%j-%c.%o";
beginfig(1);
  fill fullcircle scaled 100 withcolor red;
endfig; end.
```

This input generates a bitmap file with dimensions 100x100 pixels, with 8-bit RGBA color. It shows a red dot on a transparent background.

3 Adjusting the bitmap size

In the simple example given above, MetaPost has used the default conversion ratio where one point equals one pixel. This is not always desired, and it is tedious to have to scale the picture whenever a different output size is required.

To allow easy modification of the bitmap size independent of the actual graphic, two new internal parameters have been added: `hppp` and `vppp` (the names come from Metafont, but the meaning is specific to MetaPost).

In MetaPost, ‘`hppp`’ stands for ‘horizontal points per pixel’; similarly for ‘`vppp`’. Adding ‘`hppp:=2.0;`’ to the example above changes the bitmap to be 50x100 pixels. Specifying values less than 1.0 (but above zero!) makes the bitmap larger.

4 Adjusting the output options

MetaPost creates a 32-bit RGBA bitmap image, unless the user alters the value of another new internal parameter: `outputformatoptions`.

The syntax for `outputformatoptions` is a space-separated list of settings. Individual settings use `<keyword>=<value>` syntax. Currently supported are:

```
format=[rgba|rgb|graya|gray]
antialias=[none|fast|good|best]
```

No spaces are allowed on either side of the equals sign inside a setting.

The compiled-in default could be given as:

```
outputformatoptions
  := "format=rgba antialias=fast";
```

However, the `outputformatoptions` variable value itself is initially the empty string, because that makes it easier to test whether a user-driven change has already been made.

Some notes on the different PNG output formats:

- The `rgb` and `gray` subformats have a white background. The `rgba` and `graya` subformats have a transparent background.
- The bit depth is always 8 bits per pixel component.
- In all cases, the current picture is initially created in 8-bit RGB mode. For the `gray` and `graya` subformats, the RGB colors are reduced just before the actual PNG file is written, using a standard rule:

$$gray = 0.2126 * r + 0.7152 * g + 0.0722 * b$$

- CMYK colors are always converted to RGB during generation of the output image using:

$$r = 1 - (c + k > 1 ? 1 : c + k)$$

$$g = 1 - (m + k > 1 ? 1 : m + k)$$

$$b = 1 - (y + k > 1 ? 1 : y + k)$$

If you care about color conversion, you should do a `within <pic>` loop inside `extra_endfig`. The built-in conversions are intended as a fallback.

5 What you should also know

MetaPost uses Cairo (<http://cairographics.org>) to do the bitmap creation, and then uses libpng (<http://www.libpng.org>) to create the actual file.

Any `prologues` setting is always ignored: the internal equivalent of the `glyph of` operator is used to draw characters onto the bitmap directly.

If there are points in the current picture with negative coordinates, then the whole picture is shifted upwards to prevent things from falling outside the generated bitmap.

◊ Taco Hoekwater
<http://tug.org/metapost>

Converting Wikipedia articles to L^AT_EX

Dirk Hünniger

Abstract

It is often desirable to have access to Wikipedia articles in L^AT_EX format. A translation by hand is typically time-consuming and error-prone. Thus it is natural to look for algorithmic solutions to this problem. Our solution is currently available free of charge under an open source license for Windows and Debian GNU/Linux. It is not limited to Wikipedia but supports all servers running the same wiki software (MediaWiki) as Wikipedia. In particular, it is also possible to process local wikis available only on private networks inside institutions.

1 Introduction

A wiki provides a very convenient way of working on a document with many contributors, without needing to learn the details of specialized version control and typesetting software. MediaWiki provides a function to export PDF files. But the possibilities for incorporating individual requirements on the output layout are very limited and usually insufficient for professional publishers. Also the typographic quality of the output is far less elaborate than what is provided by L^AT_EX. Furthermore, the embedding of formulas as raster graphics is often criticized.

2 User experience

In the default mode, our program takes a url to a web page on a MediaWiki server and writes a PDF version of that page generated with L^AT_EX to local hard disk. It is also possible to retrieve the corresponding L^AT_EX source code, including images.

Also in the default mode, the HTML generated by the MediaWiki server is evaluated. There is also an extended mode where the source code of the wiki page written in the wiki markup language is processed. The wiki markup language provides a mechanism similar to the L^AT_EX \newcommand directive, called “templates”. In this mode it is possible to map templates to L^AT_EX commands and implement them using \newcommand or similar methods in the headers. This mechanism provides a fine-grained control over the conversion process and thus gives the user the full flexibility of L^AT_EX.

3 On the history of the problem

Quite a few attempts have been made to tackle this problem programmatically. We would like to emphasize the successful work of Hans Georg Kluge, who modified MediaWiki’s original parser to produce L^AT_EX (<http://code.google.com/p/wiki2latex>).

Unfortunately it needs to be installed on the server running the wiki in order to run and Wikipedia is currently not attempting to install it. This is partly because the security of the code is currently being discussed, which is particularly a concern since it is written entirely in PHP.

There have also been several attempts approaching the problem with regular expression or Backus-Naur forms. Recently we were able to provide a simple proof, based on the pumping lemma, that improper bracketing of HTML tags, as often found on Wikipedia, causes the grammar to no longer be context free, thus rendering it indescribable by Backus-Naur forms and regular expressions. This, in turn, rules out most standard parsing technology.

In our approach, we run all software on the user machine, thus bypassing any security concerns of Wikipedia. We opted for monadic parser combinators as parsing technology, and were able to handle the non-context-free grammar well with that approach.

4 Technical details of the implementation

The program is entirely written in the purely functional language Haskell. To do the necessary image processing the ImageMagick library is used. We currently use X_QL^AT_EX as the default compiler, although we recognized that the source (with tiny changes limited to the headers only) does also compile with pdfL^AT_EX and LuaL^AT_EX.

Currently there still is no freely available font that covers the whole range of Unicode. A problem in this respect is also that certain code points used for some Asian characters are used for more than one symbol and Wikipedia does not always provide a means to find out which symbol is actually meant by a Unicode character. For now we use FreeSerif as the default font, which omits Asian glyphs entirely. So we also offer a computationally combined font, made of several fonts available under the same open source license that actually covers the full Unicode range. In pdfL^AT_EX we use just this one font with the CJK package and thus can handle the first 16 bits of the Unicode range. This approach allows the user to still use custom fonts like Utopia, Courier, etc. For X_QL^AT_EX we provide a set of fonts for bold, italic, typewriter, small caps, and combinations thereof. This approach basically works also with LuaL^AT_EX, but unfortunately caused huge memory and CPU usage in our tests.

◊ Dirk Hünniger
http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger/wb2pdf
 dirk dot hunniger (at) googlemail dot com

A survey of text font families

Michael Sharpe

Abstract

This is a survey of text font families, both free in some sense and commercial, that might be considered suitable for general text use within L^AT_EX, seeking to tabulate the qualities that matter most there, as well as X_L^AT_EX and Lua_L^AT_EX.

Introduction

In a *TUGboat* article twenty years ago, Berthold Horn [1] noted that there were over 14,000 fonts in Type 1 format but only a handful of T_EX math fonts to accompany them. There are now considerably more choices of math fonts than there were then, and it seems appropriate to ask instead *where are the text fonts?* It's not that there are fewer than there were in the 1990s, but our typographic expectations are higher than they used to be. Moreover, the appropriate count should be text font families rather than individual faces, and for most purposes, one should count only serifed font families, as those are the only serious candidates for the main text family where the output may include paper or PDF.

For L^AT_EX usage, the current minimal standard for a text font family is, in my opinion:

- (A) upright and italic shapes in both regular and bold weights (four styles);
- (B) real (i.e., not faked by reducing capital letters) SMALL CAPS in upright regular weight;
- (C) full set of common f-ligatures—f_i, f_l, f_f, f_f_i, f_f_l (fi, fl, ff, ffi, ffi) in each style;
- (D) oldstyle figures 0123456789 in regular weight, upright shape.

Both Computer Modern and its modernized form Latin Modern (lmodern) meet these expectations, and more, but many fonts derived from legacy PostScript fonts do not. Most commonly, (C) fails, but (B) and/or (D) may also be lacking, and in some cases, (A) fails, usually because there is no **Bold Italic**.

A more demanding user would likely raise the bar to the following stronger conditions:

- (A') upright and italic shapes in two weights (four styles) and preferably three weights (six styles) such as regular, semibold and bold;
- (B') real SMALL CAPS are provided in all upright styles, and preferably in all styles;
- (C') full set of common f-ligatures—f_i, f_l, f_f, f_f_i, f_f_l (fi, fl, ff, ffi, ffi) in each style;

- (D') oldstyle figures 0123456789 in all styles.
- (E') other figure styles (e.g., proportional lining, tab oldstyle, superior) in at least upright regular style.

The free fonts considered in this article are mostly available from CTAN and the commercial fonts are mostly from the current Adobe Folio. Only fonts with serifs are considered, as those are overwhelmingly the most common main text font types except when output is intended on a low resolution screen, where sans serif, or perhaps a slab serif, renders more clearly. In most cases, the fonts are in OpenType format, which may be used directly by Lua_L^AT_EX and X_L^AT_EX, and which may be converted using utilities such as `otfinst` or `autoinst` to PostScript font families with L^AT_EX support files. It seems that there are now close to thirty font families, many of them free, which come very close to satisfying conditions (A')–(E').

I don't have licenses for most of the commercial fonts listed below, and in those cases I've relied on information from the web site <http://www.myfonts.com>, from which one may obtain glyph lists and other essential information about most commercial fonts. To search manually, go to the site and follow **Find Fonts** → **Search**, and enter the font name, e.g., **Goudy Oldstyle**, which leads to a screen with broad matching to that name. If you select '**Goudy Oldstyle family of 5 fonts from Adobe**' you reach a screen showing the five individual fonts. Press the first (Regular weight, upright shape) to see a selection of its glyphs. Press **Glyphs** to see the entire glyph catalog for that selection, from which you may determine that Regular weight, upright shape has oldstyle figures, small caps, only f_i and f_l ligatures, and a limited selection {1, 2, 3} of superior figures. (This is more or less typical for fonts derived from older PostScript fonts.) Examining the other variants establishes that they all have oldstyle figures, but none has small caps.

While in the screen showing all glyphs, click on a letter to bring up an enlarged image, which may be saved for further examination. The glyph images are drawn from anti-aliased .gif bitmaps which seem to have been made at the scale 1px = 3em, which is handy for estimating the vertical stem widths, which provides information about the relative weight of a font. The information provided in the tables below comes from these estimated values and from values obtained from FontForge for fonts to which I own licenses.

Table 1: Free (at least of cost) fonts, in approximate order of heaviness (VStemW)

Name	Source	fLigs	Smc	VStemW	OsF	OF	XH	CH	IA	Notes	TL
quattrocento	impallari	2		70/113			459	660	-13	28	✓
kpfonts	public	A	A	73/89/117/135	A	A	441	670	-11	10, 20	✓
antt	public	A	A	75/97/118/143	A	A	473	703	-9.5	17	✓
EBGaramond	public	A	RI	80		2	405	656	-17	26	✓
GFSBodoni	public	A	R	84/117		A	476	705	-12	1, 24	✓
venturis	arkandis	A	A	84/139/178		A	432	643	-16	16	✓
LinLibertine	public	A	A	85/123/140	A	A	431	647	-12	14	✓
GFSArtemisia	public	A	R	85/132		A	470	692	-12	22	✓
Computer Modern	public	A	A	89/144		A	431	683	-14	1, 10	✓
Latin Modern	public—GUST	A	A	89/144		A	431	683	-14	1, 3, 10	✓
garamondx	URW—AFPL	A	A	91/133		A	426	692	-16	2, 14	○
garamond	mathdesign	2		91/133			426	692	-16	10, 21	○
mathpazo	public	A	R	96/141		R	459	689	-10	5, 10	✓
Pagella	T _E X Gyre	A	A	96/141		A	459	689	-10	5, 13, 19	✓
newpxtext	public	A	A	96/141		A	459	689	-10	5, 18, 19	✓
pxfonts	public	A	RB	96/141		A	459	689	-10	5, 10, 13	✓
PT Serif	public	2		96/150			500	700	-12	25	✓
fourier	GUT	2		99/160			490	693	-13	9, 10, 12	✓
kerkis	public	A	RB	99/117/174		RB	485	681	-10.3	4, 10	✓
utopia	mathdesign	2		99/160			490	693	-13	10, 21	✓
GFSDidot	public	A	R	100/140		A	456	689	-12	5, 23	✓
Bonum	T _E X Gyre	A	A	100/176		A	485	681	-10.3	4, 13	✓
charter	mathdesign	2		102/145			488	679	-11	10, 21	✓
CharisSIL	SIL	A	A	102/145			488	679	-11	8, 11	○
mathptmx	public	2	RB	102/162		A	450	662	-15.5	7, 10, 12, 13	✓
newtxtext	public	A	A	102/162		A	450	662	-15.5	7, 14	✓
Termes	T _E X Gyre	A	A	102/162		A	450	662	-15.5	7, 13, 14	✓
baskervald	arkandis	A		103/153/180			415	667	-16	15	✓
librebaskerville	impallari	A		104/146		A	530	770	-15	27	✓
Schola	T _E X Gyre	A	A	112/180		A	466	722	-15	6, 13	✓

Keys to font tables

In the font property tables, the following abbreviations are used:

fLigs indicates the type of f-ligatures available: A indicates that all (fi, fl, ff, ffi, ffl) are available in all variants, and 2 indicates that only the two basic ones (fi, fl) are provided;

Smc indicates availability of real small caps: A indicates all variants, R indicates only regular weight, upright shape, RI indicates regular weight, upright and italic shapes, RB indicates regular and bold weights, upright shape only, and blank indicates none;

VStemW indicates the vertical stem widths (in em units, which in most cases is $100\text{em} = 1\text{pt}$) of each weight available in an upright shape—these provide one simple measure of the relative weights of fonts, though other factors such as contrast (ratio of widest to narrowest stems) and side-bearings play a rôle as well;

OsF indicates availability of oldstyle figures: A means all variants have oldstyle figures available as the default text figures, R means they are available only in regular weight, upright shape, and blank means they are not available at all;

OF indicates, if A, that other figures sets are available: e.g., superior figures other than {1, 2, 3}, or proportional figures other than oldstyle;

XH gives the x-height in em units;

CH gives the cap height in em units;

IA gives the italic angle, e.g., -10 means slanted 10 degrees clockwise from vertical;

Notes are given after the tables;

TL indicates whether the font is included in T_EX Live.

Table 2: Commercial fonts

Name	Source	fLigs	Smc	VStemW	OsF	OF	XH	CH	IA	Notes
StempelSchneidlerStd	Adobe	2		50/68/96/151/184			450	715	-12	B
ChaparralPro	Adobe	A	A	55/80/120/172	A	A	420	650	-10	H
CaeciliaLT	Adobe	2	A	55/86/113/147	A		516	699	-5	F
BriosoPro	Adobe	A	A	57/78/86/108/131	A	A	405	622	-10	E
RockwellStd	Adobe	2		57/102/176/264			472	679	-13	B, F
GaramondPremierPro	Adobe	A	A	60/83/90/119/140	A	A	393	646	-18	K, Q
ArnoPro	Adobe	A	A	61/84/124/142	A	A	398	618	-11	D
UsherwoodStd	Adobe	2		63/83/114/168			467	627	-12	B
NovareseStd	Adobe	2		64/99/149/218			460	640	-12	B, O
KinesisStd	Adobe	A	A	66/84/114/132	A	A	439	629	-6	M
HorleyOldStyleMTStd	Adobe	2		66/85/106/142			419	705	-9	B
BerkeleyStd	Adobe	2		66/87/116			426	635	-8	B
CentaurMTStd	Adobe	A	R	66/100		A	363	631	-13	G
VersaillesLTStd	Adobe	2		67/93/141/206			496	712	-12	B
WeidemannStd	Adobe	2		69/97/129/160			507	711	-12	B
WarnockPro	Adobe	A	A	76/90/129/142	A	A	440	659	-15	
KeplerStd	Adobe	A	A	73/98/132/158/180	A	A	430	634	-13	L
MaiolaPro	Adobe	A	A	73/113		A	414	611	-11	D
PerpetuaStd	Adobe	A	R	76/129		A	353	573	-12	P
VeljovicStd	Adobe	2		77/110/150/204			452	626	-12	B
TiepoloStd	Adobe	2		77/111/148			469	614	-9	B
GoudyStd	Adobe	2	R	79/123/152/243	A		418	704	-7	J
LegacySerifStd	Adobe	2		79/105/141/183			422	635	-12	B
DanteMTStd	Adobe	A	R	80/104/124	A	A	404	596	-9	I
FairfieldStd	Adobe	2	R	80/111/150/201	A	A	418	678	-9	
WeissStd	Adobe	2		82/108			407	694	-8	B
BemboStd	Adobe	A	R	82/111/140/168	A	A	396	622	-11.5	D
HiroshigeStd	Adobe	2		82/113/148/193			504	692	-9	B
BellMTStd	Adobe	A	R	84/105/137		A	410	644	-16	C
MeridienLTStd	Adobe	2		85/114/150			460	634	-12	B
MinionPro	Adobe	A	A	85/116/134	A	A	437	650	-12	
JensonPro	Adobe	A	A	86/113/127	A	A	388	649	-8	D
LeawoodStd	Adobe	2		86/134/183/207			554	709	-12	B
MinsterStd	Adobe	2		87/121/167/228			456	722	-10	B
Berling	Adobe	2		88/126			447	709	-12	B
GalliardStd	Adobe	2		88/132			442	680	-14	B
MeliorLTStd	Adobe	2		90/148			465	692	-12	B
StempelGaramond	Adobe	2	R	91/134	A		429	698	-16	R
SouvenirStd	Adobe	2		92/148/183/239			473	732	-10	B
CaslonPro	Adobe	A	R	93/127/150	A	A	420	711	-22	Q
StoneInformalStd	Adobe	2		96/140/212			500	700	-12	B
StoneSerifStd	Adobe	2		97/140/211			500	700	-12	B
TrumpMediaeval	Adobe	2	R	98/146	A		477	698	-12	
SabonLTStd	Adobe	2	R	99/128	A		442	698	-12	Q
UtopiaStd	Adobe	A	RB	99/141/164/224	A	A	461	653	-13	S
NewCaledoniaLTStd	Adobe	2	RB	100/129/169/220	A		422	664	-12	A
JansonText	Adobe	2	R	100/157	A		440	711	-15	A
LucidaOT	TUG	A	RB	104/150	A		530	723	-11.25	N
NewBaskervilleStd	Adobe	2	RB	105/152	A	A	427	660	-16	A, T
NewAsterLTStd	Adobe	2		111/138/178/228			464	692	-16	B
TiffanyStd	Adobe	2		114/149/295			449	715	-13	B

Notes on free fonts

1. High contrast (ratio of widest stems to narrowest).
2. Scale down about 5%.
3. Extension of Computer Modern.
4. Extension of URW version of Bookman.
5. Extension of URW version of Palatino.
6. Extension of the URW version of New Century Schoolbook. This is the font to use for briefs to the SCOTUS. The package `fouriernc` pairs it with `fourier` math, should you wish to improve your case with mathematical arguments.
7. Extension of URW version of Times.
8. Extension of Bitstream Charter.
9. Extension of original Utopia, donated to TUG by Adobe. Can use expert fonts, if available (not free), for OsF and real small caps. (Venturis is another option.)
10. Text and math fonts included.
11. Not on CTAN, download from sil.org. Lacks kerning tables.
12. Fake small caps.
13. Oldstyle figures available, but no option to designate them as the default text figures.
14. Can use `newtxmath` as math package.
15. Similar to Baskerville. Lack of small caps and OsF is a drawback to serious L^AT_EX use. Math and tabular usage is problematic because no tabular figures are provided.
16. Based on Utopia, but not as heavy. Full-featured. Can use `fourier` for math.
17. Antykwa Toruńska text and math. Singular appearance (see samples).
18. Resolves to Pagella, with added figures.
19. Can use `newpxmath` as math package.
20. Designs based originally on Palatino, but much modified to have a unique appearance.
21. Uses a variant of the mathdesign math fonts.
22. By default, uses `txfonts` for math.
23. The name is misleading as the Roman glyphs are based on URW's version of Palatino, which is an old-style, not a Didone. Uses `pxfonts` for math.
24. By default, uses CM for math.
25. Lacks Latin script small caps and OsF, but seems unsurpassed for coverage of Cyrillic and Eastern European character sets.
26. This carries the promise of becoming a remarkable font family, though currently only regular weights of upright and italic are available. It

offers both small caps and “petite caps”, the latter having an x-height equal to the x-height of the font, and a greater selection of figure style than most commercial fonts.

27. Unlike most Baskerville renditions, this has low contrast, and the italic is quite heavy. No fixed width or old style figures, nor small caps.
28. No fixed width or old style figures, nor small caps.

Notes on commercial fonts

- A. High contrast (ratio of widest stems to narrowest).
- B. No oldstyle figures or small caps.
- C. No oldstyle figures.
- D. Scale up by about 6%.
- E. Too decorative for scientific text?
- F. Slab serif, very geometric. Maybe for slides?
- G. Limited small caps. Scale up 10–15%. Fine-looking font.
- H. Slab serif. Slides?
- I. Limited small caps. Scale up 7%.
- J. Goudy Oldstyle.
- K. Small x-height—elegant, but hard on older eyes.
- L. Update of Utopia, even denser.
- M. Slab serif with character.
- N. Scale down about 8%. Includes math fonts.
- O. Upper case italic not slanted.
- P. Scale up by 15–18%.
- Q. High italic angle—beautiful but less readable.
- R. Glyphs very similar to garamondx but more widely spaced.
- S. More extensive than Utopia in Fourier.
- T. This text font is the basis for fonts used by the SMF (*Société Mathématique de France*) for its journals, the mathematical fonts deriving from Adobe Mathematical Pi and a private release by Yannis Haralambous. See [2].

Some personal opinions

Fonts without small caps in at least the upright shapes are severely lacking, as are those without a full set of common f-ligatures in each style. I think oldstyle figures make a real difference to the appearance of a document and should be available as the default text figures. These stylistic principles have a bearing on the assertions below.

The number and quality of the text fonts in the “free” category is much improved since Stephen Hartke’s survey [3] of 2006, with LinLibertine perhaps the most notable example. As is apparent from the

above list of properties of existing commercial fonts, many have languished for years without improvement and lack some combination of amenities which I now consider essential.

Of the free fonts, I am most partial to *LinLibertine*, *garamondx* and *mathpazo*/*newpx*. *LinLibertine* and *newpxtext* (which is based on a slight modification of TeX Gyre Pagella) have the quantitative edge when scored by criteria (A'-E'), but I prefer the overall appearance of *garamondx*, even though I wish it were more generously spaced, in the manner of *StempelGaramond*. *Mathpazo* and *newpx* come with built-in math support, and matching math support for *LinLibertine* and *garamondx* are available as options to *newtxmath*. *Garamondx* may also be used with the *mathdesign* package using the *garamond* option.

Among the commercial fonts, there are some first-rate contenders. *LucidaOT* has the benefit of a math font designed from ground up to accompany the text font, and all at a very reasonable price. The “Pro” font families in the list comprise some of Adobe’s most impressive offerings, some of them surely as close to technical perfection as font families can be. Those that are the most interesting to me—*BriosoPro*, *WarnockPro*, *MaiolaPro*—may lack the gravitas required of academic research papers and books, and *GaramondPremierPro* may now be so overused as to appear old hat. I find *UtopiaStd* and *KeplerStd* too plain and too cramped for comfortable reading. My favorites among the rest come down to *BemboStd*, *CentaurMTStd* and *DanteMTStd* which, despite their slight technical inadequacies, possess, so to speak, real character. There is much to be thankful for with the Adobe fonts that are not simply licensed from others. Unlike most foundries, their fonts have licenses that allow modifications.

Math fonts that are well-matched to the Adobe fonts are not so easy to find. There is now a homogeneity of design in the newer Adobe fonts that renders many of them poor contenders for mathematical use because the italic v is almost indistinguishable from Greek \nu, requiring a substantial work-around.

The *MinionPro* package on CTAN provides a math package based on *MnSymbol* that is a good match to *MinionPro*, but which has some problematic features: (a) the symbols are on the small and light side—indeed, some are borrowed from Computer Modern; (b) math italic v and Greek small letter \nu are indistinguishable; (c) the scale is not adjustable; (d) the package is so cleverly constructed as to be quite difficult to modify.

A number of Adobe text fonts may be adapted

to the *newtxmath* package, with some amount of labor and skill required. The *minion* option to *newtxmath* provides one example of what can be done—the math italic and Greek symbols are taken from *MinionPro* text but other symbols are from *newtxmath*, the end result being a little heavier than *MinionPro* math. *NewBaskervilleStd* adapts well to *newtxmath*, but has some deficiencies: it lacks a full set of f-ligatures and has small caps only in upright shapes. *Baskervald* is not a good substitute for *NewBaskervilleStd*, having much heavier italics that don’t match *newtxmath* well, among its other issues. It may be that *Baskerville* is the new black, in a manner of speaking. The recently issued (and very expensive) *Baskerville 10 Pro* has made quite a splash—for example, it is now the Metropolitan Opera’s official font, replacing *Garamond*.

References

- [1] Horn, Berthold. “Where Are the Math Fonts?” *TUGboat* 14:3 (1993), pp. 282–284.
- [2] Haralambous, Yannis. “Une police mathématique pour la Société Mathématique de France: le SMF Baskerville”. *Cahiers GUTenberg* 32—actes du congrès GUT’99, Lyon, mai 1999.
- [3] Hartke, Stephen G. “A Survey of Free Math Fonts for TeX and LATEX”. *The PracTEX Journal* 2006 No. 1, pp. 1–26.

Appendix—Some font samples

Further, more extensive, samples are available at <http://math.ucsd.edu/~msharpe/ffsamples.pdf>

\usepackage[math]{anttor}

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

\usepackage{baskervald}
\usepackage[lite]{mtpro2}% free

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{gtamacbaskerville}
\pdfmapfile{+gtamacfonts.map}
\usepackage[lite]{mtpro2}% free
```

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[scaled=.84]{librebaskerville}
\usepackage[lite]{mtpro2}% free
```

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

%Computer Modern

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{fourier}
```

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{fourier,venturis}
```

Roman text, Small Caps, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{fouriernc}
```

Roman text, SMALL CAPS, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{gfsartemisia}
```

Roman text, SMALL CAPS, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{gfsbodoni}
```

Roman text, SMALL CAPS, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{gfsdidot}
```

Roman text, SMALL CAPS, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{kmath,kerkis}
```

Roman text, SMALL CAPS, *Italics*, **Bold roman** and **Bold Italic**, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{kpfonts}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{libertine}
\usepackage[libertine]{newtxmath}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[sc]{mathpazo}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{newpxtext,newpxmath}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage{newtxtext,newtxmath}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[lining]{ebgaramond}
\usepackage[garamondx]{newtxmath}
```

Roman text, **SMALL CAPS**, *Italics*, (no bold variants), followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[garamondx]
\usepackage[garamondx]{newtxmath}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[garamond]{mathdesign}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[utopia]{mathdesign}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

```
\usepackage[charter]{mathdesign}
```

Roman text, **SMALL CAPS**, *Italics*, **Bold roman** and ***Bold Italic***, followed by some display math:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt = \frac{1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

◇ Michael Sharpe
Math Dept, UCSD
La Jolla, CA 92093-0112
USA
`msharpe (at) ucsd (dot) edu`
`http://math.ucsd.edu/~msharpe/`

Glisters

Peter Wilson

This simple bug is tied from black Glister which is a synthetic material with iridescence and peacock like colouration.

Black Glister Bug, HARTLEY FLY FISHING

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine. This installment presents fonts in some of their aspects.

Ornament is but the guilèd shore
 To a most dangerous sea; the beauteous scarf
 Veiling an Indian beauty; in a word,
 The seeming truth which cunning times put on
 To entrap the wisest.

The Merchant of Venice, WILLIAM SHAKESPEARE

1 A font of fleurons

In an earlier column [8] I showed how printers' ornaments and flowers could be combined to make interesting patterns. A while later I obtained John Ryder's book on flowers, flourishes, and fleurons [5] in which he discussed a rather fine set of fleurons that are thought to have been cut by Robert Granjon around 1565. These are known collectively as *Granjon's Arabesque* or *Granjon's Fleurons*. I found a commercial font of these, the Lanston Type Company's *LTC Fleurons Granjon*, for Mac or Windows and I purchased the Windows version which came as both TrueType and Type1 fonts. The Type1 files were *LTCFleurGranj.afm* and *LTCFleurGranj.pfb*. The question then was: How do I use these in L^AT_EX?

I read Philipp Lehman's wonderful guide to installing Type1 fonts for L^AT_EX and it seemed pretty simple [3]. First, decide on a name for the font using the Karl Berry naming scheme. But Lanston Type Company was not a 'known' supplier and other aspects of the naming convention didn't really seem to apply, so I ignored the Berry scheme and made up a name; the *zlgf* font with family name *lgf*.

Next, copy the original *afm* and *pfb* font files to our newly named font (thus preserving the original files in case of disaster, which did happen—several times). So, we now have *zlgf.afm* and *zlgf.pfb*.

I then blindly used *fontinst* with the 'default', *latinfamily* which produced various files which I then installed in their proper locations, and ran a test file meant to show all the glyphs. It didn't.

After much huffing and puffing, trying to read encrypted binary files, looking at the font in George

Williams' amazing FontForge [6], and other possibly useful things I eventually managed to install the font on, I think, the 5th attempt (I had paid money for the font and I wasn't going to give up).

FontForge revealed that the actual font name was *LTCfleuronsGranjon* and the font's family name was *LTC Fleurons Granjon*. It also turned out from using FontForge to check the font that some of the glyphs were in L^AT_EX's normal range of 0–255 while others were above that, and L^AT_EX couldn't deal with the higher-numbered ones. I read the *Font Installation Guide* several more times and with its help eventually came up with the following:

- Opened *zlgf.pfb* in FontForge and reencoded it in *Glyph Order*, which just numbers the glyphs continuously in the order they appear in the file, then used *Generate Fonts* to keep the new encoding and regenerate *zlgf.afm* to match.
- Followed Lehman's example of installing symbol fonts. That is, I created two files; the first, based on [3, p.46], I called *makelgf.tex*:

```
% makelgf.tex fontinst file
%           for Granjon's Fleurons
\input fontinst.sty
\recordtransforms{lgf-rec.tex}
\installfonts
\installfamily{U}{lgf}{}%
\installrawfont{zlgf}{zlgf}%
{txtfdmns,zlgf mtxasetx}{U}{lgf}{m}{n}{}%
\endinstallfonts
\endrecordtransforms \bye
```

And the second, based on [3, p.17], I called *maplgf.tex*:

```
% maplgf.tex fontinst file to
%           generate map for lgf font
\input finstmsc.sty
\resetstr{PSfontsuffix}{.pfb}
\adddriver{dvips}{lgf.map}
\input lgf-rec.tex
\donedrivers \bye
```

Then I ran T_EX on them, in that order. The result was two files, the first *ulgf.fd*:

```
%Filename: ulgf.fd [...]
\ProvidesFile{ulgf.fd}
[2009/10/10 Fontinst v1.929
 font definitions for U/lgf.]
\DeclareFontFamily{U}{lgf}{}%
\DeclareFontShape{U}{lgf}{m}{n}{<->zlgf}%
\endinput
```

and the second *lgf.map* (one line):

```
zlgf LTCFleuronsGranjon <zlgf.pfb
```

Then I ran the program *afm2tfm* on *zlgf.afm* to create *zlgf.tfm*.

- Move the various files to their proper places in the TDS tree. I made a `lanston` directory in each place to hold the files in case I ever wanted to install another Lanston Type Company font. The several files ended up in the `texmf-local` tree as:

```
.../fonts/map/dvips/lanston/lgf.map
.../fonts/afm/lanston/zlgf.afm
.../fonts/tfm/lanston/zlgf.tfm
.../fonts/type1/lanston/zlgf.pfb
.../tex/latex/lanston/ulgf.fd
```

and then *refresh the database*, in my case by running `texhash`.

- Ensure the new `.map` file can be found by running `updmap [-sys]`. (Make sure that you either always run `updmap` and never `updmap-sys`, or you always run `updmap-sys` and never run `updmap`. If you should ever alternate these then access to your fonts is likely to be all messed up.) In my case, as administrator/root I ran:

```
updmap-sys --enable Map=lgf.map
```

- The fonts should now be available for use. I wrote a little test file to see if all the glyphs were available by generating a font table, using the `fonttable` package [7], and a macro to print a glyph by giving its number in the font table:

```
% testlgf.tex Test the lgf font family
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage{fonttable}
% typeset a character by number
\newcommand*\{F\}[1]{%
  \usefont{U}{lgf}{m}{n}\char#1}
% zero extra line spacing
\newcommand*\{zeroxls\}{%
  \lineskip=0pt\lineskiplimit=0pt}

\begin{table*}
\centering
\caption{The Granjon Fleurons glyphs}
\label{tab:lgf}
\nohexoct
\fontsize{12}{12}
\xfonttable{U}{lgf}{m}{n}
\end{table*}

% usage examples
\begin{center}\zeroxls
\fontsize{24}{24}\F{11}\F{12} \\
\qquad\qquad\qquad \F{13}\F{14}
\end{center}
\begin{center}\zeroxls
\fontsize{24}{24}\F{14}\F{13} \\
\qquad\qquad\qquad \F{12}\F{11}
\end{center}
```

```
\begin{center}\zeroxls
\fontsize{24}{24}\F{26}\F{47}\F{75}\F{54} \\
\qquad\qquad\qquad \F{27}\F{46}\F{74}\F{55}
\end{center}
\end{document}
```

The results from the test file are in Table 1 and the three arabesques below.



Many other arabesques may be created, like those below and the ‘moustachios’ used in a previous column as anonymous divisions setting off the `TEXMAG` articles [9].



All was well with using my fleurons font until I came to install the next version of `TeX` Live, when the fleurons suddenly became unfindable. Apparently new fonts installed as I had done had to be reinstalled whenever `TeX` Live was (re)installed. Norbert Preining advised me on how to go about avoiding this problem.

The best solution, at least at the time of writing, is to add the `Map` line(s) (the contents of the file `lgf.map`, in my case) to the file (creating it if necessary) `.../web2c/updmap.cfg` in the tree where the fonts are installed — `texmf-local` in my case.

After updating `updmap.cfg`, it’s then necessary to run

```
updmap-sys
```

which completes the operation. The idea is that each `texmf` tree in use has its own `updmap.cfg`, and `updmap[-sys]` reads them all to generate the final files used by `pdftex`, `dvi`, et al.

Slow, slow, fresh fount, keep time with my salt tears.

Cynthia’s Revels, BEN JONSON

2 Fonts, GNU/Linux and X_ETEX

Having gone to the trouble to get `LATEX` to use my new fleurons font I thought that it might have been

Table 1: The Granjon Fleurons glyphs

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111

easier to use X_ET_EX as I understood that it could handle any system font without the contortions involved in setting one up for L_AT_EX. It seems that if you are on a Mac or Windows machine installing a new system font is trivial. However, I work on a Linux box and my first difficulty was in finding out how to install a new system font. All articles on the subject that I googled had different ideas on the subject, some very complicated. I eventually, with much trepidation, tried what appeared to be the simplest method which was to:

- Copy the font afm and pfb files into a directory under /usr/share/fonts, which I created and called Lanston.¹
- As root, run fc-cache -f -v so that it will cache the new font for use.
- Run fc-list, which returns a list of the system fonts, to check that the new font is now among them.

Now for the test. A simple X_ET_EX file:

```
\documentclass{article}
\usepackage{fontspec}
\fontspec[LTCfleuronsGranjon]
\begin{document}
ABCDEFGHI
\end{document}
```

which produced:

ABCDEFGHI

¹ I tend to uppercase the first letter of directory names, but not necessarily consistently.

... an abject failure! It should have typeset the corresponding fleurons.

I had come across a method for displaying a table of all the glyphs in a font by Guido Herzog in a posting to the X_ET_EX mailing list [2]. I used X_ET_EX on this for my fleurons font:

```
% glyphs.tex -- find glyphs and their index
\parindent 0pt
%% the font to test
\font\test=LTC Fleurons Granjon at 14pt
% this next one also works
%\font\test=LTCfleuronsGranjon at 14pt

\newcount\charcountA \charcountA 0
\newcount\charcountB
\charcountB \XeTeXcountglyphs\test
\advance\charcountB -1\relax
\newcount\charcountC \charcountC 0

\def\ystrut{%
\vrule height 15pt depth 5.5pt width 0pt}
\advance\vsize 4\baselineskip

\loop
\advance\charcountC 1\relax
\leavevmode
\hbox{\hbox to 10mm{%
\hss\#\charcountA\quad\#}%
\hbox to 10mm{%
\test\XeTeXglyph\charcountA\ystrut\hss}}%
\ifnum\charcountC = 8
\endgraf \charcountC 0\fi
\ifnum\charcountA < \charcountB
```

```
\advance\charcountA 1\relax
\repeat
\bye
```

The result was a table similar to Table 1 displaying all the fleuron glyphs. This meant that X_ET_EX found my new font but for some reason my use of the `fontspec` package [4] might have been at fault. I eventually discovered that I should have called `\fontspec` in the body of the document:

```
\documentclass{article}
\usepackage{fontspec}
\begin{document}
\fontspec[LTCFleuronsGranjon]
ABCDEFH
\end{document}
```

which made the fleurons the current font, or alternatively use `\setmainfont` in the preamble:

```
\documentclass{article}
\usepackage{fontspec}
\setmainfont[LTCFleuronsGranjon]
\begin{document}
ABCDEFH
\end{document}
```

to make the fleurons the main (default) font.

Now, it seems simple enough to typeset with new fonts on a Linux box.

We started off trying to set up a small anarchist community, but people wouldn't obey the rules.

Getting On, ALAN BENNETT

3 Mixing traditional and system fonts

A little while ago I was extending an older document where I had been using several fonts set up for the traditional L_AT_EX methods—Type 1 fonts with `tfm` and `map` files. For swapping from one font to another I used a macro

```
\newcommand*{\FSfont}[1]{%
\fontfamily{\#1}\selectfont}
```

where the argument is the font's family name. This worked well.

I then wanted to use a new font, `IM_FELL_Double_Pica_PRO_Roman`, which didn't come with L_AT_EX support files. So I added it to the system fonts directory, added it to the document with my `\FSfont` macro, and used `xelatex`, together with `fontspec`, instead of `pdflatex` for processing. The new font displayed well but all the others reverted to the default Latin Modern fonts.

I eventually had to ask on `ctt` and Ulrike Fischer responded [1] that with `xetex/fontspec` the default encoding is set to EU1 but with `pdflatex` it is set to T1. Therefore I had to take account of encodings when moving from `pdflatex` to `xelatex`.

In my case I was only using the normal alphanumeric and punctuation characters which are in the same slots in the EU1 and T1 encodings. Thus, changing my `\FSfont` macro to:

```
\newcommand*{\FSfont}[1]{%
\fontencoding{T1}\fontfamily{\#1}\selectfont}
```

fixed the problem for me.

References

- [1] Ulrike Fischer. Re: XeLaTeX, `fontspec` and `fontfamily`. Post to `comp.text.tex` newsgroup, 12 July 2010.
- [2] Guido Herzog. Re: [XeTeX] Trouble with displaying word containing 3 conjunct consonants in Devanagari. Post to `xetex` mailing list, 24 September 2009.
- [3] Philipp Lehman. The font installation guide, December 2004. (Available on CTAN at </info/Type1fonts/fontinstallationguide>).
- [4] Will Robertson and Khaled Hosny. The `fontspec` package, 2010. Available on CTAN in `macros/latex/contrib/fontspec`.
- [5] John Ryder. *Flowers & Flourishes including a newly annotated edition of A Suite of Fleurons*. The Bodley Head for Mackays, 1976. ISBN 0370 11308 X.
- [6] George Williams. FontForge: An outline font editor, 2009. Available at <http://fontforge.sourceforge.net/>.
- [7] Peter Wilson. The `fonttable` package, 2009. Available on CTAN in `macros/latex/contrib/fonttable`.
- [8] Peter Wilson. Glisterings: Ornaments. *TUGboat*, 32(2):202–205, 2011.
- [9] Peter Wilson. Glisterings: Timelines, parsing a filename. *TUGboat*, 33(1):39–42, 2012.

◊ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
`herries dot press (at)`
`earthlink dot net`

Interview with Charles Bigelow

Yue Wang

Abstract

Interview of Charles Bigelow by Yue Wang, conducted in 2012.

Y: In this interview we are very lucky to have Charles Bigelow with us. Professor Bigelow is a type historian, educator, and designer. With his design partner, Kris Holmes, he created the Lucida family of fonts used in the human-computer interfaces of Apple Macintosh OSX, Microsoft Windows, Bell Labs Plan 9, the Java Developer Kit, and other systems, bringing historical and technical understanding of type to hundreds of millions of computer users. In 2012, Bigelow retired from the Melbert B. Cary Distinguished Professorship at Rochester Institute of Technology's School of Print Media. He is now the RIT Scholar in Residence at the Cary Collection, RIT's rare book Library.

C: Thank you for your visit.

1 Entering the digital type era — the birth of Lucida

Y: Let's get started. Can you briefly introduce the design goal of Lucida?

C: In the early 1980s, we saw that computers would become more widely used and that digital typography would be possible for more people. At that time, digital printers and computer screens had low resolutions. The goal of Lucida was to create a new, original family of fonts for medium and low-resolution digital printers and displays.

Y: Is this the reason why that's called Lucida?

C: Exactly. We wanted to give it a name that could suggest it was made of light and was clear despite the low resolutions. "Lucida" comes from the Latin word "lux" for light and clarity. It turned out that Lucida was the first original typeface designed for both digital printers and computer screens.

Y: Wow, really?

C: Yes. There had been previous digital typefaces designed for high-resolution typesetting machines in the late 1970s and early 1980s; a few were original types like Hermann Zapf's Edison, but none were new and original for laser printing and display screens (mainly CRTs in that era). Adobe developed their own font format called "PostScript Type 1" and digitized 35 typefaces for Apple's LaserWriter Plus

This interview was made possible with support from *Programmer Magazine* in China, <http://www.programmer.com.cn>.

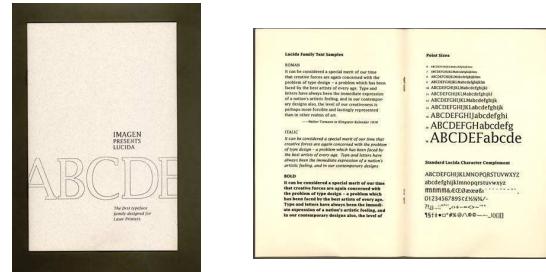


Figure 1: Lucida was the first new, original family of types designed for digital laser printers and screens. This is the first Lucida specimen, printed on a 300 dot per inch digital printer by the Imagen Corporation in California. Distributed at the ATypI conference in London, September 1984.

printer. These fonts, including Helvetica, Times Roman, Palatino, etc., had originally been designed as metal type, and some like Zapf Chancery for phototypesetting. Designed before the digital era, those faces were not created for low-resolution digital rendering. When the first commercial font of Lucida was shown in 1984, it surprised Adobe. They knew of it; they had even digitized a test version, but they hadn't thought anyone would take the risk of making new designs for the new technology of laser printing. Instead, a Silicon Valley digital printer firm, Imagen, founded by Stanford researchers and graduates, some of whom had worked with or been students of Donald Knuth, brought out Lucida first. Imagen's type director, Mike Sheridan, wanted to produce a new design for the new technology and chose Lucida. Now, 30 years later, it appears he was right, but at the time, he took a risk. Adobe licensed Lucida fonts some years later and still distributes them.

C: Here (fig. 1) is the first Lucida (seriffed) specimen, printed on a 300 dot per inch digital printer by the Imagen Corporation in California. It was distributed at the ATypI conference in London, September 1984.

Y: Cool. The specimen only included Lucida (seriffed).

C: Yes. The serifed family was first shown in 1984, and the sans-serif family was released in 1985.

Y: What makes Lucida look great even in low resolutions?

C: We first did experiments, making bitmap letters by hand and comparing them to what we thought would be the outlines that could produce them. We found out several factors (see fig. 2). First, a big x-height packs more pixels into the most visually important portions of text, the x-height parts of letters. A big x-height is an advantage for texts read mostly on screens. That's one reason Apple has been using Lucida Grande as the standard user interface typeface on Mac OS X.



Figure 2: Early studies for Lucida, comparing brush and pen written letters to bitmap redesigns for low-resolution printers and computer screen displays.

Y: That's why Lucida's x-height is bigger than most fonts, such as Times Roman or Baskerville, when composed at the same point size.

C: Exactly. There are still questions today about the importance of x-height for legibility in Latin alphabetic fonts. A vision scientist, Gordon Legge, and I recently wrote an article on the importance of type size for legibility, and we argued that x-height is the main factor that affects perception of type size [6]. The measure of x-height applies only to typography with upper and lower-cases: Latin, Greek, Cyrillic, and Armenian. For case-less writing systems, various other factors affect the impression of size.

Secondly, Kris Holmes and I observed that technical publications make frequent use of words in all capitals, such as acronyms, emphasized expressions, keywords, and the like. Therefore, we made the Lucida capital height a little shorter than the ascender height (e.g. the height of a lower-case 'h' or 't'), to reduce the distracting look of words set in all capitals. This was not a new idea in typography; in 1495, the famous Venetian printer Aldus Manutius introduced a roman type with slightly shortened capitals cut by Francesco Griffó.

Third, the weight of Lucida is darker than traditional book typefaces. We noticed that on screens with black text on white backgrounds, the letters were slightly eroded, seeming too light, so we darkened the Lucida stem weights a little bit. The stem weight is 1/5.5 of the x-height, and a little bit less than 1/10 of the body size. Its overall gray tone is roughly 22% when the text is set solid (no extra line spacing).

Fourth, at low resolutions, a single pixel is often the only space between letters rendered at text sizes (8 point to 16 point). If letterspacing is tight, which was fashionable in advertising typography in the

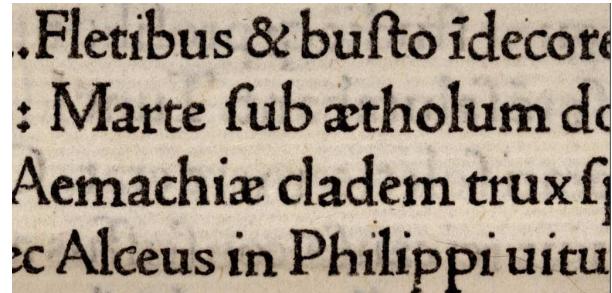


Figure 3: Scanned image from a book printed by Nicholas Jenson in Venice, 1478. These early typographic letters are rather dark and widely spaced.

**Fletibus & busto idecc
: Marte sub aetholum
Aemachiae cladem tru
ec Alceus in Philippi uitu**

Figure 4: The same text from Jenson composed in the original Lucida font. Lucida is also somewhat dark and widely spaced, for early digital printing technology and display, but it is not a copy of Jenson.

1970s and 1980s, it can cause letters to touch. Some designers called this "sexy spacing", but it turns out to impair legibility. There are still debates about whether legibility is based on recognizing whole words or individual letters. Lucida is on the side of letter recognition. Computer screens were read from greater distances than print, which visually reduced letterspacing and caused crowding of the shapes, so we gave Lucida slightly loose spacing to counteract these tendencies.

Also, we created letter forms with large open counters — the internal open spaces like in 'a' and 'e' — to keep the interiors from collapsing and reducing legibility. Another small detail, which almost nobody notices, is that we lowered the joins of the arches in letters like 'n', 'm', 'h', 'r', and 'u', to give them more definition.

Y: So it won't clog up :).

C: Yes. For instance, we didn't want the top of an 'n' to clog up and look like a smeared 'o'. In fact, most of the ideas behind Lucida were not new. Some we borrowed from very early typography. Here's a scanned image from a book printed by Nicholas Jenson in Venice, 1478, when printing technology didn't have as high a quality as in later eras (fig. 3). These early typographic letters are rather dark and widely spaced, too. The forms are somewhat distorted by the technology of early printing. Rough paper, soft metal type that wears quickly, uneven pressure and

ink squash, and so on. We borrowed some of Jenson's design ideas and believe we were the first to try them in the low-resolution digital era. Here (fig. 4) is the same text composed in the original Lucida font. It is not a copy of Jenson but shares similar goals — to make legible letters for a noisy medium. Jenson's type was around 15 point, but Lucida is more often used at smaller sizes — 10 to 14 point on screens — so we made its spacing even a little looser.

Y: Amazing!

C: Lucida, by showing some successful solutions to resolution-restricted text, also encouraged other designers to innovate. An interesting question is whether designers should try to compensate for limitations of new technology or design ideal shapes. Lucida is a design to compensate for limitations of resolution and imaging, but, in contrast, exuberant digital cursives like Zapfino or Apple Chancery are designs that take advantage of technological advances in character substitution.

Ten and twenty years after the first Lucida fonts, other designers created their own solutions to the problem of creating new faces for low resolutions. This is Lucida in 1985–87. In 1996, Microsoft released Verdana and Georgia by Matthew Carter, for the Windows operating system and Internet Explorer. Ten years after that, Microsoft released the ClearType font collection, including Calibri, Cambria, Candara, Consolas, Constantia, Corbel, and Cariadings (see fig. 5) by several designers, among them my former student, Gary Munch (Candara). These fonts take advantage of advanced screen display technology by Microsoft.

Y: They look similar to Lucida. Corbel and Lucida Sans are almost the same.

C: Well, but they are not copies of Lucida. These later designs show similar adaptations to the problem of design for screens: large x-heights, loose letter spacing, open counter-forms, and simplified letter shapes. In the alphabet samples at the bottom, the types are scaled to equal x-heights, to show similarities more clearly. Our emphasis on open counters and Renaissance forms for Lucida came from the calligraphic instruction Kris Holmes and I had with Lloyd Reynolds, calligrapher laureate of Oregon. Our idea of applying Renaissance forms to sans-serif came from Hans Meier's Syntax design, a sans-serif book typeface based on Renaissance humanist types and handwriting. It came out as metal type in 1968 and influenced not only us, but later generations of designers. The ideas of Syntax are now common in so-called "humanist" sans-serifs, but Syntax remains a splendid design, a great improvement on its successors.

A: Body sizes the same, x-heights vary

1984 Lucida, 1985 Lucida Sans, 1987 Lucida Bright

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

1996, Verdana, Georgia

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

2006 Calibri, Candara, Corbel, Cambria, Constantia

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

A quick brown fox jumps over the lazy dog.

B: x-heights equal, body sizes vary

Lucida Sans [12 point]

A quick brown fox jumps over the lazy dog.

Verdana [11.7 point]

A quick brown fox jumps over the lazy dog.

Corbel [13.7 point]

A quick brown fox jumps over the lazy dog.

Lucida Bright [12 point]

A quick brown fox jumps over the lazy dog.

Georgia [13.2 point]

A quick brown fox jumps over the lazy dog.

Cambria [13.6 point]

A quick brown fox jumps over the lazy dog.

Figure 5: Examples of original typefaces designed for digital printing and screens, showing convergent evolution. Lucida was the first original family for screens and laser printers. Later designs show similar adaptations to the problem of design for screens: large x-heights, loose spacing, open counter-forms, simplified letter shapes. In the alphabet samples at bottom, the types are scaled to equal x-heights to show relationships more clearly.

Y: I see. Why did Lucida Sans come out later than Lucida (serif).

C: We released the serifed family first, as a kind of proof-of-concept, and then worked on the sans-serif. After the serifed design came out, there was interest in the sans-serif version, which we released within the next year. Historically, because of the effort involved, large typeface families were not often released all at once. For a serifed family, maybe only roman and italic would be released, and later one or more bold weights. If a typeface became popular, then a firm might release more variants (like more bold weights) or companion typefaces (like sans-serif, or fancy characters). In the mid-1930s, Jan van Krimpen in Holland was the first designer to create a family that included serif and sans-serif, and also

chancery cursive variations, in his Romulus typeface family. The sans serif set was never commercially released, alas. Following Van Krimpen, we believed that a more harmonious pattern of text could be achieved if the different styles of type were designed together as an integrated set. This principle has held up well over the years. So, we created an extended family of serif, sans-serif, monospaced (typewriter), and various scripts (calligraphy, handwriting, casual), incrementally, over a period of several years.

Y: Because people use more variants in a single document? Knuth's Computer Modern is a complete font family too.

C: Yes. Donald Knuth, approaching typography with a mathematical intellect, also recognized the same principle that Van Krimpen first saw, that a typeface family could be implemented as a group of parametric variations of a basic form. Although Knuth says his goal was to imitate a metal typeface called Monotype Modern 8A, Computer Modern has many original ideas underlying its forms. In visual form, the basic serifed version of Computer Modern did imitate Monotype Modern, but in conception and technical implementation, Computer Modern was original. It is noteworthy and commendable, too, that Knuth published all the Metafont code for his designs. For commercial reasons, most typefaces are marketed with intellectual property restrictions, but Knuth saw his typographic work as part of a greater goal, the publication of scientific literature and the dissemination of knowledge. He did the same with his TeX system for mathematical composition, publishing the source code for wide usage. A paragon of enlightened generosity.

An interesting aspect of Knuth's work on Computer Modern and the way he uses it in his TeX composition system, is that he established additional semantic categories for technical typography. Technical documents usually use different font variants to indicate different semantic meanings of the text and formulae. For example, in TeX, there are three slant variants — a slanted roman variant to indicate book names, a cursive italic to indicate emphasis, and a math italic for math equations. Prior to that, roman typefaces had either true cursive italic or a slanted roman (sometimes called italic) as their companion design, but not both. Times Roman, for example, has a cursive italic, but Helvetica has a slanted roman for italic. I am ignoring the slight visual adjustments that designers make to ostensibly slanted forms. In his work, Knuth began to use three slant variants, true italic, slanted roman, and math italic, and that led to us making the three italic variants for Lucida Bright math fonts: the normal

text italic, which is semi-cursive; a slanted roman; and a cursive italic for math variables. To these, we added a chancery cursive in the 16th century Italian style, which we called Lucida Calligraphy. It can be used for math, but is more often used for display and ornamental typography.

This idea of a family of typographic variations is not new. It evolved over hundreds of years. In digital typography today, it is easier to produce typefaces than in the hand-cut metal era, so we can make bigger families within a few years instead of centuries.

One of the most fascinating trends in the history of typography is the development of new type forms and their incorporation into standard typography. Historically, roman capitals were used for inscriptions and formal handwriting during the Roman Empire (approximately 100 A.D.). Handwriting changed over the centuries and transformed the capitals into other styles that we now see as separate forms. Around 800 A.D., scribes working in the court of the emperor Charlemagne developed a "minuscule" handwriting ("small" handwriting) that had ascenders and descenders like today's lower-case. This Carolingian minuscule handwriting had no capitals. It was "mono-case" in today's terms. Around 1400, an Italian humanist scribe, Poggio Bracciolini, revived and combined the ancient roman capitals with the Carolingian minuscule to make a new kind of formal handwriting that he and other humanists thought was more legible than the gothic scripts then in wide use. (I should explain that these humanists were Italian Renaissance scholars and writers who shifted their studies from religion and theology, which had been medieval concerns, to philosophy, literature, classical languages (Greek and Latin), history, and other subjects we now call the "humanities".) Poggio made an amalgamation of what we now call upper- and lower-case in typography. A scholar friend of Poggio, named Niccolò Niccoli, developed a fast version of Poggio's handwriting. This was before printing; Niccoli copied many books, so he wanted a faster style of handwriting that was still legible. The Italians called Niccoli's style "running" hand(writing), "corsiva" in Italian. Today we use the term "cursive" in English to mean the same thing, a faster, freer script. The cursive tendency appears in other writing traditions as well. In Chinese writing, for example, there are both formal and cursive styles. You will know the Chinese names better than I do. On the formal side, there is Official style or clerical script (*li shu*) and a somewhat more cursive Regular style (*kai shu*). On the informal side, there is the semi-cursive style or running script (*xing shu*), and the very cursive style (*cao shu*).

To make a very rough comparison, Niccoli's cursive handwriting might be the equivalent of "xing shu". Some of Hermann Zapf's writing, like Zapfino, might be closer to "cao shu".

The first humanist roman types were cut around 1467, and the ancestor of most modern romans was cut in 1470 by Jenson in Venice. The first humanist cursive (italic) was not cut until 1501, and interestingly, it was cut in lower-case only. Its capitals were upright roman capitals. This shows that in those days, 500 years ago, capitals were not as tightly bound to lower-case as today. Also, cursiveness was not defined by slant alone, but by an ensemble of features, of which slant was only one. At first, italic type was an alternative to roman and whole books were composed in italic only. Italian calligraphers and type designers created many variations of italic, and later, in France, Robert Granjon cut many variations of cursive types. Around 1570, italic became a subordinate, complementary companion to roman instead of a stand-alone alternative to roman. Today, italic is an important component of a typeface family, but of secondary rank. In the 1700s, the French type designer Pierre-Simon Fournier expanded the concept of a typeface family to include variations with different widths and x-heights. In the 1800s, English typographers developed bold typefaces, which at first were separate from standard roman and italic, but by the early 20th century, especially in the designs of Morris Fuller Benton, some typeface families included bold weights as integrated members of the family. Thus, we see a pattern of incorporation of variations within a family. Sans-serif types were invented in the early 19th century but didn't become widely used for text until the 20th century. Looking at the pattern of type family evolution at the end of the 20th century, it seemed to us that incorporation of sans-serif into type families was a trend we should follow, and indeed, today in the 21st century, there are now several type families that include serifed and sans-serif variations.

Adrian Frutiger is one of the most prominent figures in this movement to extend the visual range of type families. It is difficult to describe weight and style variations in words. We have to use cumbersome names like light, extra light, semibold, extra bold, ultra bold, light condensed, and so on, and the words are different in each language, so there are international communication confusions. For his large Univers family designed in the 1950s, Frutiger developed a two digit system to differentiate the weights, widths, and slants of the variations. The base of the system was 55, a normal weight roman, upright font. The first digit of the classification ex-



Figure 6: From left: Lucida Bright, Lucida Casual, Lucida Handwriting; comparison of letter 'a'. All three designs have the same x-height.



Figure 7: Comparison of original Lucida (seriffed) and Lucida Sans, showing close similarities of forms.

pressed the thickness of the weights, for example, 4 is light, 5 is regular, 6 is semi-bold, and 7 is bold. The second digit describes the style, for example, 6 is italic, 7 is condensed. So, 56 means normal weight, slanted, whereas 65 means roman, semi-bold, and so on.

Y: It looks like a periodic table!

C: It sure does. So after Univers, designers were able to use many variants in a single document. Emil Ruder, a famous Swiss teacher of typography, demonstrated this in his book "Typography" [9]. Ruder's students continued this design approach. Thus nowadays we need large families of fonts for the most expressive kinds of modern typography.

Y: What are the common features among the big variations within the Lucida family?

C: In technical perspective, all the Lucida fonts have the same x-height, capital height, and similar series of stem weights, which helps give a harmonious look to a page that uses different font styles. Here (fig. 6) is a comparison of the letter 'a' in Lucida Bright, Lucida Casual, Lucida Handwriting. All three designs have the same x-height.

Y: I see.

C: But there are a lot of similarities among different font styles as well. For example, this (fig. 7) is the original Lucida Serif and Lucida Sans. The design is really harmonized. This (fig. 8) is an early specimen of the first four Lucida serifed and sans-serif typefaces around 1986. But we didn't stop. We created an extended font family that included serifed, sans-serif, and fixed-pitch (typewriter) designs. Around 2000, we had almost all the main variations of the Lucida typeface family of today. Here (fig. 9) is a list of them in normal form. As you can see (fig. 10), the typeface family is still highly unified and harmonized. The Lucida Bright family was developed

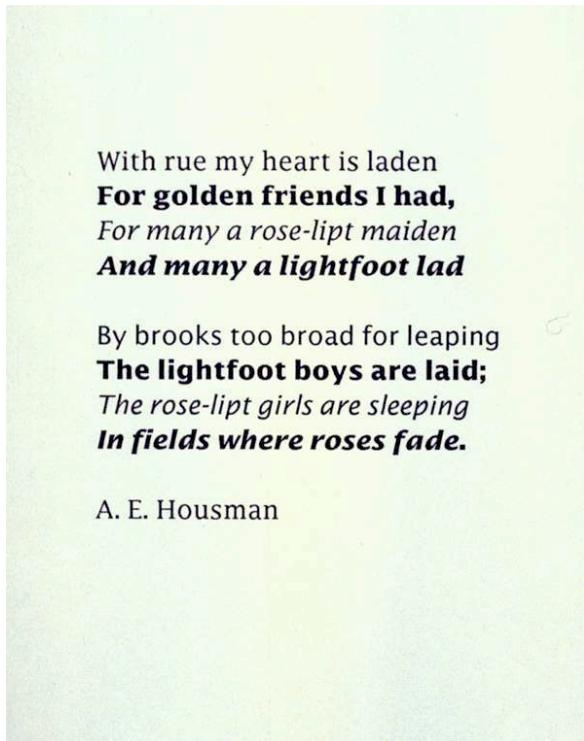


Figure 8: An early specimen of the first four Lucida serifed and sans-serif typefaces, circa 1986.

Lucida Bright & Italic Lucida Sans & Italic Lucida Casual & Italic Lucida Fax & Italic Lucida Calligraphy Lucida Handwriting Lucida Blackletter Lucida Typewriter Lucida Sans Typewriter

Figure 9: The main variations (in normal weight) of the Lucida typeface family, circa 2000.

for higher resolution systems, and was first used as the text face for Scientific American magazine in October 1987 (fig. 11).

Y: But for scientific journals there are also a lot of math equations.

C: Exactly. After Kris and I went to California, where I taught digital typography at Stanford in association with Donald Knuth, we wanted to make Lucida work well with TeX. Lucida's mathematical characters benefitted from the close association with Knuth. In



Figure 10: Study of differences in forms of different designs in the Lucida family.

SCIENTIFIC AMERICAN	
October 1987 Volume 257 Number 4	
56	The Next Computer Revolution <i>Abraham Peled</i>
In the first computer revolution the technology emerged from the laboratory as an indispensable amplifier of human intellect. A new revolution is under way: the power of computers will increase by an order of magnitude while they become as accessible and convenient as the telephone.	
66	Advanced Computer Architectures <i>Geoffrey C. Fox and Paul C. Messina</i>
Supercomputers can perform millions of operations per second. To perform billions of operations per second, a new computer architecture called parallel processing must be realized; such machines solve the elements of a problem simultaneously rather than in sequence.	
78	Chips for Advanced Computing <i>James D. Meindl</i>
The chip is the building block of computer architectures. The goal is speed, the strategy: miniaturization. As elements grow smaller their density increases geometrically and operating speed goes up arithmetically; by the year 2000 there may be a billion elements on a single chip.	
90	Programming for Advanced Computing <i>David Gelernter</i>
Parallel computers need parallel programs. Some linear programs can be "parallelized" automatically. In another approach, independent processes exchange messages as tasks are completed; in a third, elements called tuples ply tuple space, processing and becoming information.	
116	Data-Storage Technologies for Advanced Computing <i>Mark H. Kryder</i>
Advanced computers generate vast amounts of data, and yet storage devices can keep up. Magnetic devices should attain densities of four million characters per square centimeter in five years; by then magnetic optical devices may hold 10 million characters per square centimeter.	
126	Interfaces for Advanced Computing <i>James D. Foley</i>
Beyond the mouse and touch screen are new ways to make a computer natural to use. They include voice communication, an electronic pad that responds to handwriting, sensors that track eye movement and a glove that enables the wearer to manipulate objects on the screen.	

Figure 11: Table of Contents of Scientific American, using Lucida Bright. Magazine redesigned by Bigelow & Holmes using the Lucida family.

fact, we continue to learn from Knuth's examples. This (fig. 12) is sample mathematical formulae with Lucida math fonts in 1992.

Y: Oh, this is an equation sample in Knuth's *The TeXbook*. It looks better in Lucida Math!

C: Thank you, but many people still prefer Computer Modern. Our Lucida math designs give users more choice, because the families look very different in text. With Berthold and Blenda Horn of Y&Y, we augmented the Lucida math character set with many more of the math operators and arrows in the Uni-

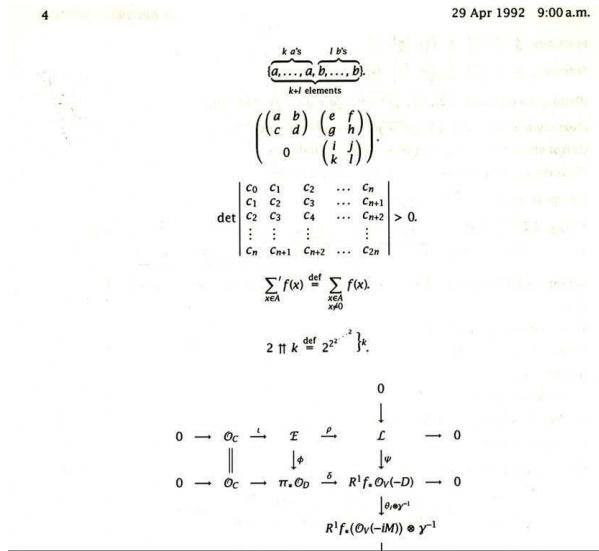


Figure 12: Sample mathematical formulae with original Lucida math fonts, 1992.

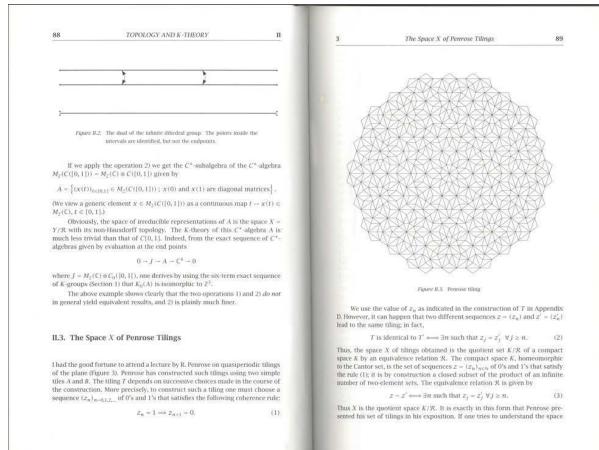


Figure 13: Lucida Bright and Lucida New Math in the design of the book *Non-commutative Geometry* by Alain Connes.

code character standard. Y&Y also developed many careful adjustments to ensure that the Lucida math fonts worked well with TeX. It was not an easy task. Here (fig. 13) is a book called *Non-commutative Geometry* by Alain Connes, which uses Lucida Bright and Lucida New Math. The book was designed by Peter Renz. Recently (2012) we upgraded the Lucida math fonts in cooperation with TUG, the TeX Users Group. We expanded the character set to include the latest Unicode math character blocks, including a new math script face by Kris, and the fonts were produced in OpenType format with the indispensable help of Khaled Hosny and others in TUG.

Y: So even without TeX, we can access these symbols using Unicode values?

C: Right. Because of Unicode encoding, computer

fonts can finally contain a wide range of characters, letters, digits, glyphs, symbols, ideograms, logos, and many others. You can include glyphs from various languages into the font. So we designed a lot of glyphs from various languages for Lucida Sans. This gave birth to Lucida Sans Unicode. We made Lucida Sans Unicode for Microsoft to show the possibility of what a Unicode font can do. Kris Holmes and I wrote a paper about this in 1993, which can be found on the web, "The Design of a Unicode Font" [2].

Y: Why make Lucida Sans Unicode? I remember Lucida Bright came out before Lucida Sans.

C: Lucida Sans was chosen to do this because of its popularity. For some typeface families that include both serif and sans-serif faces, one or the other is more popular. For Frutiger, the original sans-serif family is more popular than the Frutiger Serif, which is Frutiger's classic Meridien serifed design re-worked and given additional weights and condensed italics by Frutiger and Akira Kobayashi and released in 2008. In contrast, with Palatino, the original serifed design remains more popular than the very new and interesting Palatino Sans, by Zapf with Kobayashi, released in 2007. For text faces, it takes time for new designs to become widely accepted, so the balance of popularity between serif and sans could change in those families or ours. Every new, original type design is a risk because you don't know how well it will be accepted, and if you care only for acceptance, you don't give your design the fresh but risky insight that can make it popular. I like to quote the eminent physicist Niels Bohr: "Prediction is difficult, especially about the future."

Y: Also true for Erik Spiekermann's ITC Officina Sans and ITC Officina Serif which both came out in 1990.

C: Yes. A preference for sans-serif may be because sans-serif fonts are somewhat better for display on screens, probably because the sans-serif fonts are simpler in design, with fewer details, and therefore render slightly better at low to moderate resolutions. A vision study by Robert Morris, Kathy Aquilante, Dean Yager, and me [7] found nearly no difference between the legibility of serifed and sans-serif typefaces when all the parameters (x-height, weight, spacing, etc.) are controlled — except that at small sizes on screens, sans-serif is slightly more legible. However, we did that study ten years ago, and although we controlled for resolution, today's new, higher resolution and higher contrast screen displays could perhaps alter our findings. I believe that serif types benefit from higher resolutions.

To cover more of the Unicode range for Lucida Sans, we designed characters for Unicode Extended



Figure 14: Lucida Sans (= Lucida Grande) non-Latin designs.

Figure 15: Herodotus text on-line (Perseus Digital Library): Greek text in Lucida Grande.

Latin, Greek, Cyrillic, Hebrew, Arabic, Thai, and other languages. After we did this for Lucida Sans roman, we designed extended Unicode sets for Lucida Sans demibold, and Lucida Sans Typewriter. This gave birth to the Lucida fonts used in the Java 2 developer kit in 1999 (see fig. 14). Starting in 2001, Apple's Mac OS X includes Lucida Grande, which is a further extension of Lucida Sans Unicode, as the main operating system font. For example, the on-line version of Herodotus (the first written "history" book in western civilization), released by Perseus Digital Library, uses Lucida Grande to display Greek text (fig. 15).

2 Research on digital rendering technologies

Y: It's amazing to see you follow so closely with the advancement of font technologies. Can you tell us more about how Lucida followed the development of font and rendering technologies? You said Lucida design was highly optimized for the screen. As digital fonts evolved from generation to generation, I guess Lucida changed too.



Figure 16: Pen written italic calligraphy by Kris Holmes compared to design of Lucida Bright Italic.



Figure 17: Comparison of bitmaps at different digital resolutions of original Lucida ‘a’.

C: As I mentioned before, we did many experiments at the beginning. Here are the early studies of Lucida (see fig. 2). We realized that we should change some part of the shapes of the calligraphy to make it legible on a computer screen. For example, this (fig. 16) is the pen written italic calligraphy by Kris, but the glyph in the final digital font is different.

Y: At that time, most computer systems still used bitmap fonts.

C: Yes. On the Mac and Windows, screen fonts were originally stored in hand-tuned bitmap font files that specified individual pixel locations for a font at a particular size. We released bitmap Lucida fonts in various point sizes. Most times, for a given glyph outline, we marked every pixel inside the outline as black, and white for pixels outside the boundary. But this leaves ambivalences along the borders, so sometime manual fixes were needed to make characters more legible. Take the previous ‘a’ drawn by Kris as an example: here (fig. 17) is a comparison of bitmaps for different digital resolutions.

Y: I see. Then outline fonts were widely adopted, replacing the bitmaps.

C: Adobe was the pioneering digital publishing company at that time. They invented the PostScript language as their document format together with the PostScript outline font formats. Soon the PostScript language was widely adopted and PostScript's dominance seemed assured, and computer companies moved to adopt outline font technologies.

Y: So Adobe wanted to control PostScript to earn more money.

C: Yes, because they are a business. Adobe was in complete control of the PostScript technology at this point, and published an open PostScript language font format, called Type 3, but it didn't rasterize as well or as fast as Adobe's proprietary format, Type 1. A company had to license Adobe's PostScript to get Type 1 font technology, but major system software vendors like Apple and Microsoft didn't want a key font technology controlled by another company and didn't want to pay royalties for its use.

Y: So Apple developed their own scalable font technology.

C: Exactly. The code name was Royal, and later became called TrueType in 1991. The major technical differences between a PostScript font and a TrueType font, however, is that TrueType uses quadratic B-splines to represent the outlines, whereas cubic Bézier curves are used by PostScript. (See Robert Bringhurst, *The Elements of Typographic Style*, with a nice illustration on p. 183 in the third edition.)

Y: TrueType was a new technology. Why did it use a simpler representation (quadratic versus cubic)?

C: Some background. Several outline font formats were known then. Polygonal outlines—in which curves were approximated by a series of straight lines—were easiest to rasterize and been used for some successful digital typesetting machines, but needed too many points and were aesthetically inferior at larger sizes and higher resolutions, where the polygonal approximations of curves could be detected. Outlines composed of vectors and circular arcs needed fewer points and were fairly fast to process, but the radii of shallow arcs would be very long in comparison to the very short radii of small arcs. This problem was called numerical instability. Also, at high resolutions, there were noticeable discontinuities at tangents where a circular arc joined a straight line and curvature fell to zero.

When the outline description went beyond circular arcs and vectors, computer scientists tended to choose representations more on mathematical aesthetics than visual aesthetics. Peter Karow's Ikarus, the first commercially successful digital outline font development system, used cubic splines in Hermite

form as a master format but for practical graphical output converted the Hermite cubics to circular arcs. Knuth preferred cubic splines and based Metafont on the mathematics of parametric cubics by Sergei Bernstein—also spelled Bernshtein—a Russian mathematician. Adobe chose cubic Bézier splines, developed for computer graphics by Pierre Bézier, also based on Bernstein's work.

Apple chose quadratic B-spline outlines in part because they already used them in MacDraw, a drawing program for Macintosh, so Apple had a proprietary outline technology in-house. Apple planned to use TrueType technology for the Macintosh user interface, so they wanted fast processing and believed that quadratic B-splines could be rendered faster than cubic splines.

A very interesting outline technology was developed by Vaughan Pratt, a computer scientist at Stanford, and used by Sun in a font format called F3. It was based on generalized conic curves [8]. Pratt's inspiration went all the way back to an ancient treatise on conics by a Greek mathematician, Apollonius of Perga.

I personally liked Pratt's approach best because it was a nice compromise between computational elegance, processing speed, and intuitive geometric understanding by designers. Sun did not push to establish their conic technology as a standard, so it was eventually overwhelmed by TrueType and Type 1. I tried to persuade some Sun executives to make it an open format and the standard in Solaris and Unix, but they apparently preferred to let it die than to let it out. Later, the Java language was saved from nearly the same fate.

With such a wealth of varied mathematical representations of fonts, it was difficult to tell which, if any, were artistically superior. Visually, the quadratic and cubic forms seemed more or less equally good at representing known type forms, so different firms chose font outline representations for engineering or commercial reasons or for non-visual mathematical aesthetics.

Y: So you created the TrueType version of Lucida using quadratic splines.

C: Yes. At this stage, Apple asked us to help them explore how to make things as simple as possible. We conducted a lot of experiments using Lucida. We went with Apple's font manager and chief font engineer to URW in Hamburg, Germany, where Peter Karow at URW had invented Ikarus in the 1970s. URW had over time developed a big library of digital outline fonts. To make TrueType successful, Apple needed a good supply of high-quality digital font data, and URW had the best and the most. They also

had the technical ability to write accurate conversion programs from their format to TrueType. Most of the early TrueType fonts were produced from Ikarus format data, including the Lucida fonts, because we used the Ikarus system to digitize our designs. One of the most important experiments was, how to use as few points as possible to represent a font outline. If we had fewer points, font file sizes were smaller and, importantly, computers could render fonts faster.

Y: This is also true today. Today most of the graphics and animations are offloaded and processed using special hardware in the computer. So ironically text rendering is even slower than graphics display.
C: That's interesting. Text has some advantages over general graphics, at least for alphabetic fonts, because there are relatively few characters, so once they are rasterized for a given size and resolution, the rasters can be cached and fetched very quickly, so the pages are essentially tilings of a small number of stored and repeated graphical elements. For Chinese fonts, however, the characters are more complex and many more of them are needed, so processing was still a problem until recently. In those early days, in addition to limited processing capability, we also had other problems. Computers had limited memory, and most people were still using floppy disk. Though quadratic splines use fewer parameters than cubic splines, we needed more points to represent the shape well. To save memory, it was important to use as few points as possible, but you cannot use too few of them or the glyph outlines will be distorted. Kris and I did a lot of experiments to show Apple how many points to use when creating a font outline.

There is a particularly interesting problem with TrueType splines when the number of points representing a curved quadrant is reduced below some threshold. The shape of the curve bulges out at the corner, and a quadrant of a circle or ellipse becomes hyper-elliptical, to use a term by Piet Hein. This is a subset of a general question about mathematical representations of shapes that were created by motions of the human hand. When Donald Knuth was working on Metafont at Stanford, he would meet with interested students and colleagues at lunch to discuss a wide range of questions and problems that came out of his research. He called it the "Metafont for lunch bunch". We discussed how the mathematics of the equations affected the forms of the curves in typefaces, and we wondered what kinds of curves were sufficient for representing the aesthetics of traditional typefaces. I am not a mathematician, but I found those discussions fascinating because Knuth

was leading all of us into a barely explored realm where mathematics and aesthetics met.

Today, a quarter century later, most computer-aided drawing programs and type design programs uses Bézier cubics, and sometimes I see a tendency for recent typefaces designed directly on the computer to seem similar in the modeling of forms. I believe that this is the result of interaction between vision, user interfaces, and mathematics. Bézier splines can behave in surprising and anti-intuitive ways, at least for artists accustomed to drawing and writing on paper, and they don't necessarily resemble the motions of the human hand. When designing on screens and using a mouse instead of a pen or brush, type artists tend to be conservative, using a small number of points on the curves and adjusting the off-curve control points carefully to make smooth shapes that are easier to understand and control. The curves are usually pleasant, but they are more limited than the shapes that result from the living hand moving a traditional tool through a complex path. The motions of the tip of a Chinese calligraphy brush are especially complex and subtle, for example.

Y: So, Apple asked you to help them solve very practical problems.

C: Right. Apple, Adobe, Microsoft, and a firm called Imagen, founded by two Stanford computer scientists, asked us for advice and consultation on various font technology and aesthetic issues. In the late 1980s, Apple invited us to do some new experiments. As I said, naive algorithms for rasterization cause various aesthetic problems on computer screens and low resolution printers, like irregular stem thicknesses and spacing, irregular letter heights, loss of serifs, broken hairlines, and so on. So when Adobe developed the PostScript font format, and later when Apple developed the TrueType font format in 1989–1991, font hinting was introduced to solve those problems.

I should explain that "font hinting" is the use of computer program instructions to adjust the display of an outline font so that it lines up with a rasterized grid. At low screen resolutions, hinting is critical for producing a clear, legible text. Hinting can be generic for all sizes, but TrueType hinting also has the capability of adjusting hints for specific resolutions. This localized or hand-tuned hinting has to be done by people, who can test and view different approaches. It has become a special skill practiced by a small number of experts. A typical kind of hinting problem is to make all the vertical stems of a font have the same pixel thickness, so the text looks regular in tone and rhythm. At a given size, the

mathematical thickness of a stem might be, let's say, 2.5 pixels. So, depending on how a letter falls on the raster grid, a stem might be 2 pixels or it might be 3 pixels thick. This makes for a splotchy, irregular image. With hinting, all the stems can be forced to be 2 pixels, or 3 pixels, to enforce regularity. The actual outlines are being distorted, but the results look better to readers.

Y: So Lucida has hinting instructions inside the font file?

C: Right! In fact Lucida Sans roman was the first fully hinted TrueType font in history. Apple developed the format but didn't completely hint a font. At that time there weren't mature tools for hinting, and Apple didn't have type designers on staff, so they asked us to test the format by hinting a font, using low-level tools developed for programmers to write hinting code. Kris Holmes hinted a whole font that way. It was a lot like writing macro-instruction code. Kris showed that TrueType hinting worked in a practical design context, but the experience also made us realize that hinting was a separate kind of task from designing. We decided to stick to designing forms, not hinting them.

Y: Amazing! But I heard that hinting is not used in Apple's system any more.

C: Yes. Increasing resolution screens and new font rendering technologies, often called "anti-aliasing" or "smoothing", eventually made hinting unnecessary on later generations of screens and printers. That took more than a decade of progress, because display and printing technologies improve much more slowly than the rate of Moore's law. By the late-1990s, grayscale and color display screens gained enough market dominance that rendering algorithms could take advantage of the range of gray tones available on screens.

Y: Is this related to using anti-aliasing techniques from computer graphics?

C: Yes, the term and technique come from computer graphics. For a given glyph outline, the edge of a contour usually does not fall exactly on a pixel boundary. An edge pixel might be partly inside the contour (black) and partly outside the contour (white). Anti-aliasing adjusts the gray tonal value of that edge pixel in proportion to how much of the pixel is inside the contour or black area. The resulting edge looks smoother because the intermediate gray tone is not as noticeable as an all-black pixel. This method works better at higher resolutions. Below 100 pixels per inch, viewed at a normal reading distance, the result looks fuzzy or blurry. Above 200 pixels per inch, the result usually looks smooth without objec-

tionable blur. In between, the reader's impression of sharpness or fuzziness depends on the display technology, such as LCD or e-ink, the contrast, the reading distance, and other factors. On very high resolution screens, like the Retina screens of iPhones or iPads, the edges look smooth and sharp because the human eye usually can't perceive lower contrast individual pixels at those resolutions. Vision scientists have measured the sensitivity of the human visual system to contrast and detail and found that as detail gets finer and contrast gets lower, it is harder and harder to see fine features like tiny pixels. Conversely, for fine details to be seen, they have to be high-contrast.

Y: What about ClearType?

C: As color LCD screens with resolutions above 120 pixels per inch gained in the market, subpixel anti-aliasing became feasible. Most computer color displays use pixels made up of three subpixels: red, green, blue stripes. Usually, each subpixel has 8 bits of tone value, equalling 256 possible gray levels. A white pixel has all three subpixels turned on, while a black pixel has all three subpixels turned off, and other RGB tone values produce millions of colors in-between. Because the subpixels are adjacent, a clever hack is to represent different spatial positions and line thicknesses by choosing different colors for the whole pixel that will turn on or off selected subpixels. Microsoft developed this concept into their "ClearType" technology in Windows. It effectively triples pixel resolution in one direction, because it uses sub-pixels, which are 1/3 of a full RGB pixel.

Subpixel anti-aliasing works better at resolutions above 150 pixels per inch, where color fringe effects become nearly imperceptible. At resolutions above 300 pixels per inch, the color effects are imperceptible, and resolution seems very sharp. It is also important to note that subpixel anti-aliasing works in only one direction, either horizontally or vertically, depending on the orientation of the RGB subpixels. For better resolution of letter stems and bowls in Latin alphabetic type, the RGB sequence should be oriented horizontally. For Chinese, which has more horizontal strokes, better resolution is obtained when the RGB sequence is oriented vertically. However, devices like iPad and iPhone can display in both orientations, so it isn't possible to optimize the character forms for one orientation.

Y: What's Apple's counterpart of ClearType?

C: Apple uses similar techniques in OS X and iOS, but without a trademark name. I assume that because of the cross-licensing of TrueType font technology between Apple and Microsoft, Apple can use subpixel rendering algorithms like ClearType without

infringing Microsoft's methods or patents, but that's just my guess. However, ClearType is Microsoft's trademark, so that is presumably why Apple doesn't use that name. Apple's Retina displays use both high resolution and anti-aliasing.

Y: So you also need to think about how subpixel rendering affects the display of Lucida.

C: We can think about it, but it is hard for designers to do much about it. Subpixel rasterizing of larger type sizes on high resolution screens, which now have a major share of the market, needs no special efforts by designers because the edge artifacts from rasterization, including jagged staircase patterns, fuzzy contours, and color fringing, are small in comparison to the size of the letters and do not appreciably degrade the quality of the text image. Below 14 point, and at lower resolutions, type size is small relative to the sizes of pixels, so the rasterization artifacts are big in comparison to letter details like serifs, hairlines, and stems. The artifacts are noise that obscures the signal of character shape. In extended texts, there may be thousands of characters on a screen, so en masse, the artifacts can make text visually "uncongenial". Readers may not like the look of the text, though they may be able to read it nevertheless. Vision scientists have shown that low-resolution or fuzzy text can often be read as quickly and accurately as sharply rendered high-resolution type. The care that designers put into the shapes of characters, and the ingenuity that engineers put into rendering technology, contribute more to aesthetics than to legibility. Type is both aesthetic and informative. Well-formed and well-rendered text contributes to the pleasure of reading a text.

Recognizing the importance of designing for subpixel anti-aliasing of text types at text sizes, Microsoft commissioned several new, original font families to work especially with ClearType technology. Several were for Western scripts — Latin, Greek, and Cyrillic — and one was for Japanese Kanji, Kana, and Romaji. We tested them in my course on news typography at RIT a few years, and they all looked good. I was happy to see such strong support of artistic creativity for a new technology, from a major technology company. I should say that one of the ClearType font designers was a former student of mine, and others were friends, and that Microsoft also licenses Lucida fonts, though not as part of the ClearType set.

I think the next big challenge for designers of text type will not be pure legibility, although that is the worthy goal of most text face designers and is achievable with existing designs in current rendering technology on high-res screens. Instead, I expect

to see more emphasis and experimentation with expressiveness in design, coupled with congeniality for the reader. In the past five centuries of development, Latin alphabetic typefaces have become highly refined in their forms, weights, patterns, and variations, and many have proved to be legible over centuries. More than half of the new novels published in the US in the past decade were composed with "Old Style" type designs based on typefaces first cut more than 250 years ago. Some of the designs, like various faces based on those by Garamond and his contemporaries, were first cut more than 450 years ago. So, at least for print book readers, the great old serifed fonts of the past are still the great new fonts of today, in digital format.

Digital design tools and rendering enable greater precision and regularity in type forms, but the risk is that the designs look boring — too regulated, too repetitive, too rigid, too homogenized. Randomly adding irregularity doesn't improve the appearance — the designs then look boring but awkward. Some graphic and interface designers want neutrality in typography, but I don't believe that any type design is truly neutral. Every typeface carries some degree of expressiveness, even those intended to be plain, simple, and neutral. For example, a user-interface in Helvetica expresses a different feeling than one in Lucida, but the two designs are similar in weight and x-height. Helvetica is more modernist, Lucida more humanist. Helvetica more carved, Lucida, more handwritten. Helvetica more tightly spaced, Lucida more open. A Swiss poet made a memorable comparison of the feeling of Helvetica compared to Syntax Antiqua, a very readable sans-serif typeface by Hans E. Meier, which is even more closely based on humanist handwriting and early Renaissance typography than Lucida. The poet said, when he reads a page in Syntax, it is like walking through a field of flowers, but when reading a page in Helvetica, it is like walking through a field of stones.

So, a problem for future designers will be: how much expressiveness to put into a type. What expression does the design convey to the reader? For the reader, highly expressive typefaces are lively but can look too complex for long texts. Free scripts can look graceful but may seem too undisciplined for modern readers accustomed to rigidly regular fonts.

When technology changes, there are opportunities for new designs. We can find many historical examples. More than 50 years ago, typography shifted from metal to photographic technology. Hermann Zapf's Optima, first created for metal typography, became wildly popular in photo typography because it gave greater expressiveness to the sans-serif genre.

Optima's subtly flaring terminals and classical letter structures brought a hint of Renaissance proportions and humanist handwriting into a modern idiom, through a new technology that crisply reproduced designs photographically and lithographically, without the usual wear and ink squash of metal type. Yet, the subtle qualities that made Optima so successful in photo technology were difficult to render in early digital typography because of low resolutions on screens and printers, so Optima lost popularity in laser printing. Instead, Zapf's Palatino gained popularity in desktop laser printing because it conveyed some of the handwritten vigor of Renaissance typography and calligraphy even in low resolution of 300 dots per inch. Today, as digital resolutions increase, Optima is regaining popularity for a new generation of graphic designers. We may see new type designs for screens that enjoy similar popularity in the new media of e-books, smart phones, and pad computers.

I believe that expressiveness is also an interesting challenge for East Asian scripts. Chinese type styles derived from woodblock printing, like the Song/Sung styles, were adapted to metal typography and are now widely used in many variations in a large range of sizes in digital typography. The same is true of the related Mincho styles in Japanese typography. The rectilinear structure of this type genre, which may have made it easier to cut in wood, makes it seem stiff and rigid but functional. It may be that Song style was easier to cut and cast in small sizes of type, which would have made the style more economical because small type sizes use less paper, and are thus more widely used.

Typefaces based on brush-written Chinese scripts have more handwritten grace but historically were more difficult to adapt to metal typography, and probably that is why they are less popular than Song or Mincho styles. Digital typography relaxes the technical limitations on producing and printing fonts, and makes it easier to "draw" digital characters, so we are beginning to see more expressive styles in Chinese and Japanese typography, but mostly for "display" in advertising, headlines, and other contexts, at relatively large sizes. Many of the recent fonts are not in traditional calligraphic styles, but are fanciful designs, like clouds, fat fish, childlike writing, blurred writing, and so on. Perhaps some of these were already known in hand-painted signs and banners, and now can be made into type. If "folk" styles are getting made into type, that is fascinating. In American music, folk styles went mainstream because of the recording industry and we got jazz, rock 'n' roll, and country and western

musical genres, which have since gained worldwide popularity. However, America has not produced a "breakthrough" folk typography, probably because lettering art, calligraphy, and typography have not been folk cultures, but the practice of literate elites. The ancient literate traditions of China, Japan, and Korea may, however, include styles of writing that could become newly popular in digital form. And, of course, young designers do not always want to follow old traditions, and instead invent new styles.

I think this is an exciting challenge for designers in China, Japan, and Korea — to capture the expressiveness of classical styles and adapt them to newer technology, without seeming quaint, old-fashioned, or reactionary, and to find interesting historical styles worthy of revival, but also to invent new styles. These trends are already happening in display types, used in large sizes, but the big challenge is, how to produce those kinds of expressiveness in text types that can be read at small sizes.

In English language book publishing, sans-serif fonts are very rare in literature of any kind, whether important literature or popular genres like crime, romance, and science fiction. Fiction is generally serifed. Books about graphic design, photography, and modern art, however, use sans-serif types fairly often, so the choice of type style depends on the content and on the reader. I wonder if similar distinctions occur in East Asian publishing.

The recent popularity of Japanese "cell phone novels", which are usually about the lives of young people and often written by young women, are said to use more hiragana characters than traditional Japanese literature. I wonder if this increases interest in expressive hiragana fonts, when cell phone novels are published in print. There are already many expressive kana designs, which can be combined with appropriate weights and forms of Kanji to achieve subtly different text effects. When there is a shift in literary taste, there can also be a shift in typographic taste. Another interesting mixed writing system is the Korean, which uses Hanja characters based on Chinese, along with the unique Hangul alphabet. Compared to the Latin alphabets, Hangul more accurately represents the significant sounds of speech. So, I wonder whether literary expression that favors Hangul motivates trends in the graphical design or usage of Hangul fonts. Do font styles reflect literature? Are Korean pop novels and cell-phone novels using more Hangul than Hanja characters? The Korean Hangul writing system was sans-serif in early examples, but late styles became similar to brush-written characters.

Y: What about different weights in Lucida?

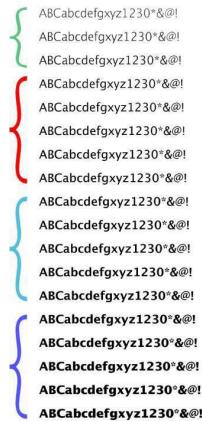


Figure 18: Spectrum of possible weights for Lucida Sans. Top group = “light” weights; second = “normal” weights; third = semi-bold weights; bottom = “bold” weights.

C: Here (fig. 18) is a series of experimental weights for Lucida Sans. The top group is for light weights, the second group for normal weights, the third for semi-bold weights, and the bottom group for bold weights. In the first generation of Lucida fonts, the low screen and printer resolutions could not support such fine gradations of weights, so we made only a few weights: normal, demibold, and bold. Now, higher-resolution display technologies and anti-aliasing techniques can render finer weight gradations, so we have designed additional weights of Lucida Sans, to be released next year. By studying the weights of popular text typefaces today, and also going back hundreds of years, we concluded that there is no single ideal weight, but a range of preferred weights, depending on printing quality, reading conditions, and, in digital displays, screen technologies.

At RIT, I did a study of “just noticeable differences” in the weight of a sans-serif face. For a given weight, how much darker must a slightly bolder weight be for a reader to notice that it is darker? The results appear to follow the Weber-Fechner law in psychophysics, which says that perception of difference is proportional to stimulus. I found that for a “normal” font of a certain weight, a just-noticeably darker font needs to be approximately 2.5% bolder than the normal weight. The same is true for a bold weight: the next perceptibly darker weight must be 2.5% darker than the bold, so perception of weight difference follows a geometric progression.

Y: The weight spectrum reminds me of the Frutiger numbering system!

C: Yes, Frutiger was a pioneer in the numbering of typeface weight systems with his Univers family and later with his Frutiger family and others. He saw

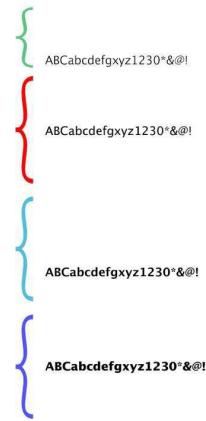


Figure 19: Current weights of Lucida Sans. Assuming stem weight of “normal” = 1.0, then “light” weight = 0.75 x normal; “demibold” = 1.5 x normal; “bold” = 2.0 x normal.

that typographic weight nomenclature was a confusing mess. Different designers, type foundries, and font vendors used different and incommensurate names. Frutiger rationalized weights within Univers and designated them with two-digit numbers. I always liked that. Recently, a three-digit numbering system has been developed for Univers, to incorporate additional weights and widths. It is useful but doesn't exactly match the original two-digit system, which makes it confusing for me because I remember the older, simpler system. Around 20 years ago, Peter Karow, developer of the Ikarus software for type digitization, made an interesting study of the statistics of typeface weights, using a large digital font database. He made a reasonable proposal for rationalizing typeface weights in an 11-step system but it was not adopted. Today, W3C recommends a set of font-weight names and associated numerical values in a 9-step system, but it is, to my mind, inconsistent with existing progressions, arbitrary, and too limited, so I don't see it as an effective solution. I'm afraid, it is a muddle that won't be cleared up soon, if ever.

Y: What's the current weight of Lucida then?

C: For Lucida Sans normal, the stem thickness is 18% of the x-height. Lucida Sans demibold is 1.5 times the normal stem, and the bold stem is 2.0 times the normal. (See fig. 19.) This approximates a progression based on the square root of 2. However, weight measured by ratio of stem to x-height, which designers like, is not the same as weight measured by percentage of black pixels in total text area, which an engineer might prefer. Using pixel area weight measure, Lucida Sans normal is roughly a 22% gray tone. Lucida Sans demibold is approximately 29% gray, and Lucida Sans bold is 36% gray, which is 1.6 times the normal weight. Thus, the gray tone

progression does not increase as much as the stem-weight to x-height ratio, because of the way weight is distributed in a Latin typeface — more of the weight is in the x-height region, less in the ascender and descender region.

Text typefaces appear to cluster into weight groupings. The normal weights of serifed roman text faces tend to have light gray tones, ranging from around 14% to 18% gray. Serifed types designed for screen display tend to be somewhat darker, around 18% to 22% gray tone. Sans-serif fonts for print and screen also tend to be darker, ranging from 19% to 23%. Of course, there are lighter and darker weights in many typeface families; I'm talking about what are called "normal" or "regular" roman text weights.

As a side note, Chinese fonts also cluster into tonal groups, but to measure the average gray tones is challenging, because the number of strokes in a character and therefore its density varies much more than in Latin typefaces, and the frequency distribution of characters can vary according to content and usage. In my very rough estimates, Song style faces have average gray tones that cluster like traditional serifed Western fonts, but slightly darker, ranging from 15% to 20%. Sans-serif Chinese fonts tend to be darker yet, ranging from roughly 22% to 35% gray tone. However, I guess that weights darker than 30% are not often used in running texts. Kanji fonts cluster into similar tonal groups. I hope that type scholars in Asia will explore some of these patterns of usage.

Back to Lucida — to make Lucida Grande work well in Apple's OSX font menu, Apple preferred the designation "bold" to "demibold", so Lucida Grande Bold in OSX is the same weight as Lucida Sans Demibold in Windows. I regret the confusion — another difference between operating systems and platforms. Weight measurements, names and numerical values remain an unsolved problem of lack of standardization, in part because of the technical needs of various systems, and in part because designers simply make weights the way they think looks best.

3 State of the art — smart fonts

Y: Interesting. What other new technologies are you involved in when designing typefaces?

C: Apart from computer graphics techniques and higher resolutions, an important font technology is the glyph substitution technique used in OpenType.

Glyph substitution makes math fonts less cumbersome because different forms and sizes of glyphs can be substituted according to context. In Arabic typography, smart fonts are aesthetically functional. They enable easy use of context-sensitive



Figure 20: Demonstrations of the joining structure of Lucida Handwriting compared to Kolibri, a script design by Kris Holmes for URW, developers of the Ikarus font software used by Bigelow & Holmes.

shape variations that are aesthetically necessary in Arabic scripts. This encourages artistic expression and experimentation, both in capturing traditional styles and in imagining new styles. In terms of glyph variations, Latin alphabet fonts were simplified during the first hundred years of typography, with most ligatures, abbreviations, and alternate forms eliminated for economic reasons. So, smart fonts are not crucial for Latin alphabet typography, but do have artistic and ornamental value. Hermann Zapf's Zapfino, a graceful yet free script with glyph substitution, has become very popular. Some of Kris Holmes' scripts like Apple Chancery, which has many glyph variants, and Kolibri, which has intricate joining, also show the aesthetic possibilities of smart fonts. Jim Wasco's Elegy script also shows elegant use of OpenType.

Before OpenType, Apple invented a similar technology called TrueType GX, later called AAT. The software that renders text parses the strings for certain combinations and contexts of letters, and, when they are found, the software substitutes alternates from the font if the substitutions have been programmed into the font. A common example in English and European languages is the f-ligatures. To keep the dot of the letter 'i' or the top of the letter 'l' from bumping into the upper arm and terminal of the 'f', typefounders used to cast special combinations of 'fi', 'fl', 'ffi', and 'ffl', and more rarely, 'fj', for words like "fjord". A few like 'fi' and 'fl' are common in most fonts today. When we were designing Lucida, glyph substitution wasn't available so we designed the 'f' with a short top arm that didn't collide with the 'i' or 'l'. In Lucida Grande, several f-ligatures are available, like 'fi', 'fl', 'ff', 'ffi', and 'ffl'.

Kris continued to experiment with more complex character sets. We designed Lucida Casual with three alternative styles, though two of them have not been released because we were experimenting to see

if glyph substitution made sense for them. However, glyph substitution is often not necessary, even for a lively script. You can see that in Lucida Handwriting. Kris crafted it so all the end strokes were placed in a single horizontal line. It looks like free handwriting, but has a simple joining method. As another example of alternative forms, Kris worked with Peter Karow at URW to design the Kolibri script, which has a more complex joining pattern than Lucida Handwriting, so that every character can join elegantly, but that requires many alternate forms. URW++ has now produced it in OpenType (fig. 20).

Y: Amazing experiment. So now we have four different 'e's. This script is really elegant.

C: Yes. Kris has a special liking for lively script faces. (I think I am permitted to boast on her behalf!) She studied dance and choreography for years as well as studying calligraphy, so her type designs learned many things from choreography, especially a feeling for motion and rhythm but also a sense of order within complexity. When Apple created a "smart" font technology based on their TrueType, it was first called QuickDraw GX in the mid-1990s but later evolved into Apple Advanced Typography, or AAT. In AAT, there can be several degrees of ligature control, old style figures, small caps and drop caps, swash variants, and alternative glyphs.

Y: This sounds very similar to Microsoft and Adobe's OpenType.

C: Exactly. When Microsoft wanted to use Apple Advanced Typography, Apple refused to license it, so Microsoft and Adobe worked together to create OpenType, which is technically somewhat different, but provides much of the same functionality. But back to QuickDraw GX and AAT — when Apple was developing the new font technology, they showed us a page like this (fig. 21). It's chancery cursive writing by Ludovico Vicentino degli Arrighi, in a wood block printed book published in 1522. Apple said, well, okay, we can do character substitution now, and technically we could produce a page like this, but we need you to design a font that would enable us to do that.

Y: So what's your design procedure?

C: The first thing we did was go back to our calligraphic teaching manual from Lloyd Reynolds, who was our calligraphy teacher at Reed College, in Oregon. Kris and I studied with him, at different times. Steve Jobs took calligraphy courses at Reed, too. Here's a picture (fig. 22) of Reynolds, standing outside his calligraphy studio at Reed College in 1967 and a sample of his italic handwriting.

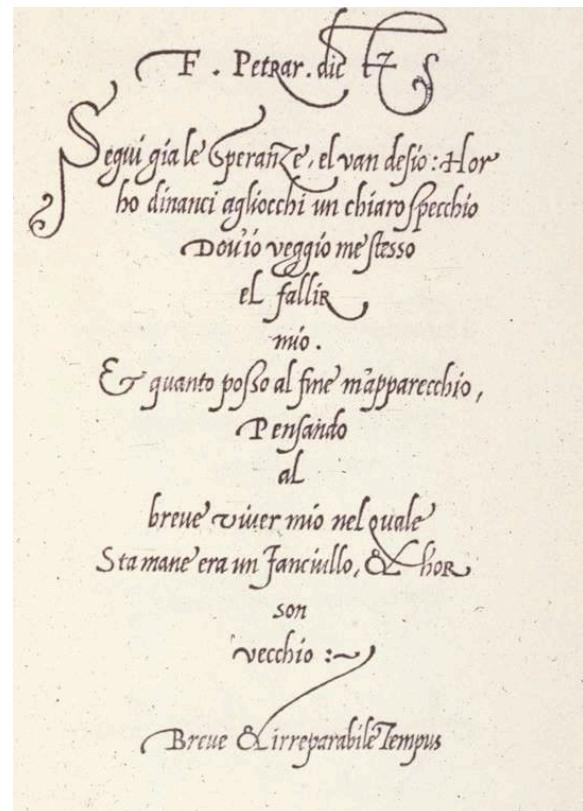


Figure 21: Chancery cursive writing by Arrighi (Ludovico Vincentino), wood block print in 1522. Apple wanted to be able to do this on their screens.

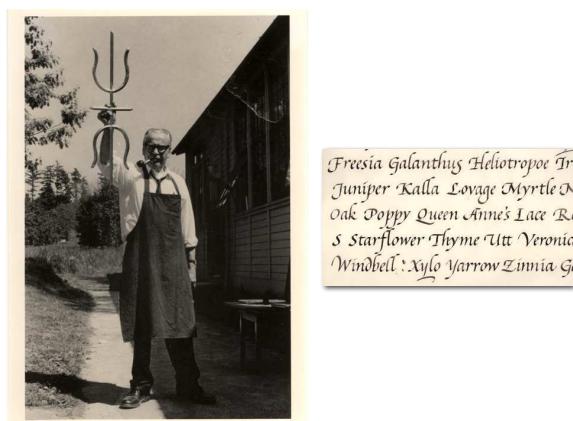


Figure 22: Lloyd Reynolds, calligraphy teacher of Kris Holmes and Charles Bigelow. Standing outside his calligraphy studio at Reed College, circa 1967. A sample of his italic handwriting.

mencement, he printed it out so all of his calligraphy students could have a copy of it.

Y: How's your reproduction process based on his teaching manual?

C: Arrighi's manual is clear and elegant, with many fine flourishes, but the letters were cut in wood and are a little more angular than examples of his actual



Figure 23: Variations of 'k'.

Lucida Calligraphy *Apple Chancery*

Lucida Calligraphy *Apple Chancery*

Figure 24: Comparison of Lucida Calligraphy to Apple Chancery; both designs are chancery cursive by Bigelow & Holmes. Apple Chancery is more like the form of calligraphy taught by Lloyd Reynolds, based on Arrighi's models. The top pair are both set at a body size of 28 pt; in the lower pair, the Apple Chancery size has been increased to equalize x-heights.

handwriting and of other scribes of that era, so Kris wrote all the characters with a pen and worked out as many variants of every letter of the alphabet as she could dream up. For example, if you look at the lower case 'k', there's a very simple 'k', a more complicated 'k', a 'k' that would go at the beginning of the line, a 'k' that would go at the end of the line, and so on. (See fig. 23.)

Kris created her samples based on Reynolds' teaching and manual, and we enlarged them, and then we redrew them. And we made a few changes to make them sturdier looking for typographic use so the hairlines were thickened up a little bit and the characters were made a little wider than they would be just with a pen written character. The result was Apple Chancery. A "chancery" was a medieval or Renaissance clerical office where scribes wrote the documents needed to organize a kingdom or city or organization. A special kind of handwriting used in Italian Renaissance chanceries came to be called "chancery cursive". So, we suggested that this italic handwriting, designed for Apple, could be called "Apple Chancery".

Y: It looks like Lucida Calligraphy.

C: Yes, both were based on our study of italic handwriting with Reynolds, who based his teaching on calligraphers like Alfred Fairbank, who based his on the works of Arrighi and other Italian calligraphers of the 16th century. Lucida Calligraphy has a big

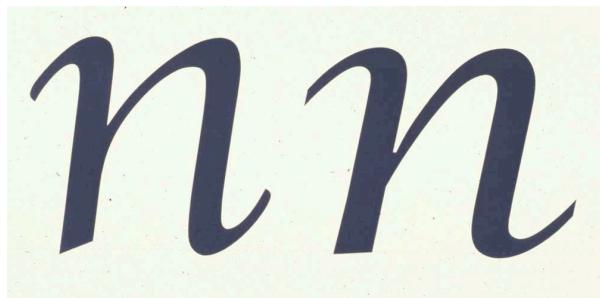


Figure 25: Apple Chancery (left) compared to Lucida Calligraphy (right).

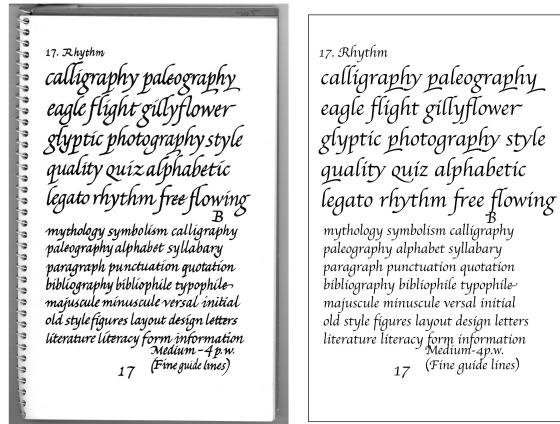


Figure 26: Reynolds' hand-drawn calligraphy compared to Apple Chancery.

x-height, much bigger than the Italian Renaissance models, so it can align with other Lucida fonts. Apple Chancery stands alone, so it has a smaller x-height, more like the traditional chancery handwriting and fonts of the Renaissance. Here (fig. 24) is the comparison of Lucida Calligraphy (big x-height) to Apple Chancery (small x-height), at the same body size. The type with the smaller x-height looks smaller. But when we designed Lucida Calligraphy, the old Canon printing technology tended to increase dark shapes and some of the details would clog up, like the 'n' here. We made modifications to the shapes to prevent this from happening. But in Apple Chancery we didn't need that any more (fig. 25).

Y: So the shape is more beautiful.

C: Apple Chancery is more like the Renaissance proportions of calligraphy taught by Lloyd Reynolds, based on Arrighi's models. We produced a huge character set for this font. In the end it had more than a thousand glyphs in it. This was how Apple Chancery came into being.

Y: So Apple Chancery was the testbed of smart font technology?

C: It was the most extensive use of Apple's TrueType GX font technology in its first release. Apple

also used smart technology in other fonts released around the same time, so Apple Chancery wasn't the only pioneering smart font, but it was the most ambitious at that time. Zapfino is a smart font that came later, in OSX, with even more variant characters in a free calligraphic style. When we finished the Apple Chancery project, Kris made this (fig. 26): on the left is a page from Reynolds' calligraphy book. And on the right is the same page duplicated in Apple Chancery. You can see the difference. The typographic forms are a little lighter. They're a little wider, not quite as rich in variation. But we were very pleased with this, because I think that the spirit of Reynolds' calligraphy is in here. Steve Jobs was at Reed a few years after Kris. He also studied calligraphy there, so he was influenced by the same ideas from Lloyd Reynolds, which he described in a commencement speech at Stanford some years ago. So these traditions and interactions fit together in a coincidental but intriguing way. Apple Chancery is intended to honor Lloyd Reynolds' memory, and in a way also commemorates Steve Jobs' experience studying calligraphy.

Y: Given that all recent fonts are moving to incorporate AAT or OpenType features, what's your recent plan for Lucida?

C: Good question. We recently adopted OpenType for a very functional purpose: a new version of Lucida Math for \TeX . Almost 20 years ago, we worked with the firm of Y&Y to make a set of Lucida Math fonts in PostScript Type 1 format for \TeX . Berthold and Blenda Horn did a lot of work to make Lucida fonts compatible with \TeX . Since then, the Unicode standard has added several blocks of math symbols and alphabets, and OpenType enables glyph substitution. To upgrade Lucida to OpenType, we added more math symbol sets, a new math script alphabet, plus Greek and Cyrillic alphabets, and we encoded all the characters in Unicode. Previously, we offered basic text fonts plus \TeX -oriented math fonts like "Math Italic", "Math Symbol" and "Math Extension" for \TeX . Those are now combined into one math font in OpenType (<http://tug.org/lucida>). Karl Berry coordinated the project on behalf of TUG, we designed the new glyphs and Khaled Hosny combined the new character sets with the older ones and built the fonts in OpenType format. Several people from the \TeX community helped test and critique the fonts. Mojca Miklavec, Hans Hagen, Ulrik Vieth, Will Robertson, Michael Sharpe, Taco Hoekwater, Bogusław Jackowski, and Barbara Beeton. I hope I got all their names right. An international undertaking.

The new Unicode standards for math symbols incorporate style variations as semantic variations.

As one small example, in addition to the usual text versions of 'a', we provide separate math versions for upright 'a' and italic 'a', as well as sans-serif and bold variations, which have different semantic meanings in math.

Y: Yes. Because in math equations, upright is used to mark labels, while italic is for variables. Bold marks are used for vectors.

C: So now in Lucida Math OpenType, we include all these variations that are specified in Unicode. Now there are more than 3100 math glyphs in Lucida Bright Math and around 1700 in Lucida Bright Demibold Math.

Y: Amazing. So you are using the new OpenType MATH table feature introduced in Microsoft Word 2007?

C: Yes, but we didn't make the math tables, Khaled Hosny did them. First, Kris and I designed the glyphs, using various tools, old and new, including Ikarus, Illustrator, and FontLab, and then Khaled Hosny assembled the fonts and generated the MATH tables using FontForge.

Y: The latest \TeX engines like $\text{Xe}\text{\TeX}$ and $\text{Lua}\text{\TeX}$ fully support OpenType, so it's much easier to use them.

C: Yes, that's why TUG suggested we make the upgrade. We also took the opportunity to redesign the math operators. When Donald Knuth designed \TeX and his Computer Modern typeface, he used relatively large operators compared to the alphabetic characters. I think perhaps as a mathematician he thought the operator relationships were more important than the variables themselves. But, when we first designed math characters for Lucida in the early 1980s, we made the operators relatively small because we were thinking that the symbols should be proportioned like the alphabetic characters, and that it would be helpful if most of the operators were the same width, either like figures or some other set width, so the symbols could easily be used in tables. Later, we agreed more with Knuth's practice, so we increased operator symbol sizes for the Y&Y Lucida Math fonts. And, after more years of experience working on math fonts and seeing them used by mathematicians and computer science, we believed that Knuth had been right all along, so we increased the sizes of the operators again when making the OpenType Lucida Bright math fonts. Now they are close to the proportions Knuth chose more than 30 years ago.

Y: But I still like the original flavor. Maybe you can leave this as an option for users?

C: We kept some of the smaller symbols as alternates in the fonts for those who preferred them.

The older operator designs are also in the PostScript Type 1 Lucida math fonts, which are still available from TUG, so they aren't lost.

Y: You mentioned that there is a demibold version of Lucida Bright math.

C: Yes, when we were working with Y&Y years ago, we added bold operators because Y&Y and some of their customers said, "We need bold for the symbols as well!" because bold is a semantic category for math variables; logically, bold could apply to operators, too, though currently, not all operators have a semantically bold form. So, for Lucida Math OpenType, we made a whole math font in Demibold. Not only because bold characters have semantic meanings, but because mathematicians and technical authors are logical — they think, if we have bold letters, bold greek, bold scripts, and so on, why don't we have bold symbols? Because mathematicians keep thinking of new ideas and need new symbols to represent them, they keep making little bits of new work for type designers. We try to keep up, but math fonts are never really finished, because mathematics keeps expanding. It isn't clear which math characters really need to be bold, so Lucida Bright Demibold Math doesn't offer bold versions of all the characters in the normal weight. We added a bold typewriter to the Lucida Demibold Math font, because there is a Math Typewriter alphabet in the normal weight Lucida Math, and the same for a bold script, in two styles, chancery and English roundhand, and bold arrows as well as bold operators. I expect we will get requests for more bold characters.

Y: Do you make the symbols bold by hand or by using software to automatically make it bolder?

C: Design by hand. We use software to input shapes and edit contours, but not to make automatic bolds. Some math symbols are easy to embolden because their geometry is simple and clear, but some take a little more work, though most are not as difficult as emboldening alphabetic characters. Generally, you get a sharper, crisper, better design by emboldening by hand because you see what you are doing. For outline font formats, there is no single algorithm to make good bold weights, though I have seen that the algorithms are getting better at making pseudo-bold weights. Early methods made a smearable mess of the shapes. In Metafont, however, there is an easy way to embolden shapes, because Metafont can use a pen metaphor: a nib of a certain size, shape, and orientation follows a path and the image of the character is the trace swept out by the nib. You can keep the same path but make the nib bigger to make the shape bolder. More subtle methods change the size, shape, and orientation of the nib.

An outline format like TrueType does not use that metaphor. Because of its pen metaphor, Metafont is closer to traditional writing than to traditional metal typography, which used carved outlines.

For bold characters, a design challenge is, how to prevent acute angled joins, like where the hairline of the arch meets the stem in an 'n', from clogging up when printed with heavy ink or toner? In the first versions of Lucida for 300 dot-per-inch printers, we opened up more white space in those areas, but as printers improved, we removed the cut-outs. On screen, there can be the opposite problem — the backlit background can make characters look lighter. Digital technology keeps presenting new challenges for designers.

Y: Can the Lucida Math characters be used without TeX?

C: Yes, the characters are encoded in the fonts with Unicode. Applications that let users find characters by Unicode code point or that show the whole glyph set let users access the characters. Equations may not look quite the same when Lucida fonts are used with Microsoft Word's math engine, because the Lucida Bright Math fonts don't have exactly the same metrics nor all the same characters, as Microsoft's Cambria Math font. Our goal was to augment the TeX-friendly Lucida Math fonts for OpenType, not emulate Microsoft's font, but we always enjoy designing new characters, so if user feedback tells us that we should include the Cambria math set as a subset of Lucida Bright Math, I expect we will eventually include those characters.

Y: You mentioned chancery script and roundhand script. Are both included, and what is the difference?

C: Originally, Y&Y used our chancery script, Lucida Calligraphy for the default math script. A chancery script for math is found also in Herman Zapf's script capitals in the Euler fonts. A chancery script is "cursive", which means a fast, "running" style, but the letters usually don't join. Some TeX users asked for the English style of roundhand script, which is more common in math composition than the chancery style. In English roundhand, the lower-case letters join, and there is a strong difference between thick stem and thin hairline strokes. Kris designed a true English roundhand face, based on her studies of English writing masters. That is now the default in the Lucida Bright OT math fonts, but the Lucida chancery characters are still in the font as alternates for those who prefer them. Both the chancery and roundhand scripts have normal and bold weights. A very different set of capitals in Lucida Math is the Blackboard Bold set, in which the forms are based more on geometry than handwriting, but they are

not purely Euclidean constructions — the capitals relate to the Lucida Sans capitals. We also made a bold Blackboard Bold for Lucida Bright Math demibold.

4 Metafont and the TeX world

Y: You mentioned Metafont's pen metaphor. Do you think the idea of the pen is still useful in the design of a font?

C: Yes, the *idea* of the pen is still powerful, but the long history of metal typography firmly established the outline metaphor. Whenever a type designer, called a punch-cutter until the end of the 19th century, tried to imitate a handwriting style, he had to cut it in steel. Also, the letters had to be cast separately, and for economy and efficiency, there could not be many variant letters or ligatures — characters tied together. Probably the greatest punch-cutter of scripts of all time was a 16th century Frenchman named Robert Granjon. He cut many different fonts of roman, italic, chancery, and cursive blackletter, as well as Armenian, Cyrillic, Syriac, Arabic and other non-Latin scripts. So, punch-cutting could imitate handwriting in the hands of a master. It is much easier to create type today; it doesn't have to be laboriously cut in steel, but even now, most typographic scripts are created as outlines.

In early digital typography, companies were in a hurry to reproduce metal or photo type in digital form. Helvetica, Times Roman, etc. Even Lucida, an original design for digital, was based on an outline metaphor. But, at Stanford, Knuth explored the pen metaphor in his own creation of Computer Modern, and also commissioned the Euler font designs from Hermann Zapf.

I should say a little about how the Euler fonts were produced in Metafont. Zapf drew the letters as outlines; after a career of four decades, he knew well how to render handwriting in outline drawings. The reverse process was much harder for those of us working for and with Knuth at Stanford: how to turn drawn outlines into pen-based paths in the Metafont metaphor. Knuth himself could have done it, but he was busy finishing TeX and Metafont, so he assigned the project to one of his talented graduate students, but progress was slow, so then my students also became engaged in the project, and yet it still went very slowly. The students despaired of ever getting true Metafont characters to match Zapf's drawings. Eventually, I advised them to give up on the "meta" aspect and the pen metaphor, and instead digitize Zapf's drawings as outlines, using a simple-minded hack: set the Metafont pen nib to be very small — one pixel — on a high resolution field, and code the outline contours as paths. This worked

well, the characters matched Zapf's drawings, and the production went much faster. However, the resulting characters were not "meta". Normal weight could not be turned into bold by changing pen nibs, serifs could not be altered by changing nibs. Outline representations of characters are basically unintelligent blobs, whereas Metafont representations have structure, but we were not able to reconcile these two different approaches.

I regretted that neither I nor my students could see how to solve the more difficult problems. Given a shape traced by a pen or brush, we can digitize the graphical trace in various ways, but given a drawn outline, it isn't at all clear what path and what pen produced that shape, nor even if that shape can be made by a pen and a path. How to make the characters "meta" — that is, how to design them so that bold, narrow, and other variations can be produced by substituting virtual pen nibs — adds another layer of difficulty. I've always felt guilty about turning an intellectually fascinating but very difficult problem into a simpler but achievable solution under constrained circumstances. Nevertheless, there were practical advantages to the outline solution. After the Euler fonts were produced as digital outlines with Metafont, a group of mathematicians and programmers were able to translate them into the PostScript Type 1 format: Berthold Horn at Y&Y, Henry Pinkham and Ian Morrison at Projective Solutions, and Douglas Henderson at Blue Sky Research. A few years ago, the Euler fonts were revised with further contributions by Zapf [4].

I should mention that the Euler project at Stanford was using Metafont79, not the current Metafont(84). In mf79, only the pen metaphor was available; in mf84, outlines are also directly supported. Indeed, it was partly because of the Euler experience that Knuth completely rewrote Metafont to support outlines as the primary drawing mechanism.

The pen metaphor is still valid as inspiration, but it has mostly been ignored in commercial font development. Today, nearly every digital font designer uses a visual application like FontLab or Fontforge, not Metafont. So I don't think there is any need to use the pen metaphor for actual production. Even to capture handwriting, as long as the shapes produced by pen strokes can be turned into outlines, designers are happy about it. I am sometimes sorry to see that the spirit and grace of the moving hand and tool, whether pen, brush, or reed, are lost in modern typographic technology, but now that the basic problems of outline font technology are solved, perhaps someone in the future will work on restoring the human action.

Y: So you think Metafont is too hard for designers.

C: Yes, at least for visually oriented designers. Metafont is mathematically based, whereas most designers rely on their visual intuition and avoid mathematics. Metafont uses an abstruse programming language to describe characters, which must be written and tested like computer code, and which makes it nearly impossible for visually trained designers to learn to use it. The intersection of Programming Experts and Design Experts is nearly the Empty Set, though some younger designers both write code and create typefaces, but in the outline metaphor. If Knuth had developed a more user-friendly interface to Metafont, or if someone else had successfully worked on a project to automatically record a real pen or brush movement and determine the virtual pen that produced the resulting shape, I think the pen metaphor would have been more widely adopted. Remember, too, that at the output end, all the font engines for screens and printers were biased toward outlines, beginning with PostScript, and followed by TrueType. Nevertheless, a fair number of fonts have been produced with Metafont, especially for non-Latin alphabets and character sets, symbol sets, and others.

Y: I think the same for \TeX too.

C: \TeX fits in with a technical, logical intellect. For visual designers who prefer WYSIWYG interfaces, \TeX is difficult. Very few graphic designers or typographers appreciate it. However, that is not true for the thousands of mathematicians, physicists, computer scientists, and others who use \TeX to write scientific and engineering papers.

A personal anecdote to support that claim: My neighbor is a retired mathematician, Norman Alling. He wrote a book on real elliptic curves and taught himself \TeX in his 50s, so he could compose his book and papers himself, and he still uses \TeX now, in his 80s. He says \TeX liberated math journals and authors from dependence on commercial math typesetting, which was slow, expensive, and fraught with typographical errors needing proofreading and correction. When I told him that some people suggested that Knuth could have better spent his time finishing the Art of Computer Programming books instead of spending a decade developing \TeX , he replied: Oh no, \TeX liberated so many mathematicians and scientists from the bottleneck of typesetting that it was a great boon to all of math and science, more important for the world-wide science and technical professions than Knuth's unpublished books on computing, however excellent they might be. That's just one opinion, of course, but it suggests how liberating \TeX was and still is. And, of course, Don Knuth is still working

on his books, and many people hope for his success in finishing them.

As a side note, did you know that Knuth's work on typography was anticipated by the Italian mathematician Giuseppe Peano, a founder of mathematical logic? Peano was concerned with the precise forms of mathematical notation in print, and was frustrated by the difficulties of getting mathematics typeset and printed, so to further his grand ambition to publish an encyclopedia of mathematical formulae, he purchased his own printing shop and took classes in composition and printing. Typography is the graphic art that seems to appeal most to mathematicians (apart, perhaps, from the prints of M. C. Escher). Do you know the mathematician who developed the Unicode \TeX fork for non-Latin scripts? He also wrote the book *Fonts & Encodings* [5].

Y: Oh, you mean Yannis Haralambous's Omega?

C: Yes. His book is a massive volume of information. It touches on nearly every subject in digital typography, often in great detail. His Omega system deals with non-Latin typography, which Kris Holmes and I also find fascinating, but we look at it from the character design aspect, not the programming aspect. Haralambous developed an actual system. It's really an impressive body of work.

Y: Yes. And the source code has been merged into the future version of \TeX called $\text{Lua}\text{\TeX}$.

C: It's wonderful how many dedicated people continue to contribute to the expansion of \TeX .

Among computer scientists who showed early interest in digital typography were the developers of Unix at Bell Labs. In 1979, Ken Thompson, Brian Kernighan, Joe Condon, and perhaps others, wrote software for Unix systems to drive the Linotron 202, a new digital typesetter that Bell Labs had bought. They found that the Linotron 202's factory-installed software was buggy and that the font encryption prevented them from inputting their own graphics. In a brilliant summer project, amusingly and succinctly described in an internal Bell Labs report, they disassembled the typesetter's own operating system and replaced it with their own software. They also decrypted the typesetter's font encryption scheme so they could input their own graphics. As well as being less buggy, their new software was faster at processing mathematical texts, although slower at processing newspaper texts—the Labs published technical papers, not newspapers. They also developed software to input digital graphics like diagrams, chess pieces, logos, and so on. The Labs' internal report was a nice description of problem-solving by intelligence. They didn't use big brute force number-crunching to decrypt the machine's software, but

simply studied and analyzed its workings, then experimented with their own code. Looking back, it is clear that they were seeing the future, six years before PostScript printers and imagesetters revolutionized digital text and graphics imaging. Their approach could have been more widely exploited, but I believe the paper was not published and their software not distributed with Unix because of legal issues with reverse-engineering. However, their paper has finally been scanned and released for its historical interest; it's on Brian Kernighan's page on the Bell Labs site: <http://www.cs.bell-labs.com/cm/cs/who/bwk/202.pdf>. A modern revival is being reprogrammed by David Brailsford for the Document Engineering 2013 conference.

5 Beyond Latin alphabets

Y: The Lucida Grande fonts in Mac OS X have several non-Latin alphabets, like Greek, Cyrillic, Hebrew, Thai, and Arabic. Do they use advanced typography as well?

C: Yes, to some extent, but not a lot. Modern Latin, Cyrillic, and Greek fonts don't really need advanced typography like Apple's AAT or OpenType. Latin fonts may benefit from the aesthetic possibilities of glyph substitution, but they don't *need* it for legible text. In metal typography, simplification of character sets made typesetting and printing more economical, because fewer characters needed to be cut, cast, stored, and composed. Hence, by the middle of the 16th century, most abbreviations, ligatures, and variant forms of characters had been eliminated from standard roman and italic fonts. For Cyrillic type, a similar simplification took place in the early 18th century under the direction of Czar Peter the Great of Russia. Greek fonts, which could be very complex because of many ligatured forms and complex sets of accented vowels, were gradually simplified over the centuries by elimination of ligatures and variants. In the late 20th century Greek "monotonic" standard, ligatures are eliminated and the number of accented letters greatly reduced.

Typefaces based on cursive handwriting, however, tend to have more joining forms and context-sensitive complexity. Apple Chancery has more than 1,000 characters, with hundreds of variants and ligatures. Herman Zapf's Zapfino has more than 1400 glyphs, including letter variants and ligatured forms using advanced typographic features.

However, the Arabic writing system really needs advanced typographic support in order to make traditional styles practical in typesetting. The first release of Lucida Grande Arabic in 2001, a sans-serif design but in the Arabic Naskh style, definitely made

use of advanced typography, in the form of Apple's AAT system. Most Arabic typefaces today use OpenType, and many interesting and elegant Arabic typefaces have been designed in the past decade, because of the new freedoms of advanced typography and glyph substitution.

The Devanagari writing system used for modern Hindi and some other languages of India, and also for classical Sanskrit, also benefits greatly from advanced typography. For Sun Microsystems we designed a Lucida Sans Devanagari face that uses OpenType, but it was not included in Lucida Grande. **Y:** How did you expand Lucida from Latin to other alphabets?

C: Our teacher of calligraphy, Lloyd Reynolds, emphasized that written forms must have life and action. He liked to quote an ancient Chinese art philosopher, Xie He, whose first principle of painting was, "qiyun shengdong", spirit breath rhythm life movement. Nearly all typographic forms were originally imitations of handwriting, though the subsequent evolution of typefaces takes different routes. Because Lucida was based to a large extent on Italian Renaissance handwriting, we tried to base the Lucida Greek alphabet on older Greek handwriting. Kris practiced writing medieval styles of Greek, and then we modernized them into sans-serif styles. Of course, many Greek capital letters are shared with Roman forms, but by starting with handwriting for the lower-case, we tried to give it more life and action. We used similar principles for Cyrillic, though its modern forms are more directly derived from typefaces, not traditional handwriting.

Again, for Hebrew and Arabic, we first studied traditional calligraphy. Arabic writing has a long tradition of elegant calligraphy, but it is difficult to distill that to fonts that are legible in small sizes on computer screens. Apple asked that Lucida Grande Arabic look almost as big as Latin at small screen sizes, to be legible in menus, captions on icons, and so on, so we designed it as a sans-serif design in the Naskh style, based on a design we had also done for Sun Microsystems' Java Developer Kit. Lucida Grande looked very legible at small sizes, and was shown in a book about Arabic typography by Huda Smitshuijzen AbiFares [1]. Later, however, some people told Apple it looked too big when printed, so Apple replaced it with a more traditional looking Arabic font as default. However, after Lucida Sans Arabic for Java and Lucida Grande Arabic, several new sans-serif Arabics have been designed with big "looks", so the design idea has become popular.

Our first international font was Lucida Sans Unicode for Microsoft, in 1993. It was one of the first

TrueType fonts to incorporate several different alphabets — Latin, Greek, Cyrillic, and Hebrew, plus mathematical and technical symbol sets. We wrote an academic paper about how and why we did it [2].

Apple asked to include that kind of international Lucida in an operating system that never came out. It only had a code name but was never released.

Y: It's called Copland.

C: Yes, Copland. So Apple acquired a license for Lucida Sans. At that time it was not called "Lucida Grande", but when we included more glyphs for Latin-based orthographies, including Turkish, Czech, Slovak, and many others, plus Greek, Cyrillic, Arabic, Thai, and other international languages, it became much grander, so Apple thought it should be called "Grande" to emphasize its larger, more grandiose character repertoire.

Y: So in 1999 if Apple wanted to use something new, why use Lucida? It was already 15 years old. Why not ask you to create something new for the 21st century?

C: Great question. I think it would have been a good idea to do something totally new. We love to do new designs, but Apple didn't have time for the development and testing of a totally new font. A fact about text fonts is that it takes most of them years to prove themselves. Ornamental faces can become quick "hits", but text fonts are usually slow to become popular. Continuous reading is a subtle process and preference for fonts emerges slowly. Adobe considered Lucida in the very early days of PostScript, 1983–84, but Lucida had not yet been released and Adobe was unsure about whether it would be popular on the Apple LaserWriter, so they chose well known existing fonts. By 1999, Lucida was well known and proven in practice, so Apple wasn't taking a risk by making it the system font for OS X. It was already well liked by computer users.

Y: Speaking of Mac OS X, why there is no italic variant in the Lucida Grande typeface?

C: Oh, interesting question. There are true italics for Lucida Grande, but Apple did not release them with OS X. Next year (2013), we plan to release them ourselves. Lucida Sans Italic is a cursive design, based on the same Arrighi chancery handwriting of the 16th century that inspired Lucida Calligraphy and Apple Chancery, but we simplified it greatly for the sans-serif style. Eric Gill first did this with his Gill Sans Italic in 1928, and Hans Meier's Syntax italic of 1968 is also a cursive design, though he kept the humanist form of lower-case 'a' and 'g'. Lucida Sans Italic was first released in 1985. In the decades since then, sans-serif italics have become a

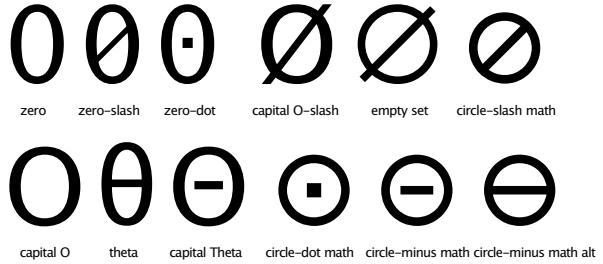


Figure 27: Variations on zero, and related forms.

style popular with several designers. For example, in Frutiger Next, released in the year 2000, a true cursive italic replaced the original slanted roman of 1976. I think it is a sign that sans-serif is continuing to mature and evolve.

On the computer screens of 10 to 12 years ago, simple forms usually looked better than complex ones, and some designers preferred the simplicity of slanted roman to the more complex cursive of true italic. In a different way, Donald Knuth used both true italic and oblique styles in TeX, for different semantics. Oblique designs are not a new idea, because sloped roman for italic was proposed by Stanley Morison in 1926, and most sans-serifs used slanted romans, not cursive italics. In our Lucida Math fonts, we provide both true italics and obliques.

Y: Here's a question from a friend. Why in Lucida Grande is the en dash actually shorter than the hyphen?

C: Your friend has spotted an interesting problem. In Lucida Grande, the default hyphen is not a true hyphen but a hybrid between hyphen and minus sign from the ASCII standard. It is longer than a true hyphen but shorter than a true minus, because people use it for both functions. The en dash is by definition one-half of an em square wide, including a little space on each side. The minus is wider than an en dash, because Lucida math symbols are fairly wide. So, when we made the hybrid hyphen-minus, it turned out wider than the en dash. I hope to adjust the disparity in the next version of Lucida Grande when we release it ourselves. Currently, it is only distributed by Apple. (Has your friend spotted any other anomalies that need fixing? Now is a good time to ask about them.)

Another interesting problem that affects technical users is the design of the zero (fig. 27). If you peek into the unencoded glyphs of Lucida Grande, you can find alternate forms of zero. The encoded form is the standard open or empty zero, nothing inside. It's a nice iconic symbol, like an empty set. However, in computing, the problem of confusion between zero and capital 'O' has been debated since

the 1960s, e.g., in the journal of the ACM (Association for Computing Machinery). So, some computer fonts have a zero with a slash. OK in English, but in Danish and Norwegian, there is a slashed capital ‘O’ and a slashed lower-case ‘o’, which can be confused with the slashed zero. Other people prefer a zero with a dot in the center, but that can be confused with Greek capital Theta, and programmer friends of mine say it is aesthetically displeasing and call it the “fly-speck zero”. For Apple, in Lucida Grande, we provided all three variants, and recommended that Apple could use whichever one seemed correct for any given localization. The open zero is the default. When Apple asked us to redesign Monaco for TrueType and System 7, we made the zero with a slash because programmers were using the font. We also differentiated the capital ‘T’, lower-case ‘t’, and figure ‘1’, for programmers. But, some people still don’t like the slashed zero. In Lucida Console, we used the open zero because there it is less likely that zero and capital ‘O’ will be confused, because the ‘O’ is shorter than the zero. But, in the next version of Lucida Console, we will use a slashed zero.

The more characters in a font, the more design puzzles and potential conflicts between design, culture, and technology we encounter. Here’s another little example, but it takes a while to explain. When we made Lucida Console for Microsoft, we were asked to include the Unicode character 010F, called the “dcaron” or “Latin small letter d with caron” (‘d’), which is used in Czech and Slovak, two related Slavic languages of Central Europe. The d-caron marks a phonetic variant of the sound represented by the letter ‘d’. In early Czech orthography, it was a little dot above the letter, and that eventually became an inverted circumflex, called a “haček” or “caron” in English. The capital form is a ‘D’ with haček above it, but in printing, the lower-case became a ‘d’ followed by an apostrophe, probably because that was easier to make in metal type. It is difficult to fit a haček above the ascender of the ‘d’ in metal type. In a fixed-pitch font like Lucida Console, the apostrophe variant is difficult to design because the apostrophe takes up space to the right of the letter. A designer can squash the width of the ‘d’ and cram in the apostrophe, which I don’t like, or fit a caron over the bowl of the ‘d’ but not above it, but some Czech readers don’t like that. For Lucida Console, we couldn’t hang or kern the apostrophe beyond the right edge of the fixed-width cell, nor put it above the ascender, because those violated a screen display rule in Windows NT, and in any case could overlap a neighboring letter. So, we made three different versions and put them in the font we delivered to

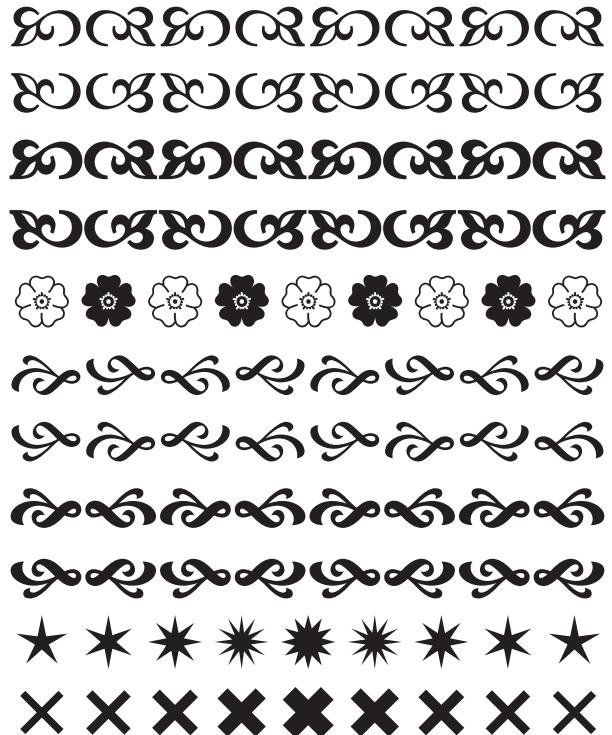


Figure 28: Wingdings fleurons (originally for Lucida). From top, the rosebud fleurons have four symmetry variants in normal and bold weights. The eglantines (wild rose flowers) have outline and filled variants. The vine tendrils have four symmetry variants in normal and bold. The stars have a range of points and the Xs have a range of weights.

Microsoft, suggesting that Microsoft ask their Czech localization experts which one is best, and use that in the standard Unicode encoding. The font got released with the caron positioned above the bowl of the ‘d’, and no one complained, until several years later, Microsoft told us that some Czech users, probably programmers, because they are the main users of Lucida Console, didn’t like that default version. Microsoft asked us to fix it, and we said, sure, it’s easy. Just replace it with the alternate character from the original font we delivered. That was maybe 10 years ago, but I don’t think it has been replaced yet. We will release a new version next year, with the preferred version, now that we know it is preferred.

6 From the present to the future

Y: I also have a question related to Microsoft. What was in your mind when designing the Wingdings typeface? Is there any connection between the symbols and their letter representation?

C: No. It is a complicated but instructive story. The characters in the Wingdings font originally came from three fonts of non-alphabetic characters called Lucida Symbols, Lucida Stars, Lucida Arrows, and Lucida Icons. We designed them to work with Lucida,

and to be useful or decorative, or both. There are several pretty ornamental “fleurons” or flowers in Wingdings, in addition to more functional designs (see fig. 28). Microsoft licensed and distributed them in a beta-test release with Windows 3.1 in 1991. Then Microsoft bought outright the icons, arrows, and stars fonts, to make a new, exclusive symbol font for Windows. The other Lucida fonts were licensed, so B&H still owns them. Back then, Microsoft Windows was distributed on floppy disks, and Microsoft found if they included all three symbol fonts, they would need an extra floppy to hold the files, so they decided to merge only their favorite symbols from the three fonts into one font.

Y: But floppies are cheap.

C: Yes, but since there would be tens of millions of copies of Windows sold, it would have cost Microsoft a lot more money. Another issue in those days was that symbol fonts had to be mapped to the keyboard for characters to be accessed. One symbol was mapped to capital ‘A’, another to ‘B’, and so on. Some Microsoft managers and font advisers chose their favorite symbols from the three fonts and had them merged them into one font. This merging meant that all the original mappings from the B&H fonts were changed by Microsoft. This font became Wingdings.

Y: But then users found interesting sequences.

C: Yes. The first discovery was that the character sequence “NYC” in Wingdings was rendered as a skull and crossbones, a Star of David, and a thumbs up gesture. This was interpreted as an anti-semitic message, in an article in the New York Post newspaper, but the popular magician, Penn Jillette, wrote a column for a computer magazine debunking the Post’s story, pointing out that the assignment of symbols to letters on a keyboard inevitably results in sequences that can be interpreted as meaningful, even when no meaning was intended. The problem was that Microsoft hadn’t sent the newly encoded Wingdings font out for beta-testing. From a technical engineering view, the font worked perfectly. It was in user psychology that the problem arose. This is why software should be tested outside a firm. Later, we were told about many other messages supposedly found in Wingdings. One was that the symbols assigned to the sequence “LBJ JFK” proved that Lyndon Johnson was complicit in the assassination of John F. Kennedy.

Y: And when 9/11 happened, you became the most reported type designer in history.

C: Yes. An email went viral on the Internet, claiming that in Wingdings, typing ‘Q33 NY’, supposedly

the flight number of the first plane to hit the Twin Towers, would show an icon sequence of a plane flying into two towers, followed by the skull and crossbones symbol and the Star of David. But, the real flight number wasn’t Q33 NY. Somebody just made it up. In the Wingdings design, the rectangles were icons for documents with text, not buildings. And the font was made 10 years before 9/11. But none of that mattered to gullible journalists who didn’t check the supposed “facts” they read on the net, and asked me after September 11, 2001, “Why is Wingdings associated with the terrorists?” This was back when some naive journalists still believed what they read on the web.

Y: [Laugh.]

C: I also heard that people were typing the names of their husbands and wives in Wingdings, to find clues to whether their spouses were “cheating” on them. People like to find hidden messages, even when the messages are noise, not signal. There is a whole field of study about why people like to believe in hidden messages and conspiracy theories.

Y: So Wingdings became a hot topic at that time. Many people talked about it. Has there been an increase of public awareness and interest in typography in general over the years?

C: Yes, very much. People talk and write much more about these issues now. About legibility. About whether some typefaces make a text more believable, some less believable. 30 or 40 years ago such discussion only appeared in design journals. But now I see discussions of fonts every week in newspapers like the NY Times or magazines, and of course on the Internet. I recently saw a book of Guatemalan poetry entitled ‘Times New Roman Punto Doce’ (Times New Roman 12 point).

People also react strongly to typefaces used in movies. In the movie “Avatar”, the font Papirus was used in subtitles for the Na’vi language of the alien people, probably because it has a charming, rough, hand-made look, but the movie-makers didn’t think about the reactions of font-familiar viewers. Instead of enhancing the experience, it distracted viewers from the story: young people felt cheated because they recognized Papirus as an Earthly typeface bundled with millions of personal computers. My students made complaints like, “This is not a new font from outer space! We’ve already seen it in Mac OSX!” It caused a lot of comments on the web, most of them negative.

Y: Haha, interesting.

C: What’s interesting to me is not the specific opinions, which on the Internet are often either love or

hate, but that so many people voiced their opinions. Here, I should make an appeal, as some of the commenters on the web also did, for the type design profession. James Cameron spent \$300 million dollars making "Avatar". He even hired a linguist to invent the spoken language of the alien people, the Na'vi. So, instead of using a common font found on millions of computers, he should have commissioned a young, or old, type designer to create a totally new, unique typeface for those subtitles.

Speaking of subtitles, I enjoy watching Chinese movies but have to read English subtitles to understand the dialogue. I understand that in China, movies are also subtitled, so speakers of different dialects of Chinese can understand what is being spoken. In the American release of "Crouching Tiger, Hidden Dragon", the English subtitles used Lucida Sans Italic. An exciting HK-crime movie was "Infernal Affairs". The Chinese poster is typographically intriguing because the design of the characters is like a maze and suggests the complexity of the story, and the title means "Endless Path"; a nice integration of visual form and symbolic meaning. Because "dou" is "tao", the movie is a Buddhist and Taoist lesson :-).

Y: Yes. And at the same time, movies promote typefaces too. I really love the 2007 film "Helvetica" directed by Gary Hustwit. Maybe sometime in the future we can make a film about Lucida.

C: Yes, Helvetica is a good movie that reveals a lot about why people like type. And since you mention the idea, I should say that a movie is now being made about Kris Holmes and her work. It will include Lucida, and other things. So, we can hope that will someday be shown on the big screen, too.

Ah. Another example of types and personalities is in the presidential elections of 2008 and 2012; both sides cared very much about the typefaces they used.

Y: In 2008, Obama used Gotham, which is also used in Batman.

C: Yes, Gotham is an urban sans-serif, while Optima, a more delicate semi-sans-serif was used by the 2008 John McCain campaign. The public analyzes the typeface to tell the personality of the candidates. The International Herald Tribune praised the Obama choice for its "potent, if unspoken, combination of contemporary sophistication with nostalgia for America's past and a sense of duty." Wow! In the 2012 election, many of the Romney signs used Trajan, which is a modern revival of lettering used on imperial Roman inscriptions. I wonder if the public got the idea that Romney's ideas were 2,000 years old, or that he wanted to be an Emperor. Certainly, the

visual impression of Trajan is formal and stiff, like Romney.

Y: In the election happening just now in Taiwan, this is also true. The Democratic Progressive Party in Taiwan cares about campaign design very much. I really love all the posters, photos and clips they made. It reflects the novelty and neutrality of Tsai Ing-wen, the candidate. In my view, the Democratic Progressive Party does a much better job than the Nationalist Party of China. But it's a pity that typography does not mean everything. Today, this morning they lost the election. But it's interesting to see that Asians are following closely.

C: Yes. I don't think typefaces can influence elections much, unless the candidates are otherwise indistinguishable and one uses Comic Sans and the other, Times Roman. But speaking of elections in Taiwan, digital typography has made it much easier to develop and use expressive typefaces for Chinese. My student, and your friend, Xuan Zhang and I did a study of the expressiveness of Chinese typefaces, but, alas, we didn't finish the study before he graduated. Nevertheless, I was intrigued to study the wide variations in Chinese type designs available today.

Back to your question about public awareness of typography, there are numerous blogs about typefaces now, and discussion groups like Typophile. People write how they love or hate certain typefaces. There is a site that express how much the blogger hates Comic Sans (Ban Comic Sans). It's amusing and not too serious, more fun than nasty.

Y: The same for Arial too!

C: I haven't seen the anti-Arial sites, but I confess, I disapprove of Arial for ethical reasons. I feel it is a too-close imitation of Helvetica, a nearly identical style with the same width metrics, x-height, capital height, stem weights, and proportions so it can replace Helvetica but be just different enough in little details to not be an obvious rip-off or plagiarism. It was said that Monotype offered Arial to Microsoft much less expensively than what Linotype wanted to license Helvetica, so Arial is a font made for business reasons, not for artistic integrity, and as such, it doesn't advance the art of type design.

Y: Oh, I have another question related to Helvetica. As you know, Apple switched to a Helvetica flavored typeface on the iOS platform and many professional Mac apps as well instead of continuing to use Lucida Grande. What's the motivation behind that?

C: I don't know the answer. I guess that Steve Jobs saw the Helvetica movie and decided he wanted to switch to that instead of Lucida. Did you know that Steve Jobs and Kris Holmes studied with the same

calligraphy teacher, Robert Palladino, a former monk who taught at Reed College in Oregon? But not at the same time. Typefaces have a lifetime. Text faces get adopted slowly, sometimes over decades, and slowly become old-fashioned, also over decades, though some older designs get revived. Lucida took several years to become widely used in the mainstream computer world, but by the 1990s, it had been adopted by Microsoft and Sun Microsystems, and was licensed by Adobe as well. Apple licensed it for System 7, but then postponed release until the Copland system, and when that was never released, Lucida was postponed again until it became the user interface font for Mac OS X.

Y: Then Apple's Mac OS X has been using Lucida for more than ten years.

C: Exactly. Designers of operating systems sometimes change fonts, just as magazine designers do. Helvetica, in my opinion, was not very legible at low resolutions and small sizes on screens. It is too tightly fitted and the letter shapes are too similar. That's one of the reasons we designed Lucida, with more humanistic letter forms and looser spacing for better legibility on screens. But, as screen resolutions increase, and anti-aliasing techniques improve, we see better and better displays (like Apple's Retina displays), so Helvetica's subtleties can now be rendered more clearly.

Y: And what's your opinion on that?

C: Well, we designed Lucida in the 1980s to be an alternative to Helvetica, so that reveals my opinion. Though the designs have roughly the same x-height and stem weights, the letter spaces between Lucida letters are proportionally greater than in Helvetica, and the Lucida letters are more differentiated in shape. Thus when rendering on a computer screen you will find it much easier to read Lucida than Helvetica. It's easy to demonstrate — see, now you are reading questions in your Mail application which uses Helvetica, so you have to move your head (and eyes) much closer to the computer screen.

Y: Oh! Yes, I never think about that. And there is no way to switch back to Lucida in the Mail application.

C: If you keep that posture for too long, maybe your neck will hurt. No typeface is perfect for every size and reading distance. Digital technology makes it easy to scale type to any size, but human vision puts different constraints on type designs. Among other factors, there is a visual phenomenon called "crowding", which limits how closely objects can be spaced. When objects such as letters are too close to each other, you have trouble recognizing them unless you bring the text closer to your eyes, thus making the

images, and their spaces, bigger on the retinas of your eyes. In a practical way, type designers have known this for five hundred years and have adjusted smaller fonts to have more space between letters. If you enlarge a photo of a 6 point font cut in 1550 to the same size as a 12 point font cut at the same time by the same punch-cutter, the 6 point font will be wider and more widely spaced. So, types intended for small sizes on screens should be spaced more widely than types intended for large sizes.

Y: Similar question. You and Kris are also two of the authors of Monaco. Monaco was originally the main console font for Terminal application, as well as the font to display code in their Xcode development tool. But they are migrating away from Monaco.

C: Yes, Actually Monaco is one of the Apple city fonts that were originally bitmaps in the first Macintosh. The bitmap fonts with "city" names were created by Susan Kare, an artist and graphic designer who created many of the interface elements for the Apple Macintosh in the 1980s. Later, she left Apple.

Y: Yes. She came to NeXT with Steve Jobs and served as Creative Director at NeXT.

C: OK, and she still designs icons and other digital images. In 1989, Apple asked Kris and me to make new versions of the bitmap city fonts — New York, Monaco, Geneva, and Chicago — to vectorized form in the TrueType font format that Apple invented. So we did. The new fonts began with Susan Kare's designs, but it was impossible to make them exactly the same in vector format, so we had to change several features and proportions. In Monaco, we had legibility in mind, along with the need to differentiate certain letters for better recognition by programmers and technicians. Characters are distinct, and it is difficult to confuse 0 (figure zero) and O (uppercase O), or 1 (figure one), | (vertical bar), I (uppercase i), and l (lowercase l). We tried to maintain the hint of cursive that was seen in the original 'a' in the bitmap Monaco, but we innovated several other features. We wrote a short paper about the project [3].

Y: But now Apple is switching from Monaco to Menlo. What's the main reason behind this move?

C: I don't know, but I guess that one reason might be that Menlo has a full set of italic and bold weights, whereas Monaco has only roman. Years ago, we offered to expand the Monaco family with bold weights and italics, but Apple never chose to do so. I assume Menlo was named after the Menlo Park city in California. Menlo is also free and open source. It's a revision of open source Bitstream Vera and the open source Deja Vu font family based on Vera. Thus I'm

not astonished to see Apple adopt it. It's free and they can modify it as they wish.

Y: And I guess because Menlo came from an open source font, most Linux or other open source operating system programmers are familiar with it. So it makes them happy to switch to develop applications for Mac.

C: Yes, that sounds reasonable.

Y: You just mentioned screen resolution got much higher over these years. Do you think more people will switch to screen reading?

C: Yes. In 2009, less than 3 percent of publishers' book sales were e-books, but today, around 20 percent are e-books. Based on the current adoption rate, I guess that screen publications will outnumber paper ones in 10 years or sooner, including books, magazines, and newspapers. Of course, prediction is difficult, especially about the future. Maybe it will be sooner, and maybe later. In the 1980s, screen resolution was not high enough to render type well, so most people still read newspapers. However, by the 1990s, computer screens got better and better. While for older people, newspapers are still their main reading media by habit and preference, many young people spend more time reading computer screens than reading print newspapers. Now, on this graph, you can see that the readership trend line for print newspapers is dropping quickly as screen resolutions increase. Now that we have very high resolution displays, for example Apple's Retina displays, I expect that in the future more people will read from screens than paper. The trend is accelerating with the iPads from Apple, the Android tablets from various firms, and the Kindles from Amazon. For 500 years, printing on paper was the dominant information technology for Europe and most of the world. Now, digital media are the information technology of the 21st century, but for humans to receive the information, it must be read, and reading requires typefaces and fonts. Happily, most fonts have made the leap from analog to digital. Not all of them work as well in digital, but that provides opportunities for designers to create new typefaces, and to revive and revise older ones for new technology. Print might be dying, but typography is living better than ever.

Y: In the era of LCD screens at max 500 DPI, must the design of a font be influenced by the screen resolution, which is still low compared with the 600 DPI of a common laser desk printer, so that it can be used also in a tablet/ebook reader without many problems?

C: Good question. With high resolution screens, finer details of a typeface can be shown. Hinting

has already been abandoned on Apple's Retina displays. Steve Jobs claimed the Retina display has so high a resolution that the human eye cannot see single pixels any more. This is not always true, according to some scientific study, but yes, you can expect that screens and prints will look much more similar these days.

Y: So there is a trend of convergence between digital and print type, as the quality of display improves?

C: Yes. I think so. But there's still something one should care about—for display fonts, a larger x-height and more letterspacing will make the font easier to read on a display. Also, as I have mentioned, the same glyph looks thinner on display than on prints so a font with a little bit of darkness will be better. You should care about these things when designing a typeface.

Y: Even without background illumination, most E-ink readers also choose dark fonts.

C: Yes. This is because the screen's resolution is still not high enough. And moreover there are two other reasons to make the problem worse. First, E-ink has a far coarser gray scale than modern computer screens and doesn't have RGB subpixels. Thus, subpixel rendering doesn't work on those devices, and normal font anti-aliasing works more poorly than on a computer screen. Also, the background color of an E-ink display is already gray. A darker font is more legible.

Y: You just said people will read more and more on a screen. I have a related question. What do you think of the recent hype over web fonts?

C: Oh, yes this is a good thing: web fonts enable readers to read web pages just as books—previously only a limited collection was available for designers to use.

Y: Yes, with the release of the WOFF (Web Open Font Format) specification as an open standard, more and more browsers support it. So designers can use whatever font they like.

C: And moreover, hobbyists are able to create their own fonts and release them to the public to get wider adoption, while previously all of these can only be done by professionals. Web fonts lower the barrier of typographic design and you will see more typefaces appear in the near future.

Y: Then what about the downsides?

C: Well, a web font is much easier to get pirated, and it's harder for type designers to make money from it. That's why TypeKit or similar businesses were born. We should explore this market more to find a reasonable business model for the type designers to make a living.

7 Ancient type digitizations

Y: You said that nowadays hobbyists create their own typefaces, and release them to the public. Last month, font designers in China had a heated argument on one font created by a hobbyist. His name is Digidea. He bought a Kangxi Dictionary which was the standard Chinese dictionary during the 18th and 19th centuries. The dictionary contains 47,037 characters including obscure, variant, rare, and archaic characters. Then he scanned all the characters into the computer, and use auto-tracing tools to trace the outlines of those characters. Finally he released the font called KangXiZiDianTi (see fig. 29).

C: This looks amazing!

Y: Yes, but when you scale the font, you see problems. And even if you don't scale the font large enough, you will see uneven thickness among glyphs, even in a single stroke as well.

C: Ah. I see that now. But first of all I should say, if he did not use auto-tracing, this font wouldn't be possible.

Y: Right. It's a huge amount of work — manual font creation would take one person years to do fifty thousand characters.

C: But in most situations you have to. This typeface is lucky, because it's a reproduction of a typeface in a dictionary, where almost all glyphs are presented. The "reviver" of the typeface is lucky to have such a wealth of characters to start with. But in most cases, type designers are not lucky. What if we want a typeface in Xizhi Wang's style? Or Mengfu Zhao's?

Y: We should ask someone who is really good at those styles to write them, or at least we should ask experts to analyze these styles and figure out the underlying logic of the handwriting to provide guidance for the type designers to make glyphs according to these rules.

C: Yes. This is a wonderful challenge involving art, practice, and logic. You write the Slender Gold style devised by Emperor Huizong of Sung. Perhaps you could write a large set of characters at a big size and scan those! And being a computer scientist, you could think about the logic of artistically combining the strokes to make new characters that aren't in any extant examples of Slender Gold. A good hobby for a computer scientist! There was a calligrapher in Japan, Yanagida Taiun, who studied Xizhi Wang's style very well. He practiced Xizhi Wang's Lantingji Xu so well that he could create very good copies, at least, I am not able to distinguish them. I have read that none of Xizhi Wang's original writing survives, only copies, so modern calligraphers make copies of copies. In western calligraphy, there

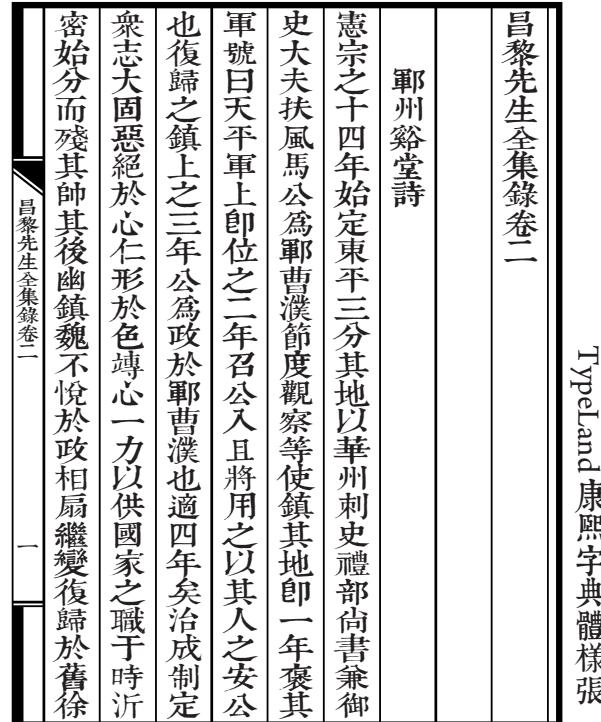


Figure 29: The KangXiZiDianTi font, (re)created by Digidea Lee, 09-26-2010, <http://blog.typeLand.com>.

are similar traditions. Many 20th century calligraphers learned the chancery cursive style from a book by Ludovico degli Arrighi, published in 1522. But, the calligraphy was cut in wood blocks, so what some people today are really copying are the wood cuts, not the original handwriting, though some actual samples of Arrighi's handwriting do survive. What Kris Holmes and I and many others learned of chancery cursive was based on modern calligraphers — our teachers — who had reinterpreted how to write chancery from the early printing. We were taught handwriting, not woodblock graphics. For Apple, Kris Holmes designed Apple Chancery, which is an interpretation in digital type of our calligraphy teacher, Lloyd Reynolds, whose handwriting was based on Arrighi's manual and on manuals by English calligraphers who reinterpreted Arrighi.

Y: But in the previous case, if all the glyphs, or perhaps most of the glyphs are available, or those not available can be derived from available parts, do you think ancient type can be made by auto-tracing, or they must be fine tuned by a human?

C: This is a very important question: what is the best way to "revive" a script or typeface from old times? Calligraphers do it by learning to write so their results resemble scripts in surviving old manuscripts. It is like choreography for the hand. You learn a dance of the pen or brush and the traces of

your moves are the graphic image of the script. Kris Holmes studied dance, and that is why her scripts are so lively with implied motion. (Lucida Handwriting, Kolibri, Isadora.) We can understand why some of the old calligraphers especially in China and Japan, saw a mystical aspect of calligraphy, influenced by Taoism or Buddhism. Movement, which involves rhythm, breathing, discipline, relaxation, and so on, creates the graphical forms, which are 2-dimensional intersections of 3-dimensional paths in time, so 4 dimensions total. There is an amazing movie by a mathematician (Thomas Banchoff) showing a 4-dimensional cube, a tesseract, moving through 3-space, and of course projected into 2-space on a screen. At one screening years ago, the audience cheered as they understood what was happening. I think this is analogous to the mystical aspects of calligraphy — those two higher dimensions in 3- and 4-space, that we can infer from the 2-D graphical forms. Of course, there are good calligraphers and type designers who don't believe in the mystical aspects, who care only about the 2-D images, but the higher dimensions can enrich our appreciation.

For typography, the problem of revival is more like signal processing: how to distinguish signal from noise? The hand motions of the punch cutter are not important because the fonts are a kind of shallow sculpture, bas-relief. The engraved contours are the important things. For some typefaces, like some 16th century cuts by Garamond and Granjon, and 18th century cuts by John Handy, who cut type for Baskerville, we can get a very good idea of the signal because their hand-cut steel punches survive, and sometimes their matrices, the impressions that the punches make when driven into a blank of copper. But, reproducing the face of the punch is not a perfect solution for today, because the old punch-cutters compensated a little bit for subsequent processes, especially ink-squash of type on the paper. The problem is much harder for typefaces for which no punches, matrices, or old type survives, like the types of Jenson, Aldus, and Fournier. There are two ways of doing things. First, let's preserve the original printed form as much as possible. This includes some noise along with the signal.

Y: Then you will get very ugly fonts.

C: Yes, I think so, but people who like this approach don't think it's ugly; rather, they say these imperfections preserve the feeling of ancient typography.

Y: Just like metal type typesetting was dead years ago, but now Apple makes letterpress cards for customers. Sometimes old-fashioned things get revived.

C: Exactly. We like the imperfections of the old methods because they have more personality than our modern methods, which often seem to lack soul, despite their advantages. The second approach is to re-interpret the type by trying to understand the intentions of the original type artist, and the limitations of the medium, and then reinterpret those intentions in modern media. A vision scientist who studies reading once told me that he doesn't really care how a typeface is made or printed — what he cares about is the image on the retina of the eye; that is what is communicated to the brain.

Y: Why not make the revival as authentic as the original one?

C: Sometime you can't. Between the old days of early type and digital type now, technology has changed, from wood block to metal type to mechanical type to phototype to digital type, from paper to CRT screens to LCD screens, to e-ink. And aesthetics and taste have also changed. In Europe, from old-style typefaces like Garamond, to modern styles like Bodoni, to sans-serifs like Helvetica. After high resolution digital typography made well-rendered classical designs cheap and easy, young designers in the 1990s rebelled against perfection and used "grunge" types and "distressed" types, full of dirt, errors, jaggies, and other noise.

In the KangXiZiDianTi you mentioned, the earliest printed editions were cut in wood blocks, I assume. Is that correct? First, some calligraphers had to write every character in the dictionary on paper, as models for the wood-cutter. Second, probably many wood-cutters cut the characters in wood blocks for printing. I don't know if individual characters were cut in wood, small individual pieces of wooden type, or a whole page on a single block. I think the latter, a whole block per page. After the calligraphers have died and their original handwritten examples transferred to wood and lost in history without a trace, there's no chance you can find them. Today, you cannot find the earliest wood blocks either. Third, after the wood cutting, there was the printing process — which slightly deforms the glyph shape as the ink is squeezed onto paper. And, after many impressions, the characters on wood become worn and less distinct. The same is true for metal type. The paper that early printers used, the ink they made, all have effects on the image in the final book. There may be more "signal" information lost in these processes. Fourth, the book preserved to this day may not look the same as it was hundreds of years ago. The humidity and temperature of the environment may change the glyph shape as well, not to mention disasters like insects eating the paper.

Fifth, during the photocopy and scanning process conducted today, there might also be other information losses. I believe that scanning at 600 pixels per inch is not enough to capture all the artistic information in text sizes of type. 1200 pixels per inch is much better, but it takes more time, and thus costs more. Even at 1200 ppi, the image is not perfect, because of noise. Then, if you scan with some level of gray depth, later you may have to threshold down to bi-level pixels for fitting curves around contours, although there are also methods of fitting contours to gray-scaled images.

Y: So it's not possible to trace down the original shape any more.

C: Exactly. In the western world we have already known that since the late 19th century, when English typographers began to revive old types by enlarging photographs of old books from the so-called "cradle" of printing, the years before 1501. Well before the digital era, it was recognized that data about the image was being lost in analog restorations. So most designers did a kind of creative renovation instead of trying to just remove noise from enlargements of the original form.

Y: I see.

C: Take the Jenson text as an example. I showed you a page typeset in Jenson earlier (fig. 3). Jenson was the first great creator of the "humanist" roman types that became the model for all subsequent printing in the Latin alphabet, but all his original punches, matrices, and types have been lost. A great pity. So all Jenson reproductions are redesigns from the images in books that Jenson printed from 1470 to 1480. There is a wide variation of weights and shapes of modern types modeled on Jenson.

Bruce Rogers' Centaur, created in 1914–15, is crisp and sharp, designed by reworking photographic enlargements with a pen. It was re-cut by Monotype in 1929 in a range of sizes. Morris Fuller Benton's Cloister of 1913 is darker and sturdier, made by engineering-style drawings based on enlargements, with attention to mechanical letterpress printing of the early 20th century. Robert Slimbach's Adobe Jenson of 1996 is a careful reconstruction for digital typography. Their variations show the visual senses, goals, and artistic or technical limitations facing modern designers.

Y: I know this typeface [Centaur], it was created for the Metropolitan Museum of Art.

C: Yes, that's true, so you know. Centaur has a lighter, sharper quality than Adobe Jenson, perhaps reflecting Rogers' pen-inscribed approach. It is rather light in digital imaging today because it was designed

A: Body sizes the same

Centaur (Rogers 1915) [14 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

Adobe Jenson (Slimbach 1996) [14 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

Breughel (Frutiger 1981) [14 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

B: x-heights the same

Centaur [17.45 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

Adobe Jenson [16.45 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

Breughel [14 point]

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz&1234567890

Figure 30: Comparison of Jenson revivals.

for letterpress printing, which added weight because of ink-squash. Ron Arnholm's Legacy of 1993 is Jenson modernized to late 20th century taste, with a larger x-height than the original. George Abrams' Venetian, circa 1999, is another careful and noteworthy Jenson revival. Also there is Hermann Zapf's Aurelia of 1983, for digital typesetting, which has some of the calligraphic accents of Palatino.

I include among Jenson revivals Adrian Frutiger's most intriguing Breughel design of 1981. (See fig. 30 and also http://odaddyo.com/typography/type_class/FrenchOldstyle.pdf.) It is not concerned with imitating superficial aspects of Jenson's types, but is a deep attempt to render the philosophical spirit of Jenson's era, when handwriting was reduced to sculpture and mechanical reproduction. Breughel was released by Linotype 501 years after the death of Jenson. I think typographers weren't ready for such an innovative design in 1981. Maybe fifteen years later, in the era of punk and grunge typography, Breughel might have become more popular – unusual, a bit irregular, but legible. The Swiss typographer, Bruno Pfaffli, who was Frutiger's studio partner for many years, used it very well in catalogs and posters for French museums.

Y: No two of them look exactly identical.

C: And this is good in some sense—rather than restricting modern type designers to historical details, the new digital reproduction processes give them the freedom to create something new. So such a discussion in China, regarding the Kangxi dictionary characters, is wonderful. I am really happy to hear of such a discussion if it turns out to generate new thoughts and ideas.

Y: Thank you very much, Prof. Bigelow, for taking the time to do this interview. I have learned a number of things I didn't know. And many thanks also for your great contribution in digital type design and research, especially the work of the Lucida typeface family.

C: Because of its high legibility, I'm happy to see Lucida on computer platforms like Apple's Mac OSX, Sun's Java platform, Bell Labs' Plan 9 and Microsoft Windows. We're working on a web site for B&H at <http://www.lucidafonts.com> which will have (even) more information. For now, let's close by mentioning a very different example of Lucida in use—in a Colorado corn field in 2002 on the Fritzler farm; the corn plants are used as “pixels” to render Lucida Handwriting: <http://www.fritzlermaze.com/mazes.html>.

Acknowledgments

Many thanks to Prof. Charles Bigelow for doing this interview. Many parts of this talk have Prof. Kris Holmes' contributions as well. Without their help this interview would not be possible. Thanks to Xuan Zhang for introducing me to Prof. Bigelow. Jiang Liu provided the funding and invested a lot of resources to edit and publish this interview. A few questions in this interview were collected from Jiang Jiang, Rex Chen, Karl Berry, Luigi Scarso, and Shiang-Ing Ji.

References

- [1] Huda Smitshuijzen AbiFares, *Arabic Typography*, Saagi Books, London, 2001.
- [2] Charles Bigelow and Kris Holmes, “The design of a Unicode font”, *Electronic Publishing*, 6:3 (1993), 289–305. <http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume6/issue3/bigelow.pdf>
- [3] Charles Bigelow and Kris Holmes, “Notes on Apple 4 Fonts”, *Electronic Publishing*, 4:3 (1991), 171–181. <http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume4/issue3/ep050cb.pdf>
- [4] Hans Hagen, Taco Hoekwater, Volker RW Schaa, “Reshaping Euler: A collaboration with Hermann Zapf”, *TUGboat*, 29:2 (2008), 283–287. <http://tug.org/TUGboat/tb29-2/tb92hagen-euler.pdf>
- [5] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, Sebastopol, CA, 2007.
- [6] Gordon E. Legge and Charles A. Bigelow, “Does print size matter for reading? A review of findings from vision science and typography”, *J. Vis.*, 11:5 (2011) 8. <http://www.journalofvision.org/cgi/content/abstract/11/5/8>
- [7] Robert A. Morris, Kathy Aquilante, Charles Bigelow, and Dean Yager, “Serifs slow RSVP reading at very small sizes, but don't matter at larger sizes”, Submission to Symposium session “Legibility and Usability Issues for Text Displays”. SID Symposium Digest of Technical Papers, 33:1, pp. 244–247, May 2002. <http://www.cs.umb.edu/~ram/rsvp/publications/SerifsSubmittedV2.doc>
- [8] Vaughan Pratt, “Techniques for Conic Splines”, ACM SIGGRAPH Computer Graphics, 19:3, July 1985, pp. 151–160. <http://dl.acm.org/citation.cfm?id=325225>
- [9] Emil Ruder, *Typographie*, Verlag Arthur Niggli, Sulgen, Switzerland, 1967. <http://www.designers-books.com/typography-emil-ruder-1967/>

◇ Yue Wang
yuleopen (at) gmail dot com

Oh, oh, zero!

Charles Bigelow

Abstract

Despite exponential increases of computing power over the past half-century, at least one problem involving ones and zeroes has defied easy solution: how to shape the graphical forms of numeral ‘0’ (zero) and capital letter ‘O’ (Oh) so a human reader can easily distinguish between them.

1 Introduction

What follows is a look at three aspects of the zero-Oh problem.¹ First, a survey of computing and typographic literature discussing the problem in the 1960s and 1970s. Second, examples of practical solutions in digital fonts from the 1980s to present. Third, examples of the origins of the problem in the typography of the Italian and French Renaissance, and in English and American typography during the Industrial Revolution. The focus is on typographic symbols. For histories of mathematical notation before typography, see Cajori (1993) and Ifrah (1998).

2 Zero versus Oh in computing

R.W. Bemer (1967), in a playfully entitled paper, “Toward Standards for Handwritten Zero and Oh: Much Ado about Nothing (and a Letter), or A Partial Dossier on Distinguishing Between Handwritten Zero and Oh” presents a compilation and discussion of proposals made between 1958 and 1966 to disambiguate the handwritten forms of zero and Oh. The goal of the study was to enable more accurate reading of handwritten code and data by the keypunch operators who typed punched cards for computer input.

It is doubtful that Bemer’s paper led to lasting changes in handwriting, but Bemer also helped develop the American Standard Code for Information Interchange (ASCII), which, along with the advent of direct keyboard input, shifted the zero-Oh problem

¹ In this paper, the first reference to a character frames it in quotes, followed by the character’s common name in parentheses. Subsequent references use the common name or a disambiguating term. Examples: ‘O’ (Oh) for capital letter Oh; ‘0’ (zero) for numeral zero; ‘1’ (one) for numeral one; ‘I’ (ell) for lowercase letter ell; ‘I’ (capital I) for capital letter I; ‘Ø’ (zero-slash) for slashed zero; ‘Ø’ (zero-dot) for dotted zero. Characters representing numbers, e.g. 0 1 2 3 4 5 6 7 8 9, are here called “numerals”, although typographic literature typically uses the term “figure” for a character representing a numeral, e.g. “old-style figures”, “lining figures” (Bringhurst, 1996). The Unicode character standard uses the term “digit” (Unicode, 2007). The Unicode standard distinguishes “character” as a unit of a writing system from “glyph” as a graphical mark representing a character, but that distinction is not fastidiously maintained in this paper.

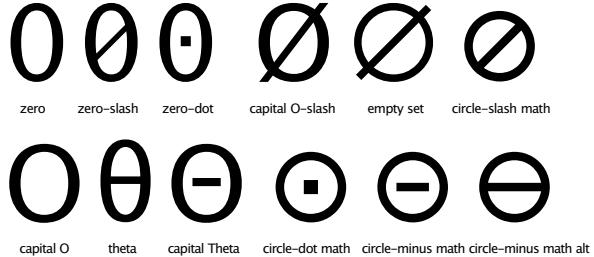


Figure 1: Zero and Oh: look-alike characters.

from handwriting reform, a fraught topic, to typographic legibility, which is no less problematic but substitutes mass-produced, prefabricated letters for the wayward penmanship of programmers. Not that this solved the problem of ambiguity in displayed or printed zeroes and Ohs, as exemplified by DIN 1450, the most recent legibility and typography standard from the German Institute for Standardization (DIN, 2013), which once again revisits the perennial problem of differentiating zero from Oh in contemporary typography.

Advances in font technology have complicated the problem by enabling fonts to contain much larger character sets, increasing the chances that several confusable letters and symbols may appear in a font or font family, especially in scientific and mathematical publishing. Figure 1 shows a set of characters similar to zero and capital Oh, from Lucida Sans and Lucida Math fonts.

The zero-Oh solutions proposed in Bemer (1967) include: a loop, flourish or stroke at the top of Oh; a slash through zero or Oh; a dot or dash in the center of Oh or zero; a rectangular shape for Oh but an elliptical shape for zero (or vice-versa); an Oh wider than zero; a lozenge orientation of Oh but square orientation of zero; a horizontal bar over Oh. One entry in the dossier briefly addresses the problem of differentiating numeral ‘1’ (one) from capital letter ‘I’ (I), and numeral ‘2’ (two) from capital letter ‘Z’ (Zee or Zed).

A subsequent tentative agreement on handwritten letter and numeral forms for computing was published in 1969 by an ANSI (American National Standards Institute) working group as “Proposed American National Standard: Presentation of Alphabetic Characters for Information Processing” (Kerpelman, 1969). The proposal recommends a handwritten loop at the top of the capital Oh to distinguish it from a plain oval zero (Figure 2).

Kerpelman makes an intriguing observation regarding an evident difference in preference between two groups of programmers: “Programmers accustomed to use of business-type languages seemed to favor marking the zero. Those using mathematical



Figure 2: Zero and Oh from (Kerpelman, 1969).

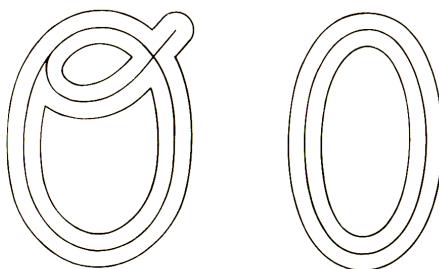


Figure 3: Modification to Oh proposed by Vartabedian (1969, 1970).

or scientific-type languages conversely favored marking the letter.” (Kerpelman, 1969). This preference difference, which can be characterized as between humanists and engineers, is a recurrent theme in the zero-Oh annals.

After these ACM and ANSI publications, the zero-Oh debate moved to a typographic forum, the Journal of Typographic Research, where psychologist Dirk Wendt (1969) analyzes the problem of discrimination and confusion between different forms of zero and Oh but does not recommend a single solution, other than an observation that zero narrower than Oh is more often interpreted correctly. In the same journal issue, a Bell Laboratories researcher, Allen G. Vartabedian, reports the results of a different legibility study and proposes that a loop or stroke be added to the top of the Oh to distinguish it from zero (Vartabedian, 1969). The proposal to add a loop to Oh is similar to that of Kerpelman.

In a letter in a later issue of the same journal, calligrapher and type designer Hermann Zapf (1970) objects to Vartabedian’s proposal and proposes a contrary modification—the addition of a short horizontal stroke to the top right of the zero. Vartabedian (1970) responds with additional argument in favor of modifying the Oh. An engineer (Vartabedian) favors altering the Oh, while a humanist (Zapf) favors altering the zero, as shown in Figures 3 and 4.



Figure 4: Proposed modification to zero by Hermann Zapf. The zero resembles a tall lowercase sigma.

2.1 Forms and ideas

Plato (or the character Socrates in Dialogues written by Plato) discusses how letters express ideas. He suggests, for example, that the letter omicron expresses roundness, though it is not clear whether Socrates (or Plato) is referring to the round shape of the letter or to the round shape of the lips when pronouncing the vowel signified by omicron. Perhaps both. Ancient Greek mathematics did not use a written symbol for the concept of nothing, but the atomist philosopher Democritus, possibly a contemporary of Socrates, uses the word “void” in contrast to “full”, as attested by Aristotle in his Metaphysics (1989).

In modern semiotic discourse, the question can be asked: Is the graphical symbol “iconic”? Does the glyph resemble the thing it signifies? For most typographic glyphs, the answer is “no”, but the zero glyph, a late addition to Latin script, is an elliptical or circular ring; its vacant interior containing nothing. Hence, it appears to be iconic. In writing and typography, an empty space separates symbols or groups of symbols, but does not signify something. Hence, to denote “nothing” there must be a mark that in some way delineates the presence of nothing. Yet, if the empty interior of the zero is iconic of nothing, then a mark inserted into it indicates that something is in the void, thus contradicting the iconicity of the empty glyph. In its long history, zero has sometimes been represented by a dot rather than a ring, so it could be argued that a zero-dot glyph is a double nothing, like a double negative is emphatically negative.

In set theory, an iconic representation of the empty set is a pair of braces framing an empty space: { }. The zero glyph has also been used to denote the empty set, but to disambiguate the number zero from empty set, glyphs made from zero with a slash (‘Ø’) or a circle with a slash (‘∅’) have been adopted as symbols for the empty set. In Unicode, the empty



Figure 5: No Smoking symbols.

set character has code point hexadecimal 2205; Unicode does not separately encode zero-slash and circle-slash, instead considering them to be different visual forms. Both forms can still be provided as alternate glyphs within one font, as in the Lucida Math OpenType shown here. (To confound further, there is a slashed-zero variant appearance for zero itself which—though seldom used in serifed fonts—often appears in sans-serif monospaced fonts (Figure 9), as we will discuss later.)

The empty set forms do not begin to exhaust the slashed circle symbols. A circle with a slash not projecting beyond the ring ('∅') has been adopted in European (and some American) signage to signify prohibition—"no" or "not". The prohibition slashed circle is usually, but not always, in an orientation opposite to that of the empty set, with the prohibition slash running from northwest to southeast but the empty set slash running northeast to southwest. Unlike the empty set symbol, the prohibition symbol usually contains something to be negated, such as a cigarette, as in Figure 5, which shows both orientations. The prohibition symbol is at code point hexadecimal 20E0 in Unicode.

Still more: the mathematical operator "circled division slash" (\oslash) is oriented like the empty set but the slash does not protrude beyond the rim of the circle; it has code point hexadecimal 2298. And the programming language APL's "circle-backslash" character ('Q') is encoded at hexadecimal 2349; it has various possible forms combining circle and backslash.

And, though not strictly circular, let us not forget the character Oh-slash 'Ø' (O with stroke, code point 00D8), and its lowercase form oh-slash 'ø' (00F8), a common letter in the orthographies of the Scandinavian languages Danish, Norwegian, Faroese, and Sami.

Let's turn back to the common zero and Oh. In several recent fonts, excepting OCR-A and OCR-B from the 1960s and later fonts imitating them, the zero gets marked instead of the Oh. Hermann Zapf, however, who originally proposed to modify the zero with an additional stroke, found a calligraphic way to retain the purity of the empty, unadorned zero when he designed the Euler fonts for Donald Knuth and the American Mathematical Society in the early 1980s



Figure 6: Zero, Oh, oh, Theta, theta from the Euler typeface by Hermann Zapf.

(Figure 6). In the Euler roman typefaces (1987), Zapf drew the zero as a narrow elliptical shape with a calligraphic point at the top and a rounded base, almost as if it had been written with a pen in a single curved stroke. In contrast, the Euler Oh has a wider, smoother, almost super-elliptical shape. Hence, in the Euler typefaces, Zapf found a middle path for both engineers and humanists: neither zero nor Oh are marked by slashes, bars, dots, dashes, or gaps. Oh-like forms with interior marks represent traditional Greek letters, capital and lowercase theta (Θ at hexadecimal 0398 and θ at 03B8).

2.2 Patterns of marking and legibility

Upon first impression, the varied proposals by mathematicians, engineers, psychologists, and designers seem to be in free variation. Some propose to modify the zero, others to modify the capital Oh; some want to add a diagonal slash, others to add a loop, others to add a dot, or a horizontal dash, or a projection. Some propose to reshape the curves of the zero, others to reshape the Oh, and at least one (Lo, 1967) suggests characters from another writing system, Chinese. Despite such variety, a few patterns can be discerned. One is that most of the proposals call for adding marks to existing forms, but none propose deleting parts of existing forms. Strokes and dots are to be added, but not gaps or breaks in contours.

The addition of black marks is in keeping with the common view of type forms, that the black marks are what are important, while the white spaces are not significant. Type designers, typographers, and graphic designers would say otherwise, but they are a small set of professionals, not the vast majority of readers. Another pattern is that the proposed marks are usually located at or above the midpoint of the character, and more often in the right upper sector than in the left. This follows a general tendency for Latin typographic alphabets to cluster most distinguishing features above the mid-point of the lowercase letter, near the x-line, and more often in the right upper sector than the left, a tendency noted by Huey (1908) and Legros and Grant (1916).

**A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 Ä Ö Ü**

Figure 7: FE-Schrift, designed by Karlgeorg Hoefer (with later modifications by others) for German vehicle registration plates. The gap in rectangular zero helps distinguish it from capital Oh; the serif arrangement of capital I helps distinguish it from numeral 1.

There is, however, at least one instance of an open gap to mark zero: on modern German vehicle registration plates, which use a font called FE-Schrift, originally designed by calligrapher and type designer Karlgeorg Hoefer (Figure 7). The zero is semi-rectangular and has a gap in the upper right corner, making it recognizably different from the Oh, which is an egg-shaped oval with unbroken contour. The numeral 1 is differentiated from the capital I in the size, orientation, and arrangement of serifs.

Zero vs. Oh confusion was uncommon before the computer era in part because letter versus numeral ambiguity was resolvable by context. A round, open form amidst numbers was presumably a zero. A round, open form among capital letters, or beginning a sentence or a proper noun, was presumably a capital Oh. In computing, however, symbol strings often mix alphabetic and numeric characters, thus rendering context insufficient as a means of distinguishing similar characters.

Among the proposals in Bemer (1967) is disambiguation of numeral one from capital I, though not from lowercase letter 'l' (ell). Few of the character sets used in computing in the 1950s and early 1960s included lowercase (American Standards Association, 1963), so there were fewer opportunities for confusion between numeral one and lowercase letter ell compared to those between zero and Oh. The numeral one vs. letter ell became more problematic when the 1966 revision of the ASCII character set added lowercase.

The graphical forms of numeral one and lowercase ell had been differentiated in traditional typography but were merged on typewriters, where a single glyph and key was used for both graphemes. Character encoding standards distinguished those characters numerically, e.g. in ASCII the numeral one is decimal 49 and lowercase ell is decimal 108, or in Unicode hexadecimal, they are 31 and 6C respectively, but as visual designs in fixed-width fonts, they have often remained similar in appearance.

1 I I B 0 0 D 5 S Z 2
Futura (1927)

1 I I B 0 0 D 5 S Z 2
Helvetica (1957)

1 I I B 0 0 D 5 S Z 2
Frutiger (1976)

1 I I B 0 0 D 5 S Z 2
Lucida Sans (1985)

1 I I B 0 0 D 5 S Z 2 I 0 0
Lucida Grande & alternates (2001)

1 I I B 0 0 D 5 S Z 2 0 I
Frutiger Neue 1450 & alternates (2013)

1 I I B 0 0 D 5 S Z 2 I I 0 0
Lucida Grande 1450 & alternates (2013)

Figure 8: Sans-serif typefaces showing: numeral one, capital I, ell; B, zero, Oh; D, 5, S; Z, 2. All types set at same body size.

3 Zeroes and Ohs in contemporary fonts

Given the history of interest in the zero-Oh problem, and its transference to the realm of type design instead of handwriting, what solutions are found in contemporary typefaces? Many thousands of fonts are available today, but a small selection of widely used fonts can show the main features of the problems: see Figure 8.

In sans-serif typefaces, the problem of confusion between numeral one, capital I, and lowercase ell is more difficult than in serifed faces because serifs function to distinguish capital I from lowercase ell and both from numeral one. (A serifed capital I has four serifs, a lowercase ell three serifs with the upper left shaped differently than that of the capital I, and the numeral one has three serifs with the upper left serif distinguished in shape from that of both the I and ell.) In many sans-serif typefaces, capitals and lowercase ascenders are the same height, removing another distinguishing feature.

In the geometric sans-serif Futura by Paul Renner (1927), the zero is a narrow vertical ellipse and the capital Oh is wider and visually circular. The numeral one has a short horizontal stroke at upper left, and is the same height as capital I, while lowercase ell is noticeably taller than capital I or numeral one.

In the neo-grotesque sans-serif typeface Helvetica by Max Miedinger and Eduard Hoffman (1957), the zero is distinctly narrower and slightly shorter than the capital Oh. The numeral one is distinguished from capital I and lowercase ell by a ramp-like stroke at upper left. The lowercase ell and capital I are the same height and differentiated only by a slightly greater weight of the capital I—a difference that is nearly or entirely imperceptible at small sizes and low digital resolutions.

In the Transitional-style sans-serif Frutiger by Adrian Frutiger (1976), the zero is noticeably narrower than the capital Oh, but both are the same height. The lowercase ell is slightly taller than the capital I, and the numeral one is differentiated from both capital I and ell by a short diagonal stroke at upper left.

In the humanist sans-serif Lucida (1985) by Charles Bigelow and Kris Holmes, the zero and Oh are differentiated by width but not height, reflecting the study by Wendt (1969). The lowercase ell is noticeably taller than the capital I (except at very small sizes and low resolution) and the numeral one is differentiated from both by the short diagonal stroke at upper left.

In Verdana by Matthew Carter (1996), the zero and Oh are differentiated by width but not height; baseline serifs are added to the numeral one and four serifs to the capital I for greater differentiation of the three characters.

In Lucida Grande, based on Lucida Sans, the numeral one is reworked with baseline serifs but the default zero and Oh are the same as in the original version. The Lucida Grande font also includes both slashed and dotted versions of zero, as well as a serifed variant of capital I, but these are not the default forms.

In Neue Frutiger 1450, by Adrian Frutiger and Akira Kobayashi (2013), the capital I acquires four serifs, the zero a dot, and the lowercase ell a curved exit similar to that of lowercase ‘t’. An open zero and rectangular lowercase l are provided as alternates.

In Lucida Grande 1450 by Bigelow & Holmes (2013), the slash zero and serifed capital I are defaults, along with a lowercase ell with exit stroke. The numeral one has baseline serifs as in standard Lucida Grande. A dotted zero and open zero, as well

as rectangular lowercase ell and rectangular capital I are provided as alternates.

From this look at recent and widely used fonts, we can see that from the many proposals for character disambiguation made over the past 50 years, a few trends have emerged and converged. In the zero vs. Oh pair, the zero is almost always the character that receives an added element, usually an internal diagonal slash or an internal dot. The Oh does not get decorated with loops or twiddles, despite such suggestions in Bemer (1967) and Vartabedian (1969). In proportionally spaced fonts, the zero form is generally a narrow ellipse, while the capital Oh is usually a broader, nearly circular form, reflecting the findings of Wendt (1969). In terms of the debate between humanists and engineers, the ‘humanist’ side has won—the numeral gets modified, not the letter.

In the case of numeral one versus capital I and lowercase ell, the outcome is more like a draw. The numeral one is given baseline serifs in several sans-serif typefaces, while the capital I is given serifs in others, and in some sans-serif fonts, serifed versions of both characters occur. The lowercase ell is variable, sometimes differentiated from capital I and one by a serif or a stroke in upper left or lower right, or both.

These trends are seen to an even greater degree in monospaced fonts (Figure 9), which have the added constraint that zero cannot be differentiated from Oh by width because all characters in a monospaced font must of course have the same advance width. Although typewriters as machines have become obsolete, monospaced fonts developed for typewriters, despite their retro appearance and association with old technology, are flourishing in the digital era. Several new monospaced font families have been designed since the widespread adoption of digital font technology for laser printing and computer displays in the 1980s. Just as the forms of letters made the leap from handwriting to print in the 15th century, they have made the leap from analog to digital technology in the 20th and 21st centuries. Technology has influenced the forms of letters over the centuries, but the visual forms themselves can exist beyond any particular technology, reminding us of Plato’s philosophy of eternal forms.

Courier by Howard Kettler (1955) is a serifed monospaced font for IBM typewriters. It became the most used typewriter font of all time and was therefore one of the first to be implemented in digital form. A serifed font, Courier has two baseline serifs on numeral one, three serifs on lowercase ell, and four serifs on capital I, all as would be expected of a serifed typeface. Numeral zero is differentiated from capital Oh by height, because in Courier, unusually,

1I1B00D5SZ2
Courier (1956)

1I1B00D5SZ2
Letter Gothic (1962)

1I1B00D5SZ2
Lucida Sans Typewriter (1986)

1I1B00D5SZ2
Monaco (1991)

1I1B00D5SZ2 0
Lucida Console (1993) & alternate (2013)

1I1B00D5SZ2
Andale Mono (1997)

1I1B00D5SZ2 00ll
Consolas (2006) & alternates

1I1B00D5SZ2
Inconsolata (2009)

1I1B00D5SZ2
Lucida Retro (2013)

Figure 9: Monospaced typefaces showing:
numeral one, capital I, ell; B, zero, Oh; D, 5, S; Z, 2.
All types set at same body size.

the numerals are taller than the capitals and also more loosely spaced; thus zero is a tall, narrow ellipse while capital Oh is shorter and more nearly circular. Capital I is noticeably shorter than either numeral one or lowercase ell.

Letter Gothic by Roger Roberson (1962) is a “fineline” sans-serif for IBM Selectric typewriters. The numerals and capitals are the same height. Numeral one has two base serifs and diagonal upper left stroke, capital I has four serifs, and lowercase ell has a single horizontal serif at upper left. The zero and capital Oh are indistinguishable. (Letter Gothic looks lighter than the other fonts because the original design had a light stroke weight to compensate for ribbon spread in typewriting.)

Lucida Sans Typewriter (1986) by Bigelow & Holmes distinguishes numeral one, capital I, and lowercase ell by position and number of serifs: one upper left serif on ell, two serifs plus upper diagonal stroke on numeral one, and four serifs on capital I. The numeral one is distinguished from the lowercase ell by the shape of the stroke or serif at upper left, diagonal on numeral one and horizontal on ell, by the presence of baseline serifs on numeral one but not on ell, and slightly greater height of ell and other ascending lowercase characters compared to numerals and capitals. Zero is narrower than the capital Oh but has the same height and does not have other distinguishing marking.

Monaco, by Bigelow & Holmes for Apple (1991), derived from bitmap fonts by Susan Kare (1984), has capitals and numerals of equal height but slightly shorter than ascenders. Capital I has serifs, numeral one has baseline serifs as well as the diagonal stroke in the northwest, and lowercase ell has serifs only at upper left and lower right. Zero has a diagonal slash, but is not distinguished from capital Oh by height or width.

Lucida Console (1993) by Bigelow & Holmes has capitals noticeably shorter than the numerals, due to technical constraints in Microsoft Windows NT, for which the font was first developed. Most of the other letters and numerals are similar to those in Lucida Sans Typewriter. A slashed zero was considered, but the designers and Microsoft decided that the height difference would be sufficient to distinguish the two characters. However, in a new version to be released in 2013, B&H have added the slash to the zero.

Andale Mono by Steve Matteson (1997) has serif patterning similar to that of Monaco for the figure one, capital I, and ell. The zero is dotted, rather than slashed, and slightly narrower than capital Oh but the same height. Capitals, numerals, and lowercase ascenders are equal in height.

Consolas by Lucas de Groot (2006) has a slashed zero as default, but as an OpenType font, it includes an alternate dotted zero and open zero. All the zeroes are slightly narrower than capital Oh but the same height. The font also contains old-style numerals including all three zeroes, which align with the lowercase. The numerals and capitals are equal in height and shorter than lowercase ascenders.

Inconsolata by Raph Levien (2009) has a slashed zero narrower than, but the same height as, the capital Oh. The numeral one has the usual diagonal stroke at upper left but lacks baseline serifs, thus being differentiated from lowercase ell, which has three serifs, as in Consolas.

Lucida Retro by Bigelow & Holmes (2013) has a slashed zero narrower than the capital Oh and a two-seriffed lowercase ell as in Andale and Monaco. Proportions and heights are similar to those of Lucida Sans Typewriter, from which it is derived. Additional differences occur in certain other lowercase letters and symbols.

Thus, as we can see, in most digital monospaced sans-serif fonts strict modernist design purity is subordinated to legibility, because many of these fonts are used in programming and terminal and console windows of operating systems and programming environments, where legibility is paramount. Despite the fonts being sans-serif, letters subject to confusion, like capital I and lowercase ell, are given serifs to distinguish them from each other and from numeral one, which may also be given baseline serifs, contrary to sans-serif purism.

If there is a lesson to be drawn from these comparisons, it is that in the computer era, the trend has been toward more marked differentiation of confusable forms, even when the markings are contrary to historical tradition or design purity. It appears that the particular details of individual character designs are not as important as the overall structure of the set of differentiations. Zero may be distinguished from capital Oh and from lowercase oh by slashes, dots, heights, or shapes.

The numeral one, lowercase ell and capital I may be distinguished by the presence, location, and orientation of serifs, but the exact number and location of serifs may vary depending on the preferences of the designers or functions of the font.

4 A brief historical survey

Implicit in Bemer (1967) is an assumption that zero-Oh confusion results from lack of clarity in the handwriting of the 1950s, when computing began to be used widely in industry, government, and academia. The graphical problem long predates computing, however, and may be traced back to the handwriting and early typography of the Italian Renaissance, when our modern alphabets took form and when the Arabic numerals began to be integrated into humanistic and scientific writing and publishing.

Our modern roman and italic typefaces are derived from humanist handwriting, which amalgamated two distinct forms of the Latin alphabet: Roman capitals, which had reached their canonical forms by the 1st century A.D., and Carolingian minuscules, which were based on cursive descendants of the capitals re-formalized by scribes in the court of Charlemagne around the end of the 8th century A.D. Most of the minuscules, which in typography

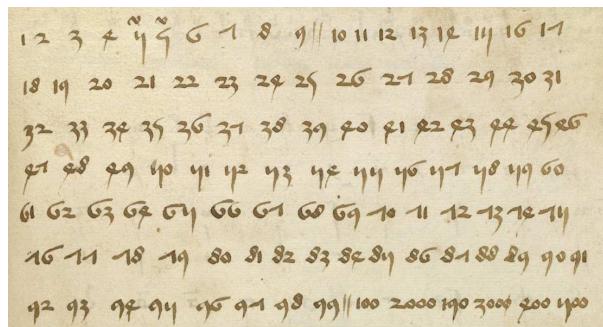


Figure 10: Numerals in gothic handwriting, 1459.

http://commons.wikimedia.org/wiki/File:Ms.Thott.290.2%C2%BA_150v.jpg

are commonly called “lowercase”, evolved into forms different from their capital antecedents, such as ‘a’ from ‘A’, ‘b’ from ‘B’, and ‘e’ from ‘E’, but minuscule ‘o’ retained the form of capital ‘O’. Capital and minuscule Latin letterforms evolved in isolation from the Hindu-Arabic numerals, which were not introduced to Europe until the end of the 10th century A.D. The zero appears not to have been used in Europe to any appreciable extent until early in the 13th century Ifrah, 1998). The influential mathematical book Liber Abaci, written around 1202 by Leonardo of Pisa (known as ‘Fibonacci’), brought Arabic numerals and zero into wider use in bookkeeping and mathematics, but roman numerals, based on letters and letter-like forms, continued in wide usage.

Although the glyphs of writing systems change form over time, the changes generally maintain differentiation between the elements of the system. The centuries of evolution of Latin letters separately from Hindu-Arabic numerals meant that there was no pressure to distinguish the numeral zero from the letters Oh or oh, because they were parts of different systems. From the 9th to the 15th century, Carolingian minuscules gradually morphed into the various gothic hands known as blackletters or broken scripts. As Arabic numerals increased in usage, they were used with gothic handwriting, in which the zero glyph tends to have a slight point where the loop stroke joins itself (Figure 10).

4.1 A sampling of early zeroes in print

In both gothic and humanist manuscripts, Arabic numerals had ascending and descending strokes, what we call “old style” numerals today. The ‘0’, ‘1’ and ‘2’ were roughly x-height; the ‘3’, ‘4’, ‘5’ and ‘9’ descended below the baseline, and the ‘6’ and ‘8’ ascended above the baseline. These features were continued in typographic fonts. The symbol zero in Renaissance handwriting and typography was usually

a circular shape roughly like lowercase oh. Occasionally, the zero was more pointed when used with gothic scripts, and more circular when used with humanist scripts. Following are some specific examples.

1473. A calendar by the German mathematician called “Regiomontanus” (Johann Müller von Königsberg; see Figure 11) was printed in Nuremberg in 1473 using Arabic numerals including zero. An example reproduced by Cajori (1993) is crude, and the forms of the numerals, including zero and oh, are more variable than would be expected from movable type, but the 1474 edition shows a nearly circular zero glyph without strong thick-thin shading, while the oh glyph is larger and darker with more contrast of thick and thin. Also noteworthy is the use of a humanist roman typeface in a book printed in Germany at this relatively early date. Thus, zero was distinguished from oh early in printed books.

The 1476 edition printed by Erhardt Ratdolt in Venice also uses the circular zero in movable type, with a humanist roman typeface. Ratdolt’s edition has the distinction of the first known ornamented title page in a printed book, as well as extensive (though not the first) use of rubrication in print.

1474. The *Fasciculus Temporum*, an encyclopedia of history, by Werner Rollevinck, printed by Arnold ther Hoernen in Cologne in 1474, makes extensive use of Arabic numerals, cut in gothic style, including zero, in dating events in history (Figure 12 shows a sample from a different edition).

1478. An arithmetic text, *Arte dell’Abbaco* (author unknown), written in Venetian dialect and printed in Treviso in 1478 by Gerardus de Las de Flandria or Michele Manzolo shows a roughly circular zero glyph that is apparently the same as the lowercase letter oh glyph. A peculiar twist is that the glyph for numeral one has a dot above it and it appears identical to the lowercase letter ‘i’. Apparently, neither author nor printer deemed it necessary to distinguish those numerals from their similar letters in this humble text. See <http://www.columbia.edu/cu/lweb/eresources/exhibitions/treasures/html/160.html>.

1491. According to Ifrah (1998), the word “zero” first appeared in print in *De Arithmetica Opusculum*, by Philippi Calandri, printed at Florence in 1491. The Arabic name for zero, “sifr” meaning “empty, void”, was borrowed into medieval Latin as “zephirum” in Fibonacci’s work, and later simplified to Italian “zefiro” and then shortened to “zero”). German “Ziffer”, French “chiffre”, and Spanish “cifra”, which include all numerals, come from the same Arabic word, as does English “cipher”, which can mean zero, or more generally a numeral,

		Tunc				Summe Monde	
		Summae				S. 556	
1	A	1 Nov. Tot	20	3	0	11	0 11
2	b	4 noñ der advent s. Staffans	21	4	0	16	0 26
3	c	3 noñ der advent s. Johannis	22	6	1	10	1 2
4	d	noñ der advent der kindlein	23	1	1	21	1 22
5	e	Noñ der advent	24	8	2	6	2 4
6	f	Bridg Obalt	25	9	2	19	2 18
7	g	A d 10	26	11	7	2	7 1
8	A	6 10 S Chort bischove	27	12	3	16	3 14
9	b	6 10 S Iulian und sem gesellē	28	13	3	29	3 28
10	c	4 10 S paul amfied	29	14	4	12	4 11

		IANVARIVS.				SOLIS	LVNAE	Apfendens
		CAPRI				S G	S G	S. m. g.
1	A	Circūfīo domīni	20	3	0	13	0	13
2	b	8 noñ Octaua.s. Stephani	21	8	0	26	0	26
3	c	3 noñ Octaua.s. Ioannis	22	6	1	10	1	9
4	d	2 noñ Octaua.s. Innocentum	23	1	1	23	1	22
5	e	Noñ	24	8	2	6	2	5
6	f	8 id9 Epiphanie domīni	25	9	2	19	2	18
7	g	A id9 Clauis.lxx.	26	11	3	2	3	1
8	A	6 id9 Erbardi episcopi	27	12	3	15	3	15
9	b	5 id9 Iuliani & sociorum eius	28	13	3	29	3	28
10	c	4 id9 Pauli primi eremite	29	14	2	12	2	11

		IANVARIVS.				SOLIS	LVNAE	CAPRI
		S. G.				S. G.	S. G.	S. G.
1	A	Circūfīo domīni	20	3	0	13	0	13
2	b	4 noñ Octaua.s. Stephani	21	4	0	26	0	26
3	c	3 noñ Octaua.s. Ioannis	22	6	1	10	1	9
4	d	2 noñ Octaua.s. Innocentum	23	7	1	23	1	22
5	e	Noñ	24	8	2	6	2	5
6	f	8 id9 Epiphanie domīni	25	9	2	19	2	18
7	g	A id9 Clauis.lxx.	26	11	3	2	3	1
8	A	6 id9 Erbardi episcopi	27	12	3	15	3	15
9	b	5 id9 Iuliani & sociorum eius	28	13	3	29	3	28
10	c	4 id9 Pauli primi eremite	29	14	4	12	4	11

Figure 11: Three different editions of the calendarium of Regiomontanus: (a) 1473 Nuremberg (from Cajori (1993), p. 97); (b) 1474 Nuremberg (<http://daten.digitale-sammlungen.de/0003/bsb00031144/images/index.html?fip=193.174.98.30&id=00031144&seite=7>); (c) 1476 Venice (from the Digital Rare Book Collection at the Vienna University Observatory, <http://www.univie.ac.at/hwastro>).

or a secret (code). See <http://www.metmuseum.org/toah/works-of-art/19.24> and <http://www.lib.umich.edu/apps/bell/map/PTO/GEO/clklg.html>.

1494. In Luca Pacioli’s *Summa de Arithmetica* (*Somma di arithmetica* in Italian) printed by Paganinus de Paganinis in Venice in 1494, the zero is nearly circular without any contrast of thick to thin strokes, whereas in the gothic rotunda text face of the book, the letter oh is taller, more pointed, and compressed. The difference is evident whether the zero appears in tables with other numerals or in linear text with numerals and letters. A high resolution digitized example can be seen on-line from the Max Planck Institute for the History of Science. (It is

Oh, oh, zero!

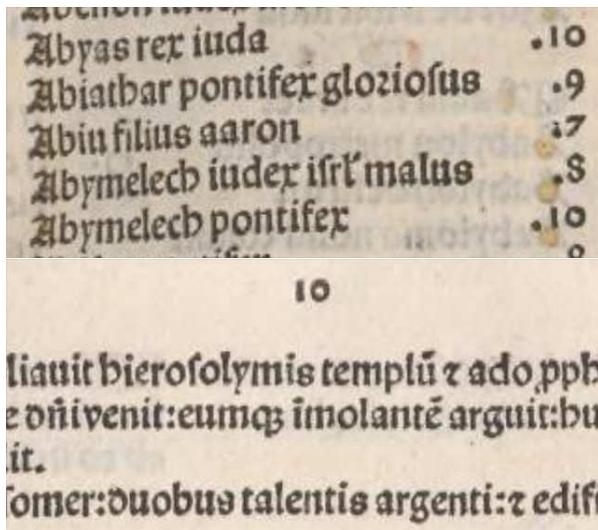


Figure 12: Zeroes and ohs from *Fasciculus Temporum* 1485, Venice: (a) Name index (A's); (b) Folio (10).
<http://archive.org/details/OEXV552P1>

clear enough that an arithmetically minded reader should be able to spot a numero-typographical error in one of the multiplication examples on (digital) page 77, book folio 31.) The printed impressions of the letters and numerals are somewhat variable due to the textured, hand-made paper and imperfections of early printing, but the zeroes in the tables noticeably differ from the lowercase letter oh in the text columns. See http://echo.mpiwg-berlin.mpg.de/ECHOdocuView?url=/permanent/archimedes_repository/large/pacio_summa_504_it_1494/index.meta&start=71&pn=77.

1498. A manuscript of another work on mathematics by Luca Pacioli, *De Divina Proportione* (Figure 13), written in Milan in 1494, is illustrated with drawings of polyhedra attributed to Leonardo da Vinci, but, alas, the scribe who wrote the text in an elegant humanist bookhand remains anonymous. The text uses Arabic numerals but the scribe, despite evident mastery of the edged pen, does not appreciably distinguish zero from oh. This is probably because it is difficult to write an unshaded circular form with an edged pen. Although Arabic numerals are used in the diagrams and calculations, the page numbers are in roman numerals, indicating the conservative power of the older system.

In books printed in humanist typefaces (our “roman” style) before the 16th century, Arabic numerals appear to have been rare in body texts, though appearing in indices, lists, tables, and calculations. Even page numbers or “folii” were usually printed in roman numerals until the 16th century. My impression of the paucity of the Arabic numerals is,

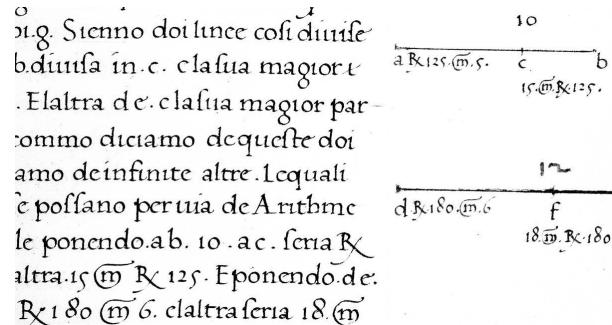


Figure 13: From *De Divina Proportione* by Luca Pacioli, manuscript dated 1498, scribe unknown. Aboca Edizione, Italy, 2010. Facsimile of Manuscript 210, Library of the University of Geneva.

however, based on a fragmentary survey, so further investigation might alter our understanding.

The reason for such rarity of Arabic numerals in humanist texts is unclear, but may have been because the humanists were interested in classical philosophical, literary, and historical works more than mathematical and practical treatises on arithmetic and accounting. Humanist artists and architects did produce treatises on the design of roman capital letters by Euclidean constructions, but these did not extend to minuscules (lowercase) or numerals. It appears that humanists were more interested in Oh than zero. Likewise, in *De Divina Proportione*, Pacioli also constructed the roman capitals, but not minuscules or numerals.

The ring-shaped zero, rather than a calligraphic zero, begins to appear with humanist roman typefaces in the last decade of the 15th century. In humanist works, context was probably sufficient to differentiate numeral zero from letter oh in most instances, but our next example shows one case of possible confusion, perhaps due to the compositor or to the absence of the characters in the roman font. (When setting type by hand, it is easy to confuse look-alike letters, such as ‘p’ and ‘q’ (hence the maxim to mind them) and possibly zero and oh.)

1498. Aldus Manutius in Venice published the *Opera of Angelus Politanus* in 1498 (Figure 14). On a page of “Epigrammatum graecorum”, the Arabic numerals (such as “1490”) use a zero that is the wrong size and alignment for the rest of the numerals. Instead, the zero looks to be the same size and alignment as the oh of the roman text font, Aldus’ first roman, cut by Francesco Griffó. The type is approximately 15 point in size, but the numerals seem somewhat smaller. Possibly, the compositor confused letter oh with numeral zero, or perhaps the set of numerals didn’t include a zero, though that

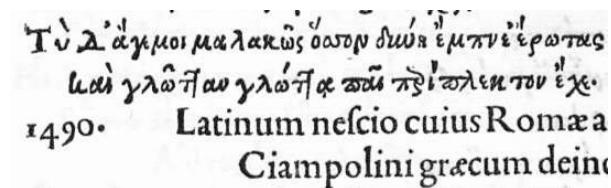


Figure 14: Politianus, *Opera*, printed by Aldus Manutius, 1498. Munich Digitization Center and Digital Library. <http://daten.digitale-sammlungen.de/0005/bsb00050563/images/index.html?fip=193.174.98.30&id=00050563&seite=903>

C ch.3.f.1.10.cōtemp̄o. pro cōtemp̄o. l.20.suspicare.p
urare. p parlare.ch.5.f.1.9. fa parturisce.ch.6.f.1.10.Gratis
ma. seguitoe. p seguirde.ch.6.f.1.14.feruli pro ferali.
Q uāt̄ime.ch.3.f.5.l.2. Ciaro. pro Ciaro. Nō senu
Veneſiis Mense dec̄embris.M.ID.in ædibus Aldi Manuti

Figure 15: Error page of Hypnerotomachia Poliphili (author unknown) printed by Aldus Manutius, Venice, 1499. Rochester Institute of Technology, Cary Collection.

seems less likely, because Griffo, or whoever cut the numerals, should have been able to cut a zero as well as the other numerals.

1499. A year later, Aldus did use a zero properly aligned with other numerals cut at a very small size, on the “errata” page of the Hypnerotomachia Poliphili printed in 1499 (Figure 15).

The Hypnerotomachia is composed in a large (approximately 15 point) humanist roman typeface cut by Griffo. The lowercase is based on the earlier roman, but the capitals are new. Numerals in the main body of the book are roman numerals, but in the “errata” page at the end, small Arabic numerals (approximately 60% of the x-height of the text face) are interposed. These numerals have the ring-shaped zero design that became standard for roman faces in the 16th century. See “20” in line 1 and “10” in line 2.

These examples suggest that Italian Renaissance readers of humanist manuscripts and printed books would have been unlikely to confuse zero with capital Oh; confusion could have occurred between zero and lowercase oh. In most instances, however, numerals and letters occurred in different contexts, which would have lessened the chances of confusion. Where they did co-occur, numeral zero and lowercase oh were differentiated by different size and/or different ductus, at least in type, where the more circular form and lack of thick-thin shading in the zero distinguished it from the humanist oh, which did have thick-thin shading.

Later, in the 16th century and especially in France, Arabic numerals gradually became more often used with roman typefaces. A type specimen

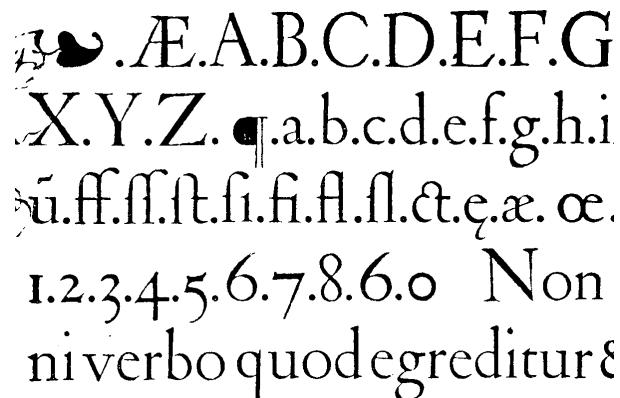


Figure 16: Type specimen of François Guyot, circa 1565. Type Specimen Facsimiles, ed. John Dreyfus. Bowes & Bowes and Putnam, London. 1963. Original document in Folger Library, Washington, D.C. Note numero-typographical error of ‘6’ substituted for rotationally symmetrical ‘9’.

¶§ Epreuve de la petite Antique ou §§
Romaine, de Rob, Granjon nom-
mée la Gaillarde. en Anvers
¶§ 1570. §§

Oratio Susannæ. Danielis xij.
Deus æterne, qui absconditorum cognitor

Figure 17: Gaillarde by Robert Granjon, 1570. Type Specimen Facsimiles II, H.D.L. Vervliet and Harry Carter, ed. John Dreyfus. Bodley Head, London, and University of Toronto Press, Toronto. 1972. Original document in Plantin-Moretus Museum, Antwerp. Original approximately 9 point.

attributed to François Guyot (Figure 16), circa 1565, displays complete Arabic numerals for several sizes of type. As in 15th century Arabic numerals, Guyot’s numerals had ascending and descending forms. The zero is cut as a small circular ring roughly the size of a lowercase oh but without thick-thin shading. Guyot’s types are cut in the style of Garamond, a canonical form in 16th century typography, and may be the earliest example of Arabic numerals cut for each size and style of type by the punch-cutter and cast by the typefounder.

A specimen of a small (approximately 9 point) roman, named “Gaillarde”, cut by Robert Granjon, is dated 1570, with the circular ring-form zero (Figure 17). (The cutting is very fine, but the photo reproduction of the printed specimen makes it look rougher than it is.)

Thus, during the second half of the 16th century, Arabic numerals became incorporated into common expectations of what characters a “font” contained—at least capital and lowercase letters, punctuation,

Oh, oh, zero!

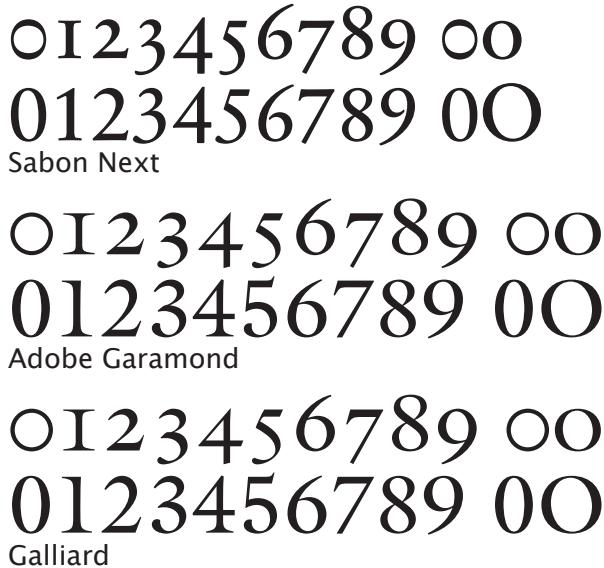


Figure 18: Old-style and lining numerals from three modern revivals of Oldstyle typefaces: Sabon (Garamond), Adobe Garamond, and Galliard. Old-style numerals are in the first line, with old-style zero and lowercase oh for comparison, and lining numerals in the second line, with lining numeral zero and capital Oh for comparison.

and Arabic numerals. This amalgamation of disparate forms became the standard in printing from the 16th to the 19th century.

Modern revivals of 16th to 18th century types often include old-style numerals in addition to lining numerals, which are usually capital height (Figure 18).

Sabon by Jan Tschichold (1967) and Sabon Next by Tschichold and Jean François Porchez (2002) are revivals of types cut by Claude Garamond circa 1550 (an exact date is difficult to ascribe because Tschichold may have used more than one Garamond model). Porchez suggests that Tschichold's design was also influenced by types cut in Garamond's style by Guillaume Le Be, a younger contemporary of Garamond. In Sabon, the old-style zero is shaded but with heavier strokes at top and bottom instead of left and right, thus reversing traditional shading so as to reduce potential confusion between zero and lowercase oh. This zero design may be an invention by Tschichold, not Garamond. In the lining numerals of Sabon, the zero is capital height and distinctly taller than lowercase oh, so there is little possibility of confusion with oh. The lining zero has traditional shading—lighter strokes at top and bottom, heavier at left and right, as with traditional capitals, which reduces the difference between zero and capital Oh, but because the zero is distinctly narrower than capital Oh, the difference between them is evident.

Adobe Garamond by Robert Slimbach (1989), another revival of types cut by Garamond, uses a monoline ring form of zero like that seen in the Guyot specimen.

Galliard by Matthew Carter (1978), based on designs by Robert Granjon, also uses the monoline ring zero. The name is taken from Granjon's "Gaillard" type but is not an exact copy of that particular size.

In the late 19th century, the typewriter was developed and became popular. Huckleberry Finn, published in 1884, is said to have been the first type-written manuscript submitted to a printer. The dominant style of printing type was known as "Modern", which includes a broad range of designs, from the elegant cuttings of Bodoni and Didot, used today for high-fashion advertising, to their workaday descendants, including Scotch Roman, American Monotype Modern 8a and Donald Knuth's Metafont derivative of it, Computer Modern.

Modern typefaces arose in the last decades of the 18th century, and included new proportions for the designs of numerals. Instead of the old-style numerals with ascenders and descenders extruding above the x-line or below the baseline, Modern-style numerals were cut so that the tops of the numerals all aligned. The first of these equal-height numeral sets was a late Transitional face cut by Richard Austin for John Bell, in 1788; the numeral height was intermediate between capital height and x-height. This style of numeral was adopted by other English and Scottish type foundries. The typeface called Scotch Roman is derived from types originally founded in the early 19th century by Scottish foundries, in particular the William Miller foundry in Edinburgh, Scotland. Certain of the Scottish types were recast nearer the end of the 19th century and sold in the U.S. under the name Scotch Roman. The dark version of Scotch Roman produced in the early 20th century by Monotype has lining numerals slightly shorter than the capitals.

The 1815 catalog of the London typefoundry of Vincent Figgins shows lining numerals for a range of text faces, and the numeral height is equal to that of the capitals. The 1828 catalog of the Edmund Fry typefoundry also shows lining numerals for a range of text faces, and the numeral height is equal to that of the capitals.

Bringhurst (1996) suggests that this change from old-style to lining numeral designs derived from handwritten numerals in English shop signs and placards in the 18th century, and was thus a consequence of the rising British middle-class. However that may be, it is evident that by the time the typewriter was developed at the end of the 19th century, the

0123456789 0Oo
Bell

0123456789 0Oo
Scotch Roman

Figure 19: Revivals by Monotype of the Transitional typefaces Bell and Scotch Roman. Numerals align in height, but are slightly shorter than the capitals, as seen in the zero Oh oh combinations.

standard model for numerals had become the Modern style of equal heights (Figure 19).

To reduce complexity, both in the mechanics of the machine and in the mind of the typist, nearly all typewriters used monospaced (fixed-width) characters. (An exception was the IBM electric Executive typewriter in the 1940s and 1950s, and some models of the IBM Selectric typewriters made in later years, which used proportional spacing.) In the “fonts” of most typewriters, therefore, zero and Oh could not be differentiated by width. Moreover, most typewriter faces were unshaded or mono-line, that is, lacking the thick-thin modulation of printing types, so zero and Oh could not be differentiated by stroke shading. Thus, a zero and an Oh with same width, height, and monoline stroke looked very much the same in typewriting.

In typewritten documents before the computer era in mid-20th century, context was presumably sufficient to distinguish numerals from letters, but in computing, there appears to have been greater mingling of numerals with letters, giving rise to the issues discussed by Bemer (1967). In computer printing, the symbol shapes were occasionally differentiated by making one more rectangular and the other more oval, but there was no agreement on which symbol should be which shape, so confusion continued. Exacerbating the problem, similar confusion of zero with Oh also occurred in speech, when the numeral zero was, and indeed still is, often pronounced as “oh”. A comment in Bemer (1967) points out that the popular Boeing 707 aircraft was called a ‘seven-oh-seven’, not a ‘seven-zero-seven’. On a telephone dial or keypad, the “Oh” for “Operator” is the zero key, not the 6 key (covering MNO). Telephone area codes are likewise spoken with the vowel oh or the spelling “oh” representing zero, as in ‘five-oh-three’ (503), the area code for northwest Oregon.

On many manual typewriters, the numeral one and lowercase letter ell were merged entirely, with one glyph on one key representing both graphemes, presumably for keyboard economy. In computing, the one-ell pair became visually confusable when low-

```

10      FORMAT(I)
DO 200 I=1,N
READ 1,110,V
FORMAT(E)
SUM=SUM + V
SUMSQ=SUMSQ +
TYPE 120,I,V
110

```

Figure 20: Part of a Fortran program printed by an ASR 33 Teletype. In this capitals-only font, the zero has a slash to distinguish it from the Oh. The sample shows that worn or poorly adjusted print heads may erode the differences between characters, such as numeral one and capital I in the expression “I=1”. http://www.pdp8.net/asr33/pics/fortran_printout.shtml

ercase was added to the ASCII standard (USA, 1967). As in the case of zero and Oh, context was not sufficient to differentiate such glyphs in the computer era.

When computer input and print-out began to be done through Teletype machines, the zero and Oh were distinguished by a slash through the zero, as in the print-out from a Teletype ASR 33 (Figure 20).

Although traditional typewriter fonts often did not distinguish zero from Oh, special-purpose fonts in the computer era have sometimes emphasized the difference, as recommended by proposals made in Bemer (1967). In OCR-A, a font devised for optical character recognition (OCR) by ANSI, the zero is nearly rectangular and the Oh is lozenge-shaped. The first version of the font was produced in 1968 by American Typefounders. Technological progress has made OCR-A obsolete as an OCR font, but it is still used as a display font to give a retro-techno look to images and documents. In the font OCR-B designed by Adrian Frutiger in 1968, the zero is taller and more rectangular than the oval Oh. The one, capital I, and ell are differentiated by the presence, location and angularity or curvilinearity of serifs (Figure 21). It is notable that some features of monospaced fonts designed four decades later imitate the design solutions that Frutiger devised in the 1960s (see Figure 9).

5 Conclusion

Latin letters and Hindu-Arabic numerals evolved independently in handwriting in separate cultures, but each of the resulting sets of characters developed a small, open, circular or elliptical form: zero in the numerals; oh in the letters. There was no problem of confusion between the numerals and letters until they were used together in European texts in the 12th and 13th centuries, when Hindu-Arabic numerals began to be adopted by European mathematicians.



Figure 21: Traditional Typewriter revival by Monotype, OCR-A, and OCR-B.

For a few centuries, however, different contexts and, sometimes, different writing styles, appear to have been sufficient to disambiguate the meaning of the similar-looking glyphs.

In the 15th century, however, printing vastly increased the production and distribution of books and also increased pressure for standardization of character shapes when marketing books to much larger international readership. Early in the appearance of the numeral zero in print, and again near the end of the 15th century, zero took on a circular shape that was more or less unshaded, that is, without thick-thin contrast. This ring-like shape helped distinguish zero from the (typically) shaded letter oh of typefaces derived from humanist handwriting. The ring-like zero was eventually adopted for most roman typefaces in the 16th and was used until the end of the 18th and beginning of the 19th century.

In the last decades of the 18th century, a new style of numeral appeared that was more or less the same height as the capital letters. The increased height made zero easily distinguishable from oh, but created a new confusable pair: zero and capital Oh. In printing types, the zero glyph was generally narrower than Oh, so the two characters could still be distinguished, but in the last decades of the 19th century, typewriters forced all characters to have the same width, thus eliminating this width difference between zero and Oh. Context was apparently still sufficient to distinguish typewritten zero and Oh in most correspondence and documents, but in the mid-20th century, computer printers using typewriter-like fonts, and computer code that used greater mixing of letters and numerals, exacerbated the confusion. There ensued several decades of design proposals, arguments, and experiments in developing better differentiation of zero and Oh. Proposals from the “humanist” camp usually were to modify the zero, whereas proposals from the “engineer” camp were usually to modify the Oh. Technological progress in computing sometimes aided differentiation but sometimes

hindered it, whether through changes in imaging or through proliferation of confusable character forms.

In actual fonts developed for digital systems over the past three decades, signs of consensus have emerged: the zero glyph is usually the one modified, whether with a slash or a dot or other means, while the capital letter Oh is unscathed. To this extent, the humanists have been victorious. Other confusable characters have arisen, however, including the triplet of capital ‘T’, lowercase ell, and numeral one. Some trends have emerged among those glyphs as well, though not as clearly as for zero and Oh. Incorporation of mathematical symbols in fonts, and increasing use of mathematical symbols in electronic documents, enable further kinds of confusion, so for type designers, document designers, and readers, the problem of look-alike symbols has not been entirely solved.

6 Acknowledgements

I thank Karl Berry and Barbara Beeton for suggesting this article and patiently waiting for it, and for making valuable suggestions to improve the text and images, and Karl again for putting it all into T_EX format. Thanks to Steven Galbraith and Amelia Hugill-Fontanel for making the splendid resources of the RIT Cary Collection available for study of this subject, and Amelia again for photographing the page of the Hypnerotomachia Poliphili. Thanks to Rolf Rehe for help obtaining and understanding parts of DIN 1450, and to Otmar Höfer for comments on Karl Georg Höfer’s design of the alphabet for German vehicle license. Thanks to Kris Holmes for assembling the image for Figure 21 and for many years of inspired collaboration on the designs of the Lucida fonts.

7 References

- American Standards Association. American Standard Code for Information Interchange, ASA X3.4-1963, June 17, 1963.
- Aristotle. Metaphysics, Books I–IX. (H. Tredennick, trans.) Loeb Classical Library No. 271, Met. 1.985b, Harvard University Press, 1989.
- Bemer, R.W. Toward standards for handwritten zero and oh: Much ado about nothing (and a letter), or a partial dossier on distinguishing between handwritten zero and oh. Communications of the ACM 10(8), August 1967.
- Bigelow, C. and Holmes, K. Notes on Apple 4 Fonts. Electronic Publishing 4(3), pp. 171–181, 1991. <http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume4/issue3/ep050cb.pdf>
- Bringhurst, R. The Elements of Typographic Style. Hartley & Marks, Vancouver, CA, 1996.
- Cajori, F.A. History of Mathematical Notations. Dover Publications, Mineola, NY, 1993. (Original edition: Open Court, Chicago, 1928–29.)

- Deutsches Institut für Normung. DIN 1450 Schriften – Leserlichkeit (DIN 1450 Typefaces – Legibility), DIN Beuth. 2013.
- Gorn, S., Bemer, R.W., Green, J. The American Standard Code for Information Interchange. Communications of the ACM 6(8), August 1963.
- Ifrah, G. The Universal History of Numbers. (D. Bellos, E.F. Harding, S. Wood, I. Monk, trans.) John Wiley & Sons, New York, 1998.
- Kerpelman, C. Proposed American National Standard: Presentation of alphabetic characters for information processing. Communications of the ACM 12(12), December 1969.
- Lo, P.P. Use Chinese for zero and oh? Communications of the ACM, 10(12), December 1967.
- Proposed revised American Standard Code for Information Interchange. Communications of the ACM 8(4), April 1965.
- Schulz-Anker, E. Syntax-Antiqua, a sans-serif on a new basis. *Gebrauchsgraphik* 7, 1970.
- Unicode Consortium. Unicode Standard 5.0. Addison Wesley, 2007.
- United States of America Standards Institute. USA Standard Code for Information Interchange, USAS X3.4-1967, revision of X3.4-1965, 1967.
- Vartabedian, A. A Proposed Fontstyle for the Graphic Representation of the Oh and Zero. *The Journal of Typographic Research* 3(3), pp. 249–258, 1969.
- Vartabedian, A. Reply. *The Journal of Typographic Research* 4(2), pp. 181–183, 1970.
- Wendt, D. O or 0. *The Journal of Typographic Research* 3(3), pp. 241–248, 1969.
- Zapf, H. Proposal. *The Journal of Typographic Research* 4(2), pp. 179–180, 1970.
- ◊ Charles Bigelow
<http://www.lucidafonts.com>

Production notes

Karl Berry

The most *TeX*nically unusual part of this article, and of the entire issue, was handling the rare characters shown in the footnote on the first page and the rundown of circle-slash characters on the next two pages. Although they could have been inserted as small images, the author (Chuck Bigelow) sent me fonts including them, so I wanted to try typesetting them directly. He wanted to typeset them all in a consistent font, rather than mixing glyphs from Computer Modern and other sources.

The first version Chuck sent me was in *.otf* format, with the characters we wanted (zero-slash, prohibition, etc.) replacing lowercase letters. So it sufficed to start up FontForge (by George Williams, fontforge.sf.net) and use its ‘Generate Fonts’ feature to create a *.pfb + .afm*, which takes the first 256 characters. Easy. (I wanted to use Type1 since this was happening quite far along in the article’s processing, and I had been using pdf \backslash TEX thus far; switching to X \backslash TEX or Lua \backslash TEX would have meant losing functionality from *microtype* and thus losing considerable time fixing line breaks.)

Then Chuck sent me a revised font with additional characters. This time it was a *.ttf*, and the characters were in the correct Unicode positions (which are far beyond the first 256 characters, of course), so I couldn’t just use the simple FontForge generation. (I could have asked Chuck to rearrange the characters, but I decided to take it as a challenge; after all, it’s an article of our faith that *TeX* should be able to use any font.)

Instead, I followed the article by H \acute{a} n Th \acute{e} Thành about using TrueType fonts directly in pdf \backslash TEX (30:1, tug.org/TUGboat/tb30-1/tb94thanh.pdf). First I created a custom encoding file, *altzero.enc*, starting like this:

```
/enclucidaaltzero [
  /emptyset    % U+2205
  /uni20E0     % prohibition
  /emptyset.var % glyph index #2225
  ...
]
```

These character names are specified in the font. I discovered them by looking at the font in FontForge and using

‘View → Goto’ to navigate to the characters; thankfully, searching for *uni...* works even when the character does not have a name of that form. Chuck told me the name of the variant emptyset glyph (zero-slash in this case), which does not have a Unicode assignment.

Still following Thành’s article, I then made the *.tfm*:

```
ttf2afm -e altzero.enc -o altzero.afm ZeroFont.ttf
afm2tfm altzero.afm
```

In the *L \backslash ATEX* document, the font was used like this:

```
\pdfmapline{+altzero ZeroFont <altzero.enc
            <ZeroFont.ttf}
\font\altzero = altzero
{\altzero\char0}%
% of our encoding: emptyset
```

All was fine, until Chuck sent me one more revision of the font. This time it was again *.otf*, but now using the Unicode positions. pdf \backslash TEX cannot read *.otf*, and converting *.otf* to *.ttf* seemed fraught with potential problems to me. So I used a third tool: *otftotfm* (by Eddie Kohler, 1cdf.org). Once I read the documentation enough times, I happily discovered that I could re-use the same encoding file. The invocation this time:

```
otftotfm -e altzero.enc --no-encoding \
          ZeroFontOT.otf altzero
```

(The *--no-encoding* option just tells *otftotfm* not to generate its own new encoding file; the final *altzero* argument is the base name of the *.tfm* and *.pfb* generated by *otftotfm*.)

Usage in the *L \backslash ATEX* source is similar to the above, but now we have a *.pfb*:

```
\pdfmapline{+altzero ZeroFontOT <altzero.enc
            <ZeroFontOT.pfb}
```

The tools themselves output the map lines needed, according to the names embedded in the font files, etc.

Moving on from the technicalities, it was a great pleasure to work with Chuck on his articles in this issue. He has had a great (and positive!) influence on me, with recommendations for schools to attend, professors to work with, and personally encouraging my lifelong interest in typography and typesetting. As it turned out, we effectively finished work on the article on Chuck’s birthday. Happy birthday Chuck!



The Treasure Chest

This is a list of selected new packages posted to CTAN (<http://ctan.org>) from March 2013 through July 2013, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TeX community. Comments are welcome, as always.

◊ Karl Berry
 tugboat (at) tug dot org
<http://tug.org/ctan.html>

fonts

divine in fonts

Informal calligraphic font from the T.26 foundry.

fontawesome in fonts

Font of high-quality web icons.

*gentium-tug in fonts

Includes Type 1 as well as OTF Gentium fonts, with both L^AT_EX and Con^T_EXt support.

*librecaslon in fonts

(L^A)T_EX support for this revival by Pablo Impallari.

nanumtype1 in fonts

Unicode font for the Korean script, in Type 1.

newpx in fonts

Revised metrics, options for (mostly) pxfonts glyphs.

raleway in fonts

Support for the Raleway sans serif font family.

sansmathfonts in fonts

Sans serif small caps and math for use with CM.

graphics

pxpgfmark in graphics/pgf/contrib

Make PGF's inter-picture connections work with e-pT_EX & dvipdfmx.

spath3 in graphics/pgf/contrib

Manipulating Ti_KZ soft paths, e.g., calligraphic paths and knot diagrams.

tikz-bayesnet in graphics/pgf/contrib

Bayesian networks, graphical models, factor graphs.

info

*detexfaq in info/german

German FAQ extracted from texfragen.de.

fonts/divine

language

greek-fontenc in language/greek

Greek font encoding definition files (LGR).

ptex2pdf in language/japanese

Script to get PDF output from pT_EX-based engines.

macros/generic

xint in macros/generic

Expandable arithmetic operations.

macros/latex/contrib

bkeepic in macros/latex/contrib

Implement eepic via pict2e.

changes in macros/latex/contrib

Manually mark up textual changes.

coneq in macros/latex/contrib

Automated layout of a continued equality.

download in macros/latex/contrib

Download via wget or curl from L^AT_EX.

ebook in macros/latex/contrib

Style for PDF e-books on small readers.

*esami in macros/latex/contrib

Typeset exams with question types, randomization, support for several languages, and more. (<http://tug.org/TUGboat/tb34-1/tb106messineo.pdf>)

factura in macros/latex/contrib

Issue invoices in Venezuelan style.

feuphdteses in macros/latex/contrib

Template for theses of FEUP in Portugal.

feynmp-auto in macros/latex/contrib

Automatically run MetaPost on feynmp output.

horoscop in macros/latex/contrib

Astrological charts for L^AT_EX.

iitem in macros/latex/contrib

Convenient writing of nested list items.

interval in macros/latex/contrib

Format math intervals with proper spacing.

leipzig in macros/latex/contrib

Typeset and index linguistic gloss abbreviations.

lengthconvert in macros/latex/contrib

Convert a given length to another unit.

lplfitch in macros/latex/contrib

Typeset natural deduction proofs in Fitch style.

matc3,matc3mem in macros/latex/contrib

Commands and a class for MathematicaC3 project textbooks.

memory in macros/latex/contrib

Declare object or array containers in L^AT_EX.

mnotes in macros/latex/contrib

Margin notes for collaborative writing.

newenviron in macros/latex/contrib

Collect and execute environment bodies.

noconflict in **macros/latex/contrib**
 Resolve macro conflicts among packages.

nox in **macros/latex/contrib**
 Arrays of text transformed to a variety of tables.

numberedbloc in **macros/latex/contrib**
 Include and sequentially label code snippets.

pfarrei in **macros/latex/contrib**
 Arrange A5 output onto A4 landscape.

pythontex in **macros/latex/contrib**
 Run Python code from within L^AT_EX.

rterface in **macros/latex/contrib**
 Access to R from within L^AT_EX.

skdoc in **macros/latex/contrib**
 Documentation class loosely based on ydoc.

skmath in **macros/latex/contrib**
 Enhanced math typesetting.

skrappo in **macros/latex/contrib**
 Class for simple reports.

snotez in **macros/latex/contrib**
 Sidenote support.

stackengine in **macros/latex/contrib**
 Customizable stacking of general objects.

tabriz-thesis in **macros/latex/contrib**
 X_{EL}L^AT_EX class for theses at the Univ. of Tabriz.

textglos in **macros/latex/contrib**
 Typeset inline linguistic examples.

thalie in **macros/latex/contrib**
 Typeset dramatic plays.

titlecaps in **macros/latex/contrib**
 Extended support for capitalization of initial letters, including accents, font changes, and more.

udesoftec in **macros/latex/contrib**
 Template for theses at the Univ. of Duisburg-Essen.

unswcover in **macros/latex/contrib**
 Thesis cover page for the Univ. of New South Wales.

uowthesistitlepage in **macros/latex/contrib**
 Thesis title page for the Univ. of Wollongong.

vdmlisting in **macros/latex/contrib**
 Add-on for *listings* to support VDM in ASCII.

macros/latex/contrib/babel-contrib

thai in **m/l/c/babel-contrib**
 Babel support for Thai.

macros/latex/contrib/beamer-contrib

beamertHEME-upenn-bc in **m/l/c/beamer-contrib**
 Color themes for Boston College and the University of Pennsylvania.

bxdpx-beamer in **m/l/c/beamer-contrib**
 Make navigation symbols and \framezoom regions work with dvipdfmx.

macros/latex/required

***babel** in **macros/latex/required**
 Specific language support now separated from the core package; new contributions welcome.

macros/luatex

enigma in **macros/luatex/generic**
 Encryption at the node level with a three-rotor Enigma machine.

luabidi in **macros/luatex/latex**
 Bidirectional typesetting in LuaT_EX.

selnolig in **macros/luatex/lateX**
 Selectively suppress ligatures.

macros/plain

expex in **macros/plain/contrib**
 Typeset linguistic examples and glosses, including referencing.

macros/xetex

xevlna in **macros/xetex/generic**
 Automatic insertion via X_{EL}L^AT_EX of nonbreakable spaces (in, e.g., Czech).

xetexko in **macros/xetex/generic**
 Typesetting of Korean, including Old Hangul.

support

classpack in **support**
 XML mastering for L^AT_EX packages.

create-struktex in **support**
 Java support for Nassi-Shneiderman structure charts.

intex in **support**
 Aid consistent typesetting of phrases, acronyms and proper names.

latex-git-log in **support**
 Generate git version history as L^AT_EX source.

latexitndent in **support**
 Perl script to indent L^AT_EX files.

try in **support**
 Python script for automated compilation via source comments.

wheretotrim in **support**
 Suggest where to trim text to reduce page count.

systems

parsitex in **systems**
 WEB change file for localized Persian and bidirectional extension of T_EX.

Preparing for scientific conferences with L^AT_EX: A short practical how-to

Paweł Lupkowski and Mariusz Urbański

Abstract

In this paper we will present a short practical how-to guide considering the complete experience of preparing materials for a scientific conference. This will cover the preparation of: a paper with figures and charts, a PDF presentation and a conference poster. The paper is based on our hands-on experience in this area.

Introduction

The main aim of this paper is to present a short practical how-to guide considering the complete experience of preparing materials for a scientific conference. Attending conferences is one of the essential aspects of being a researcher. There are a huge number of conferences which involve preparing different materials, such as papers, presentations and posters. What is more, organisers of many of these conferences demand preparing at least some of these materials (usually papers) in L^AT_EX. This paper covers our experience in attending conferences in different domains (cognitive science, psychology, logic, computer science, linguistics). Our experience is that L^AT_EX constitutes a single extremely efficient environment to prepare all the necessary materials. One major virtue of L^AT_EX is that it allows for easy use of the content in different forms (whether a paper, a presentation, or a conference poster).¹

We are not aiming at novelty in this paper. Our aim is to write down and share our tips and tricks used in everyday work with L^AT_EX in science. We hope that it will be a useful guide, especially for L^AT_EX beginners and also for scientists wondering which combination of tools will be useful in their work and is worth the time and effort to learn. We assume that our reader has a basic knowledge and skills in typesetting in L^AT_EX.² Preparing for a conference is often done under time pressure. We hope that this paper will serve as a quick reference for those of you who don't have time to read manuals and search for specific solutions.

The paper is structured as follows. The first section covers preparing a conference paper, the second section a presentation, and the third a poster.

Originally presented at EuroBachotEX 2013.

¹ It is worth mentioning that there is a method which allows to produce three types of outputs (a paper, a poster and a presentation) from one input file—see [1].

² For a paper with convincing reasons to start using L^AT_EX in the first place, see [6].

In the summary we point at some issues that we find difficult or disturbing in our work with L^AT_EX in preparing materials for conferences.

1 Conference paper

We attend conferences in different domains. Usually, the organisers ask for submission of a talk proposal in a form of a short or a long paper. In such a case you should check for a paper template provided by the organisers. Sometimes it is a specially prepared class for the paper (as for the *Logic and Cognition* conference³), or a general class used for certain domains, like the EACL class for linguistic conferences (cf. *Semodial*⁴). Also, specific requirements for the most popular *article* class might be given (cf. *IWCS 2013*⁵). You should always search for a *Call for Papers* or *Information for Authors* section. Many different solutions used in different L^AT_EX classes (especially for author and title fields) might be difficult to grasp, and sometimes reading example articles and guidelines provided by the organisers is necessary. When you prepare your paper in advance, our advice is to simply use the *article* class without any custom commands and modifications. Then adjusting your source to the conference requirements should be fairly easy.

One of the problems that we often encounter preparing our papers is how to produce high quality pictures in an easy way. Below we present a class for preparing bar charts within L^AT_EX. We also describe how you can use Inkscape to generate pictures, which might be then exported into a L^AT_EX code and used directly in your paper. It is worth mentioning that using the presented methods involves compiling your file first with L^AT_EX and then with a DVI-to-PostScript converter such as *dvips*.

Bar charts One of the simplest ways to put a chart into a paper (not using software other than L^AT_EX itself) is the *bchart* package. If you don't already have it installed, it can be obtained from CTAN: <http://www.ctan.org/pkg/bchart>.

The basic structure of a chart is the following:

```
\begin{bchart}[max=5, step=1]
\bcbar{4.5}
\end{bchart}
```

The option `max` defines the maximum value on the *x*-axis (the default is 100). If you want to define regular steps along the *x*-axis, you can use the `step` option (or, an option `steps={...}` allows you to give irregular step values, e.g. `steps={1, 3, 5}`).

³ <http://logicandcognition.org/>

⁴ <https://sites.google.com/site/semdial2012seinedial>

⁵ <http://www.ling.uni-potsdam.de/iwcs2013/>

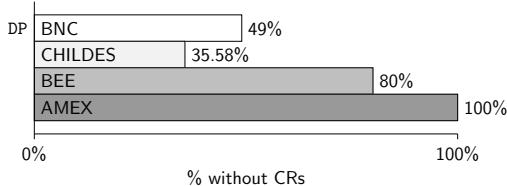


Figure 1: An example bar chart using *bchart*

The command `bcbar` puts a bar with a given value. For this command you can use several options, like `text` (to print some text inside the bar), `color` (to define the bar's colour), `label` (to assign a label to the bar, which will be visible on the left side of the bar). To add a caption to the *x*-axis use the command `\bcxlabel{...}`.

A very convenient feature is that the charts are scalable. There are two ways to obtain this effect. First, you can scale only the chart (without a text) by adding the option `scale` into the `bchart` command. The second one is to put the chart into the command `\scalebox{<factor>}{...}`.

An example chart presented in Figure 1 is generated by the following code:

```
\scalebox{0.7}{
\begin{bchart}[max=100, unit=\%]
\bcbar[label={\tt DP}, text=BNC,
       color=white]{49}
\bcbar[text=CHILDES, color=gray!10]{35.58}
\bcbar[text=BEE, color=gray!50]{80}
\bcbar[text=AMEX, color=gray!80]{100}
\bcxlabel{\% without CRs}
\end{bchart}
}
```

Other useful commands available via this package can be found in [8].

L^AT_EX and Inkscape To include a more sophisticated picture into your paper, you may want to use Inkscape (an open source vector graphic editor).⁶ Using Inkscape you can prepare your picture, then export it to `*.png` or `*.pdf` format, and afterwards include it into your paper in the usual way.⁷ However, one of the most convenient ways is to use the L^AT_EX export function in Inkscape. To do this choose *File > Save as ...* and then pick the *L^AT_EX with PSTricks* option. This will save your drawing into a `*.tex` file. Then you can put it in your paper with the `\input{foo.tex}` command, or—to obtain a self-contained paper—simply copy and paste the code into the paper's source. Remember also to load the `pstricks` package in your preamble. An example picture prepared in Inkscape and then embed-

⁶ <http://inkscape.org>

⁷ For a short, but comprehensive, guide see [2].

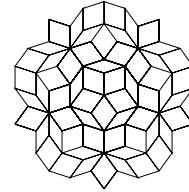
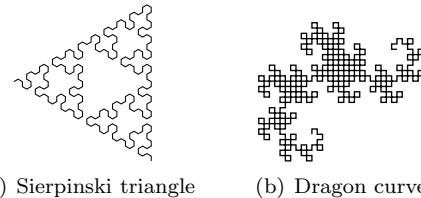


Figure 2: A drawing prepared in Inkscape and exported to L^AT_EX (figure from the Inkscape examples set, file *l-systems.svgz*)



(a) Sierpinski triangle (b) Dragon curve

Figure 3: Side by side figures (figures from the Inkscape examples set, file *l-systems.svgz*)

ded in this paper is presented in Figure 2. Such a picture might be easily scaled using the same command as in the *bchart* case, namely `\scalebox{...}`.

Inkscape (from version 0.48) can also export the graphics to PDF/EPS/PS, and the text to a L^AT_EX file, so you can obtain a vector image with the same font and size as in normal text. This might be especially useful when you want to prepare a picture with mathematical formulas included.⁸ For the Inkscape manual, see [5].

Side by side figures Sometimes there is a need to arrange pictures inside a figure environment (e.g. to save some space for the main text). This can easily be done by using the `subfigure` package.⁹ An example of such a solution is presented in Figure 3. Below you may trace the code used to generate the example. Each picture is placed with a `subfigure` command. A very useful feature is the optional argument to supplement each figure with a description. As a consequence, you may easily produce a complex figure with a caption and description for each of its elements.

```
\mbox{\subfigure[Sierpinski triangle]{%
\input{sierpinski.tex}}%
\quad%
\subfigure[Dragon curve]{\input{dragon.tex}}%
}
```

⁸ L^AT_EX and Inkscape fans might be also interested in an extension for Inkscape that allows using L^AT_EX inside the drawing program itself. See <http://www.johndcook.com/blog/2009/12/22/including-latex-in-inkscape/>.

⁹ From <http://www.johndcook.com/blog/2009/01/14/how-to-display-side-by-side-figurs-in-latex/>.

2 Conference presentation

Probably the most popular way of preparing presentations with L^AT_EX is by using the *Beamer* class.¹⁰ It allows for fast and easy preparation of presentations (especially on the basis of a previously written paper). Beamer produces a PDF file on output, which guarantees that the presentation will look exactly the same on different machines. In addition, it allows using sections and subsections to structure the presentation in a paper-like style—very convenient when we build a presentation on the basis of a paper. Last but not least, it allows preparing attractive presentations with well-balanced colour schemes and slide elements that make it easier for the audience to track the presentation (number of slides, short title, author etc.).

If you would like to take a look at the complete set of default Beamer themes available, see [9, Section 30]. You will find there names and pictures of Beamer themes (title slide and a regular one). If you are interested in the range of possible modifications of the standard themes see [9, Section 29].

Slide customisations The first modification you may want to make is to change the basic colour of your presentation. You may have a situation when you want to match the colour scheme to the colours used in your poster, or your corporate identity colour scheme etc. The simplest way is to use RGB colour description (the numerical values for a given colour might be obtained easily using Inkscape or GIMP). To use the colour you've picked, include a `\usecolortheme` command into the preamble of your presentation, as shown:

```
\documentclass{beamer}
\usecolortheme[RGB={241,200,144}]{structure}
\usetheme{Warsaw}
```

One of the useful features of Beamer is the possibility of changing the theme (and its colour scheme) of the whole presentation after it is prepared. This is might be useful especially when technical conditions for your presentation will not match the theme you used for the presentation (e.g. the room where you are giving your talk is too bright or the beamer is of poor quality). Changing one line of L^AT_EX code (an argument of the `\usecolortheme` command) might make your slides visible again—see Figure 4, where one slide is presented in two different colour themes (*dove* and *albatros*). However, you should remember

¹⁰ Other options are e.g. the *powerdot* class (<http://www.ctan.org/pkg/powerdot>), the *prosper* class (<http://www.ctan.org/pkg/prosper>), and an interesting KOMA-script based presentation—see [7] and <http://www.latextemplates.com/template/koma-script-presentation>.

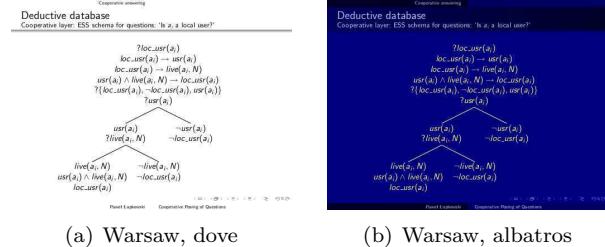


Figure 4: Beamer allows for easy modification of the colour scheme of your presentation, so you can adjust it to your needs (even after the presentation is ready)

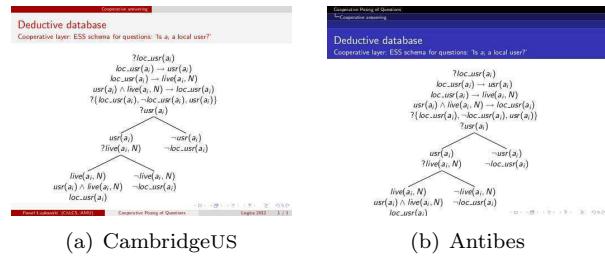


Figure 5: When changing the slide theme remember that themes have different slide architecture, and sometimes the content does not fit the new theme

that some themes have more extensive slide structure than others (see e.g. Figure 5; the content which fits the slide in the CambridgeUS theme does not fit the slide in the Antibes theme). Thus it is useful to check in advance which theme changes are safe for your content.

Beamer offers also the possibility of setting a background image for the slides (see Figure 6). To do this, use the following command in the preamble of your document [9, Section 27]:

```
\setbeamertemplate{background canvas}
{\includegraphics[width=\paperwidth,
height=\paperheight]{backgroundimg.jpg}}
```

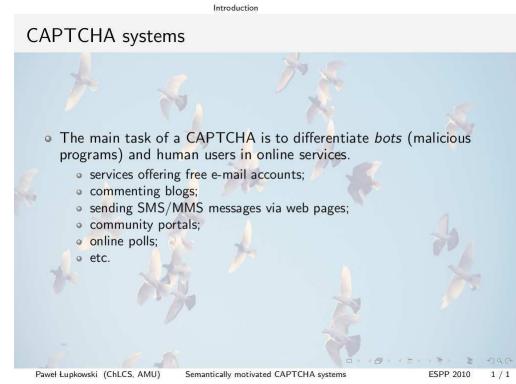


Figure 6: Custom background for a Beamer slide

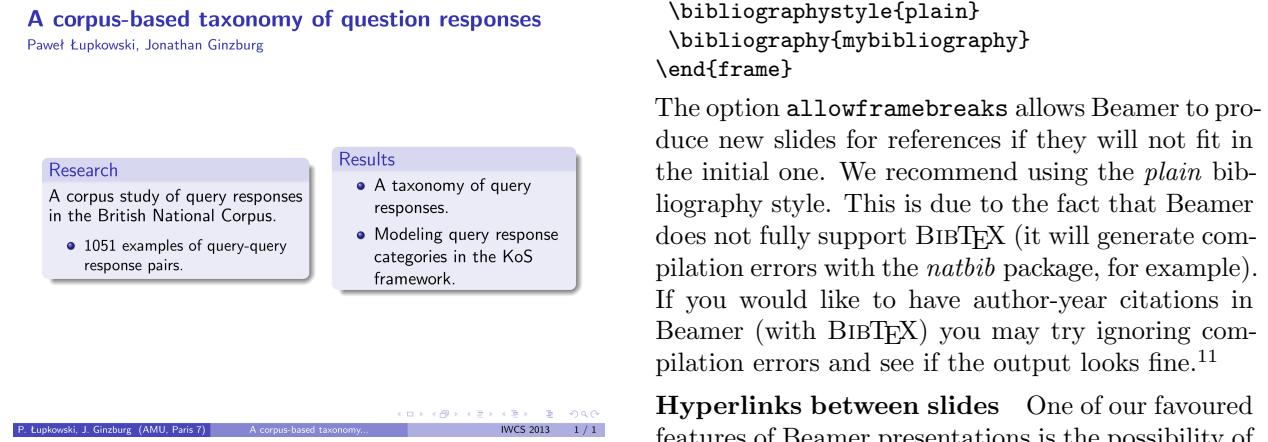


Figure 7: Side by side content of a Beamer slide with the `columns` environment

Side by side content on a slide Sometimes you may need to arrange the content of a slide in two (or three) columns. One way to do this is to use the `columns` environment. Below you will find sample code, which generates the slide presented in Figure 7. You may put pictures, blocks, lists and regular text inside columns. The option `[c]` of the `columns` command centres columns vertically. Each `column` environment constitutes one column, where you give its width as an argument of the `\begin{column}` command.

```
\begin{columns}[c]
    \begin{column}{5.5cm}
        \begin{block}{Research}
            A corpus study of query responses in the British National Corpus.
        \end{block}
        \begin{small}
            \begin{itemize}
                \item 1051 examples of query-query response pairs.
            \end{itemize}
        \end{small}
    \end{column}
    \begin{column}{5cm}
        \begin{block}{Results}
            \begin{itemize}
                \item A taxonomy of query responses.
                \item Modeling query response categories in the KoS framework.
            \end{itemize}
        \end{block}
    \end{column}
\end{columns}
```

References with BIBTEX If you want to include references in your presentation, and you want to use BIBTEX, we recommend the following method:

```
\begin{frame}[allowframebreaks]{References}
    \def\newblock{}
\end{frame}
```

```
\bibliographystyle{plain}
\bibliography{mybibliography}
\end{frame}
```

The option `allowframebreaks` allows Beamer to produce new slides for references if they will not fit in the initial one. We recommend using the *plain* bibliography style. This is due to the fact that Beamer does not fully support BIBTEX (it will generate compilation errors with the `natbib` package, for example). If you would like to have author-year citations in Beamer (with BIBTEX) you may try ignoring compilation errors and see if the output looks fine.¹¹

Hyperlinks between slides One of our favoured features of Beamer presentations is the possibility of defining non-linear changes of slide via the use of hyperlinks. Using this feature you may skip slides in case of lack of time or unveil hidden slides if it appears that you have some time left. You can also have extra slides prepared for discussion after presentation and easily access them using hyperlinks (without the necessity of skipping lots of slides in front of the audience).

First you should define the target for the hyperlink. You do this adding a label for a slide in the following way:

```
\begin{frame}[label=yourlabel]
```

Afterwards you define the hyperlink to the labelled slide:

```
\hyperlink{yourlabel}{hyperlink text}
```

This method produces a hyperlink which is typeset as regular text. If you want fancier hyperlinks you may use the following modification to the command:

```
\hyperlink{yourlabel}{\beamergotobutton{hyperlink text}}
```

The `\beamergotobutton` command produces an attractive graphic button which is a hyperlink to the labelled slide.¹²

Navigation symbols Our last tip is about changing a small detail in Beamer presentations,¹³ namely the navigation symbols visible in the bottom right corner of each slide. As it happens, we never use them, so it would be nice to get rid of them. This is very simple; it is enough to add the following line to the preamble:

```
\setbeamertemplate{navigation symbols}{}  
-----
```

¹¹ Another solution—perhaps preferable if you have prepared your paper earlier—is to use the `*.bbl` file obtained from compilation of your paper. See <http://tex.stackexchange.com/questions/3542/bibtex-and-beamer>.

¹² You can find tips and tricks for adjusting the button's appearance at <http://tex.stackexchange.com/questions/63171/beamergotobuttons-color>.

¹³ This comes from <http://nickhigham.wordpress.com>.

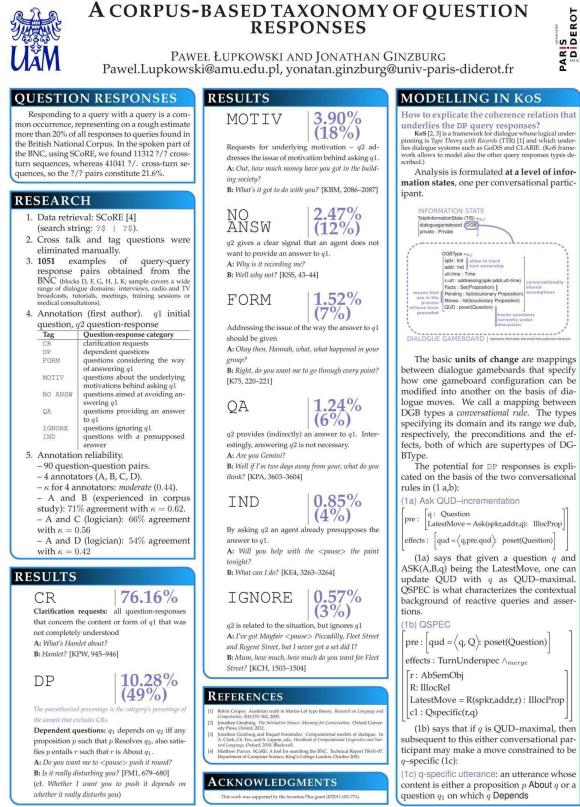


Figure 8: Example poster with the *baposter* class

3 Conference poster

Designing a conference poster is a very demanding task. You have to combine carefully picked and structured content with an attractive visual form (similar to the presentation, but here we have to be even more aware of space limitations). L^AT_EX allows us to prepare a good-looking poster with minimal effort spent on the visual form and allows you to focus on the most important thing—what content it should have.

One basic document class for preparing posters with L^AT_EX is *a0poster*.¹⁴ However, our favoured class for typesetting posters is *baposter*; it is also quite simple, and allows for various modifications of a poster structure and appearance. You will find the manual and examples at the project homepage: <http://www.brian-amberg.de/uni/poster/>. The simplest way to learn how to use this class is to download examples and analyse them. Here we will list some of interesting and useful commands available in this class. An example of a poster prepared with this class is presented in Figure 8.

¹⁴ <http://www.ctan.org/pkg/a0poster>

The class options In the class options you may declare — int. al. — the following:

- page layout: `landscape`, `portrait`;
- paper size: `a0paper`, `a1paper`, `a2paper`, `a3paper`, `a4paper`;
- `showframe`: draw a frame around the page (mainly useful for debugging).

Poster options The main environment for this class is `poster`. The environment has some options available, such as:

- `grid={yes,no}` — turns on/off the visibility of a grid (useful for designing the layout);
- `columns={n}` — number of columns (default 4 in landscape and 3 in portrait format; maximum number is 6);
- `colspacing={length}` — defines the distance between the columns of the poster;
- `background={bgtype}` — defines the background for the poster. The simplest options are `none` for no background and `plain` for a single background colour, which is defined by the option `bgColorOne={pgf-colour-name}`.¹⁵

The content is formatted in columns and placed into attractive boxes. The box is declared with a command:

```
\headerbox{\textit{boxtitle}}{\textit{name}=\textit{boxname}},  
column=0, row=0}{  
  \textit{content}  
}
```

Thus, as you can see, each box has its own label and column indication. You use the labels to place one box under another one, e.g.:

```
\headerbox{RESEARCH}{name=research,below=intro,  
column=0, row=0}{  
  \textit{content}  
}
```

This code will place the box entitled “RESEARCH” in the first poster column under the box labelled as `intro` (see Figure 8).

You can easily modify the appearance of boxes (like headers, borders, shapes and fill types). Try the following poster options:

- `textborder={type}` — type of border for the lower part of the box (possible values: `none`, `bars`, `coils`, `triangles`, `rectangle`, `rounded`, `faded`);
- `headerborder={type}` — which sides of the text box headers should have a border (`none`, `closed`, `open`);

¹⁵ Other types of background are available, such as different gradients; see the class manual for details: http://www.brian-amberg.de/uni/poster/baposter_guide.pdf.

- `headershape=metatype` — type of ornament for the text box headers (`rectangle`, `small-rounded`, `roundedright`, `roundedleft`, `rounded`).

The example poster presented in Figure 8 uses the following options:

- `textborder=roundedleft`
- `headerborder=closed`
- `headershape=roundedright`

Saving space If you want to save some space in your poster you can make lists more compact. To do this just use the command `\compresslist` after the `\begin{enumerate}` or `\begin{itemize}`.

If you are interested in other classes or general guidelines for preparing posters with L^AT_EX, see [3, 4, 10].

Summary

We hope that we have convinced you to use L^AT_EX as a unified environment for preparing your conference materials. We think that the biggest advantage of L^AT_EX is that it allows you to focus on the important thing, namely the quality of the content. It is L^AT_EX that takes care of the form of your materials (and we have to admit that we like the results). However, we should also admit that there are some issues that we sometimes find disturbing. Many L^AT_EX document classes use different approaches to some standard commands (e.g. author, title) or define completely new commands to use, so there are cases when we need to meticulously study guidelines or examples provided by the conference organisers. This—unfortunately—might take some time.

As we have pointed out, we value L^AT_EX for the possibility of easily reformatting the content from paper into a presentation or a poster. Last but not least, sometimes we find it very difficult to fit our content in the page limit using the document class provided by the organisers. However, one may argue that this is to some extent a universal problem.

Acknowledgements This work was supported by funds of the National Science Council, Poland (DEC-2012/04/A/HS1/00715).

References

- [1] David Allen. Screen presentations, manuscripts, and posters from the same L^AT_EX source. *The PracTEX Journal*, 2005. <http://tug.org/pracjourn/2005-1/allen/allen.pdf>.
- [2] Patrick Daly. Graphics and Colour with L^AT_EX. <http://tex.loria.fr/graph-pack/grf/grf.pdf>, 1998.
- [3] Tomas Morales de Luna. Writing posters in L^AT_EX. *The PracTEX Journal*, 2008(3). <http://tug.org/pracjourn/2008-3/morales/>.
- [4] Paulo Rogério de Souza e Silva Filho and Rian Gabriel Santos Pinheiro. Design and Preparation of Effective Scientific Posters using L^AT_EX. *The PracTEX Journal*, 2010(2). <http://tug.org/pracjourn/2010-2/rogerio.html>.
- [5] Johan Engelen. How to Include an SVG Image in L^AT_EX. <http://tug.ctan.org/tex-archive/info/svg-inkscape/InkscapePDFLaTeX.pdf>, 2010.
- [6] Peter Flom. L^AT_EX for academics and researchers who (think they) don't need it. *The PracTEX Journal*, 2005(4). <http://tug.org/pracjourn/2005-4/flom/flom.pdf>.
- [7] Marius Hofert and Markus Kohm. Scientific Presentations with L^AT_EX. *The PracTEX Journal*, 2010(2). <http://tug.org/pracjourn/2010-2/hofert.html>.
- [8] Tobias Kuhn. `bchart`: Simple Bar Charts in L^AT_EX. <http://www.ctan.org/pkg/bchart>, 2012.
- [9] Rouben Rostamian. A Beamer Quickstart. <http://www.math.umbc.edu/~rouben/beamer/quickstart.html>, 2004.
- [10] Han Lin Shang. Writing Posters with Beamerposter Package in L^AT_EX. *The PracTEX Journal*, 2012(1). <http://tug.org/pracjourn/2012-1/shang.html>.

◊ Paweł Łukkowski
Institute of Psychology
Department of Logic and Cognitive Science
Adam Mickiewicz University
Poznań, Poland
`pawel.lukowski (at) gmail dot com`
http://amu.edu.pl/~p_lup/

◊ Mariusz Urbaniński
Institute of Psychology
Department of Logic and Cognitive Science
Adam Mickiewicz University
Poznań, Poland
`Mariusz.Urbanski (at) amu dot edu dot pl`
<http://mu.edu.pl>

LiPPGen: A presentation generator for literate-programming-based teaching

Hans-Georg Eßer

Abstract

Literate programming techniques can be used as a teaching method—not only in book form, but also for lectures in the classroom. I present a tool which helps instructors transform their literate programs into lecture presentations: LiPPGen, the Literate-Programming-based Presentation Generator, takes a standard literate program (with LATEX as the documentation language) as input and lets the instructor comfortably generate presentation slides for each code chunk. It then assembles the provided slide texts and the code chunks and turns them into a browser-based presentation.

LiPPGen offers unique features in comparison to standard presentation programs (such as PowerPoint) in that code chunks may be larger than the space on a slide permits: if so, the code can be scrolled during the presentation. The code is also presented with syntax highlighting using simple regular-expression-based rules. Currently, C and Python are supported.

This article describes both the features and usage of LiPPGen and provides an example, showing a small literate program (the implementation of a component of an educational operating systems) and its transformation into lecture slides.

LiPPGen is available under an open source license so that others who use literate programming in an instructional environment can also use the software and modify it to their needs.

1 Introduction

Literate programming [5] is a programming technique invented by D. E. Knuth which lets developers create source code and documentation in one “literate program” from which both well-readable documentation and compilable source files can be extracted. It can be used to create textbooks on any computer science topic that involves presenting and explaining larger program code blocks. This approach has been used by a few authors, including Knuth himself, when he published the TEX source code [6], but also more recently, for example by Pharr and Humphreys who explain the art of 3D rendering in their book [10]. Additional literate-programming-based textbooks are mentioned in the “Books: Applies Literate Programming” section of the literateprogramming.com link list [7].

One of the advantages of literate programming over other development styles is that the order of pre-

sentation does not depend on syntactical constraints. For a developer this means that the original creative process can be recorded in the literate program, allowing both top-down and bottom-up approaches. Instructors can base the presentation on didactic considerations, for example, they can first give function definitions and structure declarations in an incomplete form (when the audience does not have the required knowledge to understand the full versions) and later extend them when the missing information has been taught.

Since a book is not helpful in a classroom setting, the question arises of how an instructor might create lecture slides from a literate program. While it is possible to copy and paste fragments from the literate program into a presentation and manually add text, such a procedure is tedious and will not always lead to good results. Also, whenever the author modifies the original literate program, he or she must also manually update the slides. Until now, there has been no software to aid the instructor in creating a literate-programming-based presentation.

2 LiPPGen features

LiPPGen [2] lets you select a part (or several parts) of a literate document by marking blocks with `begin` and `end` comments. When running the tool on such a file, it creates an HTML file and opens it in the browser (Figure 1). The page shows code chunks and documentation blocks separately (with most LATEX code either stripped or converted to equivalent HTML), and for each code chunk you can enter some descriptive text in an HTML editor field sitting next to the code chunk. (The program displays the documentation parts as well so that you can easily decide what information to pick for the slide

The screenshot shows a web-based interface for LiPPGen. At the top, a dark header bar displays the text "FAU Chunk: <the program> (1)". Below this, the main content area is divided into two columns. The left column contains a bulleted list of features:

- First of all, the main program file
 - It includes some headers...
 - ... and also the implementation of testdiv3().
- Yes, we can have several levels of indentation...
- ... and there's some syntax highlighting and line wrapping.

The right column displays the C code with syntax highlighting. The code includes a header file inclusion, a function definition for `testdiv3()`, and a main function that reads a number from the user and prints whether it is divisible by 3. The code is annotated with comments: `/* Test program, (c) 2013 The Author */`, `/* header file inclusion */`, `/* test function */`, `/* TANT_NAME 42 */`, and `/* main () { */`.

Figure 1: LiPPGen lets you convert literate programs into presentations, with large parts of the process being fully automatic.

content.) The editor allows for simple markup, such as enumerations, bullet lists, bold, and italics.

When you've entered all the information, you can send the data back to the program (it supplies a simple HTTP server for just this purpose), and then LiPPGen creates an HTML presentation file with the code chunks and your added input. Finally it opens the new presentation in the browser.

Additional features are:

- You can repeat the editing process several times; data entered in a preceding program run remains available.
- The presentation shows chunk names in a way that is similar to the formatting in a traditional literate program (e.g., $\langle name \rangle$), both for the definition (as in $\langle name \rangle \equiv \dots$ and $\langle name \rangle + \equiv \dots$) as well as for occurrences in other code chunks.
- LiPPGen recognizes repeated (i.e., continuing) definitions of code chunks. The first one is always shown as $\langle name \rangle \equiv$, whereas the following ones use $\langle name \rangle + \equiv$. Also, the chunk names are used as slide titles, and if a chunk occurs on several slides, LiPPGen increments a counter.
- Simple syntax highlighting (via regular expressions) is available for C and Python source code, so there's a bit of pretty printing. You can easily extend this to other languages.
- As part of the syntax highlighting LiPPGen also breaks code lines which are too long. The continuation is shown via an arrow character at the end (similar to the display in the XEmacs editor) and there are dots at the beginning of the following line. That way it is clear where a line begins and ends, without a need to introduce line numbers.
- When you give the presentation, you can scroll code chunks up and down using the cursor keys (for code chunks which are longer than the slide permits). Each code chunk "remembers" the current scrolling position, so when you later return to a slide, the display of the code chunk is as it was when you last left the slide.

3 Implementation

When attempting to convert a document which is basically in L^AT_EX syntax—though in its extended Literate Programming form—the natural choice for slide creation would be to stick with L^AT_EX and use one of the available L^AT_EX document classes for presentations, such as `beamer` [13]. However, the end result of any approach using L^AT_EX will be a PDF document, and such documents are static.

In contrast, HTML allows elements on a page to be scrollable, and this feature comes in handy when

we want to show code which exceeds the available space on a slide.

3.1 Recycling: Use what's there

Classically, open source developers are too lazy to reinvent the wheel, and so at the beginning I checked for available tools which might be able to reduce my own development efforts. I found two very helpful programs:

- The “Simple Standards-based Slide Show System” (S5) contains CSS files which let users create complete presentations in single HTML files [8]. Adding a slide is as simple as writing

```
<div class="slide">
<h1>Slide Title</h1>
...
</div>
```

in the source file, and bullet items need no more than standard HTML lists (`...`).

A LiPPGen presentation looks a lot like a standard S5 presentation, except for the added literate programming bits.

- The “NicEdit Inline Content Editor” [4] is a JavaScript program that provides an HTML editor which can be embedded in HTML pages. It is customizable, and for LiPPGen I've disabled most of the available buttons, since they are not needed.

3.2 The power of Python

Python comes with several useful libraries and allows the on-the-fly implementation of a simple web server. (This is true for many other script languages, but I know Python best—the simple reason for choosing it.) We need one for accepting the user's input on the web form, and Python's `socket` module let me integrate the web server into the program.

The complete `lippgen` script is only about 700 lines long, and these few lines of code handle parsing the literate program document, generating the HTML form, accepting the user input, and assembling the final HTML files with syntax highlighting, line breaking and other stuff.

3.3 Some JavaScript as well

In order to allow scrolling of the code chunks, I had to slightly modify the JavaScript code that is part of the S5 system. In brief, I've given HTML names to all code chunks and changed the key-press event code so that [Cursor up] and [Cursor down] scroll the currently displayed code chunk up and down.

I also modified S5's `default/pretty.css` file so that the standard font for listings (`tt`) is `M+ 1m` [9]

since this font runs narrower than the standard Courier type fonts.

4 LiPPGen tutorial

To try out LiPPGen yourself, download the software and install it; then pick a sample literate program and use `lippgen`. We'll describe the process here.

4.1 Installing LiPPGen

Do the following to install the program:

1. Unpack the archive and copy the `lippgen` and `lippgen-sanitize` files to some directory in your path (e.g., `/usr/bin/` or `~/bin`). Create `/usr/share/lippgen/` and recursively copy the `lippgen.d/` directory into that new directory. If you cannot write in `/usr/share` you can pick some other location but will then have to modify the assignment

```
LIPPGEN_D = \
    "/usr/share/lippgen/lippgen.d"
```

in the `lippgen` script.

2. Check if the pre-configured port number 12349 of `lippgen` is free on your machine—if not, change it to something else in the line

```
PORT = 12349
```

Modify the command which opens a URL in a web browser; it is currently set to

```
BROWSER_COMMAND = \
    "open %s -a \"Google Chrome\""
```

which works on a Mac with Google Chrome installed. For Firefox on a Linux machine the proper command would be

```
BROWSER_COMMAND = "firefox -new-tab %s"
```

4.2 Using LiPPGen

Assume you have a literate program in a file named `example.nw` (which is the standard file extension if you use `noweb`). Any other extension except `.html` is fine as well.

1. Mark the relevant part(s) of the literate programming source file by inserting two lines

```
%%% BEGIN LITERATE TEACHING %%%
```

and

```
%%% END LITERATE TEACHING %%%
```

(without any leading spaces) around each part that is to be included in the slides. (You can omit the last end marker; in that case LiPPGen will go on processing until the end of the document.)

2. Run `./lippgen` on the file, e.g. by issuing the command `./lippgen example.nw`; this produces a file `example.form.html` and opens it in the preconfigured browser.

The default language for syntax highlighting is C. If you want Python instead, use `Python` as a second argument to `lippgen`. If you use a different language, modify the program.

3. In the browser, fill in the text input boxes next to the code chunks. You can leave input boxes empty if you want to create slides that only have code on them. Click `Send` at the end of the page.
4. Submitting will transfer the input boxes' contents to the program's built-in server, where the processing continues. Your entries in the fields will also be saved in `example.lip` so that it will be reused if you run `lippgen` on the same file again (the input boxes will already be filled with the entries from the last time). This step creates the final presentation file `example.html` and opens it in the browser.
5. Check the resulting slides and make changes if necessary (going back to step 3).
6. Give the lecture.

4.3 Generating extra pages

It's also possible to create extra presentation pages without code, but LiPPGen has no way of knowing that in advance. After you've initially created the HTML slides, you can edit the HTML file and insert

```
<div class="slide">
<h1>Slide Title</h1>
<ul>
    <li>...</li>
    <li>...</li>
    ...
</ul>
</div>
```

blocks between other `div` elements of class `slide`. This will disrupt the enumeration of slides (and as a consequence scrolling will not work in slides after the first manually included one). Thus, for post-processing of a manually modified HTML file, there's an extra tool called `lippgen-sanitize` that will renumber the slides.

However, modifying the HTML file this way still makes it harder to keep the original literate program and the presentation in sync; when you regenerate the slides with `lippgen`, you lose the slides which you have added manually. Future versions of LiPPGen may improve this procedure.

4.4 Adding keywords

Currently syntax highlighting knows only a few keywords which typically occur in C or Python programs. They are registered in the `C_KEYWORDS` and `P_KEYWORDS` variables, e.g.

```
C_KEYWORDS = ["uint ", "int ", "char ",
  "#define", "#typedef", "struct", "return",
  "#include", "if ", "else "]
```

If you want to add your own keywords to the list (so that LiPPGen will highlight them), just append them to the appropriate variable.

A future version of LiPPGen might use the `highlight` program [12], or similar, to provide better highlighting.

4.5 Publishing a LiPPGen presentation

If you want to publish a LiPPGen presentation on a website, you will need to copy the HTML file and the automatically generated `lippgen.d` subdirectory to the web server. All files in that subdirectory are referenced via relative "`lippgen.d/...`" URLs, so the file should display properly without further ado.

5 An example

I've developed LiPPGen as part of my Ph.D. research which mainly consists of implementing Ulix [3], a new Unix-like operating system using literate programming. To test whether the literate programming approach is helpful in an operating systems class, I'm going to convert parts of the literate program into slides and use them during lectures; that first real-world test is scheduled for the winter term 2013/14 when I'll be giving a course called "Implementing Operating Systems with Literate Programming" at Nuremberg University of Applied Sciences (TH Nürnberg). When the course starts, slides will be available from the course website [1].

Figure 2 shows an excerpt from the signal handling chapter of the unpublished Ulix code which implements the `kill` system call. That part of the book contains four code chunks, and LiPPGen will convert them into four slides which may then be described.

When calling `lippgen`, a browser window displays the generated HTML form, as shown in Figure 3. You can then enter the slide contents and also provide metadata for the title slide (title, author, and organization). After clicking the *Send* button, the browser opens the final presentation file, shown in Figure 4.

6 License

The licensing for LiPPGen may look a little irritating, but since I've used other components, I need to

follow their licenses. Thus, the modified S5 code is in the public domain, the NicEdit component is available under the MIT license, and the Python script `lippgen` is GPLv2-licensed.

To summarize this, you're basically free to do whatever you like with the package. If you modify and re-publish LiPPGen, you just have to be aware of the third party code's licenses.

7 Future work

I've also looked into the alternative presentation tool Prezi [11] which allows zooming into and out of presentation parts. There, the presentation is basically a big mind map.

Since using code chunks is somewhat similar to the Prezi approach, it would be interesting to have a tool which allows quick replacement of a chunk name with the chunk content (when clicking it).

I'm also planning to experiment with the above-mentioned `highlight` program [12] since it makes no sense to invest time into developing a separate highlighting engine when similar code is available.

References

- [1] Hans-Georg Eßer. Implementing Operating Systems With Literate Programming. Lecture slides, Nuremberg University of Applied Sciences, Winter term 2013/14. <http://ohm.hgesser.de/be-ws2013/>.
- [2] Hans-Georg Eßer. LiPPGen. <http://hgesser.de/software/lippgen/>.
- [3] Felix Freiling and Hans-Georg Eßer. ULIx. <http://www.ulixos.org/>.
- [4] Brian Kirchoff. NicEdit Inline Content Editor, 2008. <http://nicedit.com/>.
- [5] Donald E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 1984.
- [6] Donald E. Knuth. *TEX: The Program*. Addison Wesley Publishing Company, 1986.
- [7] Literate programming link list. <http://www.literateprogramming.com/links.html>.
- [8] Eric A. Meyer. S5: A Simple Standards-Based Slide Show System. <http://meyerweb.com/eric/tools/s5/>.
- [9] Coji Morishita. M+ 1M font. <http://mplus-fonts.sourceforge.jp/mplus-outline-fonts/design/index-en.html>.
- [10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

- [11] Prezi website. <http://prezi.com/>.
 - [12] Andre Simon. Highlight Manual. <http://www.andre-simon.de/doku/highlight/en/highlight.html>.
 - [13] Till Tantau et al. Beamer document class for L^AT_EX. <http://ctan.org/pkg/beamer>.

◊ Hans-Georg Eßer
Univ. Erlangen-Nürnberg
Lehrstuhl 1 für Informatik
Martensstraße 3
D-91058 Erlangen, Germany
h.g.esser (at) cs dot fau dot de
<http://hgesser.de/>

323a *<kernel declarations 59b>* +≡
 void kill (int pid, int signo);
Uses kill.

(61a) ◊322c 327a▷

Note that we do no checking in this function, `kill` can be called by the kernel itself (which may send any signal to any process), but it cannot be called directly by a process. Sending by a process requires using a system call, and the system call handler will check whether the process is allowed to send the signal to the target process before calling `kill`.

It is also classical for a process to send a signal to itself; that is what the `raise` function does. We will not implement it specifically inside the kernel, but in the user mode library: `raise(sig)` is the same as `kill(getpid(), sig)`.

Here's the code for the system call handler:

323b *<initialize syscalls 94c>* +≡
 insert_syscall (_NR_kill, syscall_kill);
Uses `insert_syscall` and `syscall_kill`.

(64a) <313b

323c *`<syscall functions 93c>+≡`*
 void syscall_kill (struct regs *r) {
 // ebx: pid of child to send a s signal
 // ecx: signal number
 int ok, retval;
 int target_pid = r->ebx;
 int signo = r->ecx;
 `<check if current process may send a signal 323d>`

(86d) <100f

```
if (ok) {  
    kill (target_pid, signo);  
    retval = 0;  
} else  
    retval = -1;  
  
r->exit = retval;
```

```
    ⟨run scheduler if this was a raise operation 324⟩  
    return;  
};
```

Uses `kill` and `syscall.kill`.

We only allow sending a signal if either the sender's owner has user ID 0 or if sender and recipient have the same owner:

323d *⟨check if current process may send a signal 323d⟩≡*

(323c)

Figure 2: This excerpt from the literate program “Ulix” contains four code chunks.

LiPPGen

The Literate-Programming-based Presentation Generator

Version 1.0, (c) 2013 Hans-Georg Eßer

Title: The KILL Syscall Handler
 Author: Hans-Georg Eßer
 Organisation: FAU Erlangen-Nürnberg

TEXT	CODE
<ul style="list-style-type: none"> • The kill() function is used by the kernel only • Processes must make a system call to access kill() 	<pre><kernel declarations>≡ void kill (int pid, int signo);</pre>
Note that we do no checking in this function, kill can be called by the kernel itself (which may send any signal to any process), but it cannot be called directly by a process. Sending by a process requires using a system call, and the system call handler will check whether the process is allowed to send the signal to the target process before calling kill. It is also classical for a process to send a signal to itself; that is what the raise function does. We will not implement it specifically inside the kernel, but in the user mode library: raise(sig) is the same as kill(getpid(), sig). Here's the code for the system call handler:	<pre><initialize syscalls>≡ insert_syscall (__NR_kill, syscall_kill);</pre>
<p>The system call handler for kill</p> <ul style="list-style-type: none"> • checks if sending the signal is allowed • and if so, it calls kill() 	<pre><syscall functions>≡ void syscall_kill (struct regs *r) { // ebx: pid of child to send a s signal // ecx: signal number int ok, retval; int target_pid = r->ebx; int signo = r->ecx; <check if current process may send a signal> if (ok) { kill (target_pid, signo); retval = 0; } else retval = -1; r->eax = retval;</pre>

Figure 3: The HTML form lets you enter content for the slides.

FAU Chunk: <syscall functions> (1)

The system call handler for kill

- checks if sending the signal is allowed
- and if so, it calls kill()

```
<syscall functions>≡
void syscall_kill (struct regs *r) {
    // ebx: pid of child to send a s signal
    // ecx: signal number
    int ok, retval;
    int target_pid = r->ebx;
    int signo      = r->ecx;
    <check if current process may send a signal>

    if (ok) {
        kill (target_pid, signo);
        retval = 0;
    } else
        retval = -1;

    r->eax = retval;

    <run scheduler if this was a raise operation>
    return;
};
```

Figure 4: The HTML presentation file created by LiPPGen. The listing on the right hand side is scrollable.

Entry-level MetaPost 2: Move it!

Mari Voipio

This installment introduces some of the basic commands for moving a line or an object—i.e. a *path*—to a different position: shifting, rotating, reflecting, repeating. In programs with a graphical user interface, these operations are typically done by clicking and dragging or clicking and selecting a command on a toolbar.

MetaPost has a slightly different approach to e.g. rotation and this can be confusing at first, although it is completely logical on its own terms. It adds to the confusion that some commonly used commands like flip and duplicate do not exist in MetaPost (nor MetaPost manuals), although the operations are doable once you know what to look for.

For basic information on running MetaPost, either standalone or within a ConTEXt document, see <http://tug.org/metapost/runningmp.html>.

1 Store it first

Before we start to manipulate a path, we typically store it for further access by defining a *path variable*. In many MetaPost tutorials you see paths named *p*, *q* and *r*, but I prefer slightly longer and more descriptive names, even though that means more typing. Below we first define and then draw a diamond that is used for many examples in this tutorial.

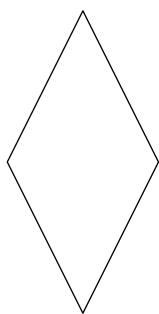
```
beginfig (1) ;
numeric u ; u := 1cm ; % define the unit

% define path variable "dmnd" (diamond shape,
% intentionally asymmetrical)
path dmnd ;
dmnd := (1u,0u) -- (0u,2u) -- (1u,4u) -- (2u,2u)
    -- cycle ;

% drawing diamond (outline)
draw dmnd ;

endfig ;
end .
```

Here is the output:



Troubleshooting: If your file compiles but the graphic is empty, you probably forgot to draw at least one path, i.e. the output “paper” is still empty. No draw/fill/filldraw command at all leads to an empty file.

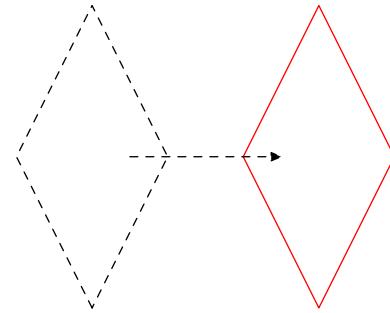
When multiple path variables are defined, they can all go at the top of the file to make sure that we define each variable before trying to use it. However, we can use the variables in any order after that and as many times as is needed. Personally I like to list my variables in alphabetic order so I can find one quickly if I need to check or change the path definition.

2 Shift (copy, duplicate)

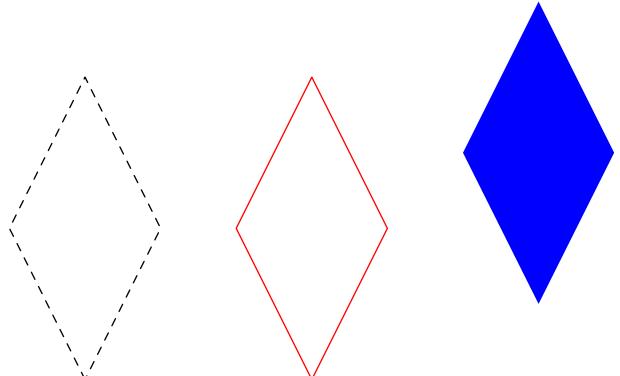
To shift means to move, and that is exactly what the command does. It works in a fairly intuitive way:

```
draw dmnd shifted (3u,0u) withcolor red;
```

That line can be read as “take a diamond, draw it to (3u,0u) using a red pen”. Visually:



If we draw the original diamond as well as the shifted one, we now have two diamonds, i.e. we have copied an object. We can change the attributes of the second (shifted) diamond, e.g. to make a coloured diamond by using the `fill` command. For example:



Here's the MetaPost code. The beginning is what we saw in the first section.

```
beginfig (1) ;
numeric u ; u := 1cm ; % define the unit

% define path variable "dmnd" (diamond shape)
path dmnd;
```

```

dmnd := (1u,0u) -- (0u,2u) -- (1u,4u)--(2u,2u)
      -- cycle;

% draw dashed diamond at original location
draw dmnd dashed evenly ;

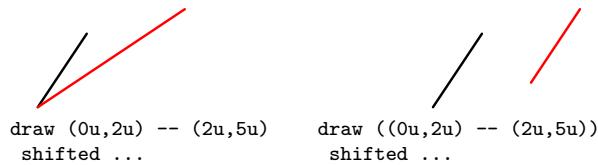
% draw second diamond to the right, in red
draw dmnd shifted (3u,0u) withcolor red ;

% draw third diamond, filled with blue,
% to the right and up from original
fill dmnd shifted (6u,1u) withcolor blue ;

endfig ;
end .

```

The `shift` command only applies to the element preceding it; we need to use parentheses if we intend otherwise. Thus `draw (0u,2u) -- (2u,5u) shifted (4u,1u)` and `draw ((0u,2u) -- (2u,5u)) shifted (4u,1u)` produce very different results:



The black is the original `(0u,2u)--(2u,5u)` line, while the red is the result of the whole expression, including the shift. Here is the code (combined for exposition):

```

...
% set the penwidth (see previous article)
drawoptions (withpen pencircle scaled 1/10u) ;

% draw original line in black
draw (0u,2u) -- (2u,5u) ;

% draw red line with shift of endpoint only:
% (first example)
draw (0u,2u) -- (2u,5u) shifted (4u,1u)
      withcolor red ;
% ... or ...
% draw red line with whole line being shifted:
% (second example)
draw ((0u,2u) -- (2u,5u)) shifted (4u,1u)
      withcolor red ;
...

```

3 Rotate

Rotation is another basic graphical transformation. In MetaPost, the basic operation is done with the keyword `rotated`. We also always need to specify the angle of rotation. However, if one is used to a graphical program (say, Inkscape), the results of `rotated` can be a bit of a surprise at first. Let's look at an example.

```

% define a path variable "tetris"
path tetris ;
tetris := (3u,2u) -- (4u,2u) -- (4u,5u)--(2u,5u)
      -- (2u,4u) -- (3u,4u) -- cycle ;

```

```
% draw solid blue tetris block
fill tetris withcolor blue ;
```

```
% draw rotated tetris block in red
fill tetris rotated 90 withcolor red ;
```

And the output (scaled down, here and in the following, from *TUGboat*'s narrow columns):



Say what?

The logic becomes more apparent if we add a small dot at origin (0,0):

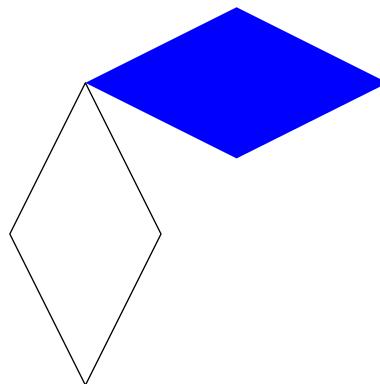
```
% mark origin with a black dot
fill fullcircle scaled 1/10u ;
yielding:
```



The lesson: **The rotated command rotates the path's bounding box around the origin (0,0), and rotation direction is counterclockwise.**

If we want to rotate the path around any other point, we have to use the command `rotatedaround`, for which we need to specify both the location of the rotation point and the angle of rotation. Example:

```
% draw a diamond rotated around its top point,
% at (1u,4u)
fill dmnd rotatedaround ((1u,4u),90)
      withcolor blue;
```

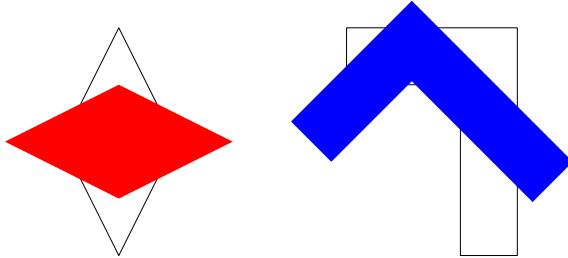


To rotate around the “midpoint” of the object (that is, the center of bounding box, the default rotation point in many programs), we don’t need to painfully compute the coordinates for the center. MetaPost has a handy keyword `center` for that:

```
% draw diamond outline
draw dmnd ;
% draw red diamond rotated around its center
fill dmnd rotatedarround (center dmnd,45)
    withcolor red;

% draw tetris outline
draw tetris;
% draw tetris block rotated around its center
fill tetris rotatedarround (center tetris, 90)
    withcolor blue;

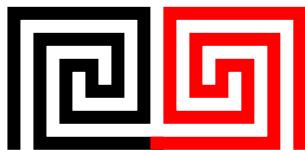
% these commands would make the centers visible:
%fill fullcircle scaled 1/10u shifted
%              (center dmnd) ;
%fill fullcircle scaled 1/10u shifted
%              (center tetris) ;
```



4 Reflect (flip, mirror)

Another command that may seem to be missing in MetaPost is horizontal or vertical mirroring (also known as flipping). The functionality does exist, invoked with the keyword `reflectedabout`, although a bit of practice is needed to get used to it—but on the other hand we can specify any straight line to be the reflection axis, it doesn’t have to be horizontal or vertical. Consequently, to use `reflectedabout`, we must specify *two* points for the reflection axis. If you find this hard, think of a mirror and where you’d place its edge to get the reflection needed.

Here I’m playing around with a Greek key pattern and its reflection (yes, they do overlap in the middle) around a vertical line.



And the code to produce it:

```
beginfig (2) ;

% design source:
% http://gwydir.demon.co.uk/jo/greekkey/turns.htm
numeric u ; u := 3.8mm ; % define the unit

% creating sharp squared joins
linecap := squared ;
linejoin := mitered ;

% set the penwidth
drawoptions (withpen pencircle scaled 1/2u) ;

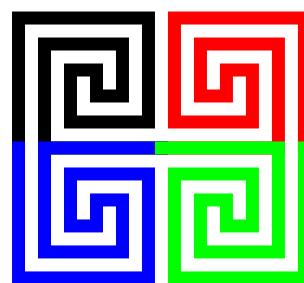
% define the path for the greek key
path gkey;
gkey :=  (origin) -- (0u,5u) -- (5u,5u) -- (5u,1u)
        -- (2u,1u) -- (2u,3u) -- (3u,3u) -- (3u,2u)
        -- (4u,2u) -- (4u,4u) -- (1u,4u) -- (1u,0u)
        -- (5.5u,0u);

% draw it
draw gkey ;

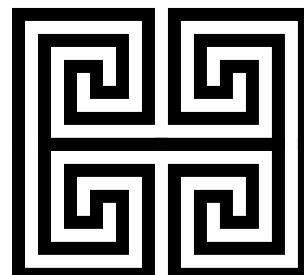
% flip it and draw it in red
draw gkey reflectedabout ((5.5u,0u),(5.5u,5u))
    withcolor red ;
endfig ;
end .
```

By adding another, horizontally flipped, key and then a key with rotation we can create a square Greek key pattern variation:

```
draw gkey reflectedabout ((origin),(5.5u,0u))
    withcolor blue ;
draw gkey rotatedarround (lrcorner gkey,180)
    withcolor green ;
```



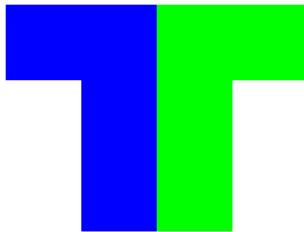
The pattern is more apparent entirely in black:



If you want to flip an object along a side of the bounding box, the MetaFun package provides a set of handy shortcuts: the corners of the bounding box are called `llcorner`, `lrcorner`, `ulcorner` and `urcorner`. Thus, to flip an object along the right edge of the bounding box, the lower right and upper right corner are called for:

```
% drawing flipped tetris block
fill tetris reflectedabout (lrcorner tetris,
                           urcorner tetris)
withcolor green ;
```

Yielding:



5 ... and repeat

If you need to repeat the same pattern at regular intervals, a combination of shift and loop is possible. Besides the angular variety above, I've also designed a rounded version of the Greek key that could e.g. make a nice header for a book. To alter the size of the "frieze" I can either change the number of repetitions or the final size, depending on what shape is desired.

```
% define one spiral
path spiral;
spiral := (0,7/4) .. (2,4) .. (5,2) .. (3,0)
.. (2,2) .. (4,2) .. (3,1) ;

% repeat to get 10 spirals in a row
for i = 0 step 5 until 45 :
  draw spiral shifted (i,0) ;
endfor ;

% add a bit of white around the pattern
setbounds currentpicture
  to boundingbox currentpicture
  enlarged 1/4 ;

% resize the whole thing
currentpicture := currentpicture xsized 7cm ;
```

And the output:



6 MetaFun

The `xsized` command used in the last line, like the `corner` shortcuts mentioned above, is part of the MetaFun package, not MetaPost proper. MetaFun is loaded in ConTeXt by default, but needs to be explicitly loaded when using standalone plain MetaPost documents, like this:

```
mpost --mem=metafun yourfile.mp
```

See <http://wiki.contextgarden.net/MetaFun> for more.

◊ Mari Voipio
 mari dot voipio (at) lucet dot fi
<http://www.lucet.fi>

Creating Tufte-style bar charts and scatterplots using PGFPlots

Juernjakob Dugge

Abstract

In this article I describe how to use PGFPlots to create bar charts and scatterplots in the style described by Edward Tufte in *The Visual Display of Quantitative Information* [1].

I demonstrate how to implement *range frames*, which are axis lines drawn only over the range of the data points, and *dot-dash plots*, which are scatterplots with tick marks representing the marginal distribution of the data.

1 Bar chart

(I adapted this section from my [latex-community.org](https://tex.stackexchange.com/questions/437/tufte-charts) post on this topic, namely [latex-community.org/know-how/437-tufte-charts](https://tex.stackexchange.com/questions/437/tufte-charts).)

Assume we have numerical data in a data file called `dataA.csv` that looks like this:

```
1, 8.5
2, 12
3, 6.5
4, 7
5, 3
6, 17.5
7, 13
8, 8.5
9, 6
10, 11
11, 5
12, 10
```

Creating a bar chart of this data using PGFPlots is simple. All we have to do is put the following code at the point where we want the chart to appear:

```
\begin{tikzpicture}
\begin{axis}[ybar]
\addplot table [col sep=comma] {dataA.csv};
\end{axis}
\end{tikzpicture}
```

The result is shown in Figure 1.

1.1 Bar colour

In his book, Tufte uses a particular shade of yellowish gray for the bars. Custom colours for use in PGFPlots can be defined using the `\definecolor` macro provided by the `xcolor` package, which is automatically loaded by PGFPlots. The colour used in Tufte's book can be made available using.

```
\definecolor{tufte1}{rgb}{0.7,0.7,0.55}
```

The most straightforward way to fill the bars of the plot using this colour and to disable the outlines of the bars is to add the keys `fill=tufte1`,

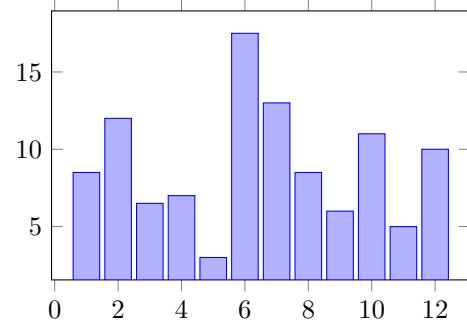


Figure 1: Bar chart using the default settings.

`draw=none` to the optional argument of the `\addplot` command. Alternatively, we can set a `cycle list` consisting of these options:

```
cycle list={
  fill=tufte1, draw=none\\
}
```

That way, no options have to be supplied to the `\addplot` command.

1.2 Bar width

By default, the bars in PGFPlots are thicker than those in Tufte's plot. The width of the bars is controlled using the `bar width` key, which can be specified in absolute units (like `5mm`) or, since PGFPlots version 1.7, in terms of axis units. The latter requires setting the key `compat=1.7` or higher.

Using axis units to specify the bar width has the advantage that the widths of the bars will shrink as more data points are added, avoiding overlap between the bars. Sometimes, though, it might be desirable to keep the widths of the bars constant and instead increase the overall width of the plot as the number of data points increases. This can be achieved by using an absolute width for the bars, and specifying the length of the x unit vector in terms of the bar width:

```
bar width=2mm,
x=1.5*\pgfkeysvalueof{/pgfplots/bar width}
```

would make the bars 2 mm wide with a 1 mm gap between neighbouring bars, and the axis would grow or shrink horizontally to fit all the bars.

1.3 Grid lines

In Tufte's plot, there are gaps in the bars at regular intervals instead of conventional grid lines in the background of the bars. This can be simulated in PGFPlots by placing white horizontal grid lines on top of the bars by setting

```
ymajorgrids,
grid style=white,
axis on top
```

1.4 Axes

In Tufte's plot, the y axis is not drawn. In PGFPlots, an axis can be hidden using the key `hide y axis`. However, this key also deactivates the tick labels. In our case, since we only want to make the axis line invisible, we can instead set its opacity to zero using `y axis line style={opacity=0}`.

The x axis line in Tufte's plot is aligned with the first and last bar. This means that the x axis needs to run from the left edge of the first bar to the right edge of the last bar. In PGFPlots, this can be achieved by setting the padding of the x axis to half the bar width using

```
enlarge x limits={  
    abs=0.5*\pgfkeysvalueof{/pgf/bar width}  
}
```

where `abs` indicates that the padding is specified in terms of absolute units and not in axis units.

Since we only want an axis line at the bottom of the plot and not on the top, we set

```
axis x line*=bottom
```

By using the starred version of the key, no arrow tip is added to the axis line.

There are no tick labels on the x axis in Tufte's plot. In most real applications this would not be recommended, but in order to recreate Tufte's plot as faithfully as possible, let's go ahead and switch the labels off using `xtick=\emptyset`.

The y tick labels are expressed as percentages. We can specify how to print the tick labels in PGFPlots using the `yticklabel` key:

```
yticklabel=\pgfmathprintnumber{\tick}\%,\%
```

The code passed to `yticklabel` is executed for every tick label. The `\tick` macro contains the current tick value, which is printed in a consistent format by `\pgfmathprintnumber`. After the tick value, we add a thin space (`\,`) and the percent sign, which has to be escaped using a backslash to distinguish it from its use as the comment character.

Note that there is also a `yticklabels` key (with a trailing `s`). This is used to provide a comma-separated *list* of tick labels, whereas the `yticklabel` key is used to provide a *pattern* for printing the labels, typically based on the tick value.

Finally, we can switch off the tick marks with `major tick length=0pt`

1.5 Creating an axis style

There are different ways of activating all these options. The keys can be directly added to the optional argument of an `axis` environment, in which case they only apply to that axis.

They can also be activated globally using

```
\pgfplotsset{  
    <keys>  
}
```

which applies the keys to all `axis` environments that follow.

Lastly, we can create a new PGF `style` that acts as a container for the keys, and apply that new `style` to the `axis` environment:

```
\pgfplotsset{  
    tufte bar/.style={  
        <key1>,  
        <key2>,  
        ...  
    }  
}  
\begin{axis}tufte bar
```

Grouping options using `styles` is a very useful technique, as it helps to keep the code readable and maintainable.

Putting all the keys described above into a style and applying that style to the axis in the first listing results in the plot shown in Figure 2.

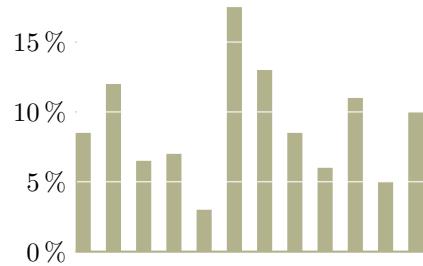


Figure 2: Bar chart in the style of Edward Tufte

2 Scatterplot

A scatterplot can be created in PGFPlots just as easily as a bar chart. A simple plot of some random data points like the one shown in Figure 3 can be created using

```
\begin{tikzpicture}  
    \begin{axis}[  
        only marks,  
        domain=1:10  
    ]  
        \addplot  
            ({cos(rnd *x+rnd)}, {((rnd+1)*x+rnd)});  
    \end{axis}  
\end{tikzpicture}
```

The key `only marks` instructs PGFPlots not to draw connecting lines between the data points.

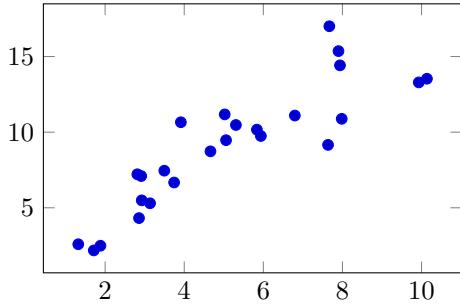


Figure 3: Default scatterplot

2.1 Basic scatterplot adjustments

With only a few options, we can already get quite close to the scatterplots in Tufte's book. The plot marks are black and slightly smaller than in the default plot, so we set

```
cycle list={black},
mark size=1.5pt
```

(the default mark size is 2 pt).

The remaining adjustments are related to the plot axes.

2.2 Range frames

Instead of conventional axis lines, Tufte uses so-called *range frames*: The axis lines are not drawn over the entire extent of the plot, but only between the levels of the lowest and highest data points.

One way of creating range frames in PGFPlots is by erasing the standard axis lines and drawing lines of the required lengths using `\draw` commands. For this, we can make use of the fact that PGFPlots stores the lowest and highest data coordinate values in internal macros. These macros, called

```
\pgfplots@data@xmin
\pgfplots@data@xmax
...
```

can be made accessible by putting

```
\makeatletter
\let\pgfplotsdataxmin=\pgfplots@data@xmin
\let\pgfplotsdataxmax=\pgfplots@data@xmax
\let\pgfplotsdataymin=\pgfplots@data@ymin
\let\pgfplotsdataymax=\pgfplots@data@ymax
\makeatother
```

before the specification of the `\draw` command. Then the values can be referred to as `\pgfplotsdataxmin`, `\pgfplotsdataxmax`, and so on.

To draw the lines, we can use

```
\draw ({rel axis cs:0,0}
      -|{axis cs:\pgfplotsdataxmin,0})
      -- ({rel axis cs:0,0}
            -|{axis cs:\pgfplotsdataxmax,0});
```

This might look a bit intimidating at first, so let's go through it step by step. The basic command is `\draw (A) -- (B);`, which simply draws a straight line between points A and B.

In this case, A is defined as

```
{rel axis cs:0,0}
-|{axis cs:\pgfplotsdataxmin,0})
```

This is a coordinate specification of the type (C–D), which describes the point located at the intersection of a horizontal line through C and a vertical line through D.

Here, C is `rel axis cs:0,0`, which is the point in the lower left corner of the axis, and D is `axis cs:\pgfplotsdataxmin,0` which is the point with an x component equal to that of the leftmost data point and a y component of zero.

If the x axis was at $y = 0$, we could simply use

```
\draw (axis cs:\pgfplotsdataxmin,0)
      -- (axis cs:\pgfplotsdataxmax,0);
```

Since that is not necessarily the case, though, we'll have to make use of the more complicated expression.

To automatically execute the `\draw` command at the end of the plot, we pass it to the axis like this:

```
after end axis/.code={
  \draw ({rel axis cs:0,0}
        -|{axis cs:\pgfplotsdataxmin,0})
        -- ({rel axis cs:0,0}
              -|{axis cs:\pgfplotsdataxmax,0});
  \draw ({rel axis cs:0,0}
        |-{axis cs:0,\pgfplotsdataymin})
        -- ({rel axis cs:0,0}
              |-{axis cs:0,\pgfplotsdataymax});
}
```

To complete the plot style, only a couple of minor adjustments are needed.

Erase the default axis lines using

```
axis line style={opacity=0}
```

Only show the tick marks on the left and bottom edge of the plot by setting

```
tick pos=left
```

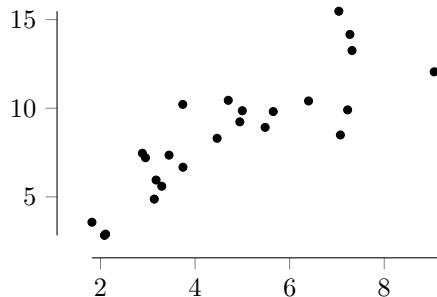
And finally, align the tick marks on the outside of the plot area using

```
tick align=outside
```

Wrapping all these keys in a new style and applying that style to the plot shown in Figure 3 results in the plot shown in Figure 4.

2.3 Dot-dash plot

Another technique used by Edward Tufte is the *dot-dash plot*, which is a combination of a conventional

**Figure 4:** Scatterplot with range-frame

scatterplot (the dots) and plots of the marginal distributions of the data points in the form of short lines along the outside of the plot (the dashes).

This can be implemented in PGFPlots surprisingly easily. By default, the tick marks are placed at regular intervals along the axis. By specifying

```
x tick = data,
y tick = data
```

tick marks are placed at the data points' locations.

Removing the axis lines and tick labels, and making the tick marks black and a bit longer is all it takes to create a basic dot-dash plot:

```
axis line style = { opacity = 0 },
x tick labels = {},
y tick labels = {},
tick style = black
```

To aid the viewer in reading the plot, we can label the first and last of the tick marks. We can do this by again using our macros that store the limits of the data. PGFPlots makes it possible to highlight some tick positions by placing additional tick marks that can be specified using

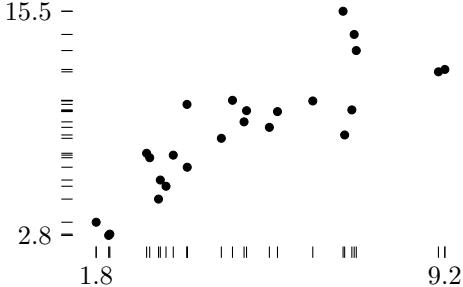
```
extra x ticks = {
  \pgfplots data x min,
  \pgfplots data x max
}
```

These extra ticks can be formatted independently from the standard ticks by specifying the required keys in

```
extra tick style = {
  <options>
}
```

In this case, we need to reactivate the tick labels for the extra ticks. By setting

```
extra tick style = {
  x tick label = { \pgfmath print number [
    fixed,
    fixed zerofill,
    precision = 1
  ] \{ \ tick \} },
  ...
```

**Figure 5:** Dot-dash plot

```
y tick label = { \pgfmath print number [
  fixed,
  fixed zerofill,
  precision = 1
] \{ \ tick \} },
scaled ticks = false
}
```

the values of the outer ticks will be printed as fixed point numbers rounded to one decimal digit. Setting `scaled ticks=false` is necessary when the number formatting style is explicitly set to `fixed`. Otherwise, PGFPlots would print a separate scaling factor when the axis contains very large numbers.

Applying all these options to the plot results in the plot shown in Figure 5.

3 Conclusion

In this article, I aimed to demonstrate the flexibility of PGFPlots by recreating plots from Edward Tufte's *The Visual Display of Quantitative Information*.

While some of the techniques used in this article used internal PGFPlots macros, no alteration of the code was necessary to implement reasonably advanced features like range frames or dot-dash plots.

I hope that this article will succeed in encouraging some of the readers to try their hand at more intricate plot customisations of their own. Once the necessary options and values have been found, the `style` feature of the `pgfkeys` key-value framework used by PGFPlots makes it very easy to reuse plot styles. This helps in creating plots with a consistent appearance with very little effort.

Acknowledgments

I would like to thank Dr. Christian Feuersänger for his very helpful review of this manuscript, and for his continued excellent work on PGFPlots.

References

- [1] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, Cheshire, Conn, 2nd edition, 2001.

Appendix: Complete styles

Make the extreme values available and define the colour:

```
\makeatletter
\let\pgfplotsdataxmin=\pgfplots@data@xmin
\let\pgfplotsdataxmax=\pgfplots@data@xmax
\let\pgfplotsdataymin=\pgfplots@data@ymin
\let\pgfplotsdataymax=\pgfplots@data@ymax
\makeatother

\definecolor{tufte1}{rgb}{0.7,0.7,0.55}
```

3.1 Bar chart

```
\pgfplotsset{
    tufte bar/.style={
        ybar,
        axis line style={draw opacity=0},
        xtick=\empty,
        ymin=0,
        bar width=2mm,
        x=2*\pgfkeysvalueof{/pgf/bar width},
        ymajorgrids,
        grid style=white,
        axis on top,
        major tick length=0pt,
        cycle list={
            fill=tufte1, draw=none\\
        },
        enlarge x limits={
            abs=0.5*\pgfkeysvalueoff{/pgf/bar width}
        },
        axis x line*=bottom,
        x axis line style={
            draw opacity=1,
            tufte1,
            thick
        },
        yticklabel=\pgfmathprintnumber{\tick}\,\%
    }
}
```

3.2 Basic scatterplot

```
\pgfplotsset{
    tufte scatter/.style={
        only marks,
        cycle list={black, gray!50},
        axis lines*=left,
        mark size=1.5
    }
}
```

3.3 Range frame

```
\pgfplotsset{
    range frame/.style={
        tick align=outside,
        axis line style={opacity=0},
        after end axis/.code={
            \draw ({rel axis cs:0,0}
                -|{axis cs:\pgfplotsdataxmin,0})
            -- ({rel axis cs:0,0}
                -|{axis cs:\pgfplotsdataxmax,0});
        \draw ({rel axis cs:0,0}
            |-{axis cs:0,\pgfplotsdataymin})
            -- ({rel axis cs:0,0}
            |-{axis cs:0,\pgfplotsdataymax});
    }
}
```

3.4 Dot-dash plot

```
\pgfplotsset{
    dot dash plot/.style={
        tufte scatter
        axis line style={opacity=0},
        tick style={thin, black},
        major tick length=0.15cm,
        xtick=data,
        xticklabels={},
        ytick=data,
        yticklabels={},
        extra x ticks={
            \pgfplotsdataxmin,
            \pgfplotsdataxmax
        },
        extra y ticks={
            \pgfplotsdataymin,
            \pgfplotsdataymax
        },
        extra tick style={
            xticklabel={\pgfmathprintnumber[
                fixed,
                fixed zerofill,
                precision=1
            ]{\tick}},
            yticklabel={\pgfmathprintnumber[
                fixed,
                fixed zerofill,
                precision=1
            ]{\tick}}
        }
    }
}
```

◇ Juernjakob Dugge
 Schwaerzlocher Str. 64
 72070 Tuebingen
 Germany
 juernjakob (at) dugge dot de

**Typographers, programmers and mathematicians,
or the case of an aesthetically pleasing interpolation**

Bogusław Jackowski

Abstract

The reason for preparing this report is that the author has been convinced for many years that John D. Hobby's algorithm for connecting Bézier segments, implemented by Donald E. Knuth in METAFONT and later transferred by Hobby to METAPOST, based on the notion of a “mock curvature”, is a genuine pearl which deserves both proper acknowledgement and a far wider awareness of its existence. Of course, one can find nearly all the necessary details in the relevant papers by Hobby and in the METAFONT source, but, needless to say, it is not easy to dig through the publications. The present paper provides a full mathematical description of Hobby's interpolation algorithm, discusses its advantages and disadvantages (in particular, its instability) and compares Hobby's approach with a few selected simpler approaches.

1 Introduction

The most popular curves in computer graphics applications are based on polynomials of degrees 2 and 3. These include Bézier curves and B-splines of orders 2 and 3. Bernstein polynomials (cf. [3], [6, p. 14] and [8]) allow Bézier curves to be generalized to degree n as defined by the formula

$$\mathbf{B}(t) \stackrel{\text{def}}{=} \sum_{i=0}^n \binom{n}{i} t^i (1-t)^{n-i} \mathbf{B}_i, \quad (1)$$

where \mathbf{B}_i , $i = 0, 1, \dots, n$ are points in a k -dimensional space.

In practical applications, $k = 2$ or $k = 3$. The present paper discusses the case $k = 2$, $n = 3$, i.e., planar Bézier curves; such curves are used in the PostScript language (including PostScript fonts), in page description formats such as SVG or PDF and in graphic design programs such as CorelDRAW, Adobe Illustrator and the (free software) Inkscape program. In such applications, the main problem is interpolation, i.e., connecting Bézier curves in an aesthetically pleasing manner. A single Bézier arc is capable of expressing relatively few shapes, whereas combining multiple Bézier arcs provides more possibilities.

It might seem that discussing the notion of “aesthetic pleasingness” or “beauty” with respect to a strictly mathematical problem makes no sense. However, the physiology of the human eye provides a valuable insight: the eye surprisingly easily identifies straight lines and circles in a muddle of lines (cf. figure 1). Therefore, it seems reasonable to assume that perhaps for this reason we prefer curves changing the direction uniformly (circles) or not changing the direction at all (straight lines).

Mathematicians use the notion of curvature to investigate changes of curve direction.

A method for connecting Bézier segments that minimizes curvature change has been proposed by J. R. Manning [7]. It turns out, however, that preserving exact curvature at junction points is computationally complex and not necessarily needed. J. D. Hobby's paper [2] presents an interesting solution, substantially reducing the computational complexity of this process due to a smart curvature approximation. The solution has been implemented by D. E. Knuth in the METAFONT program [6] which is basically meant for creating fonts. The same algorithm has been transferred later by Hobby to METAPOST (a modification of METAFONT which generates PostScript output).

This paper aims at investigating Hobby's method, discusses its advantages and disadvantages, and compares it with a few other interpolation methods. Hobby's approach presents an interesting case which illustrates how important a mathematician can be as a link between

This paper was presented at the BachoTeX 2010 conference, and originally published in Polish in *Acta Poligraphica* 1/2013. Translated and reprinted by permission.

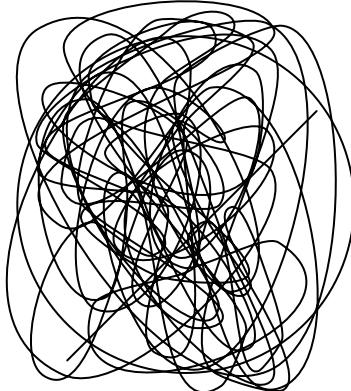


Figure 1: Exercise: find the segment of a straight line and a circle.

an artist, e.g., a typographer who creates fonts, and a programmer, who prepares relevant computer tools. A mathematician creates reality models (here: “smoothness” models) and equips programmers with a theoretical basis for creating appropriate tools.

It is assumed that readers have a basic knowledge of mathematics, for example, they are aware that the derivative of a function $f(t)$ at a point t has something to do with a limit $\lim_{\Delta t \rightarrow 0} (f(t + \Delta t) - f(t))/\Delta t$. On the other hand, although professional mathematicians would find them unnecessary, the details of the derivation of almost all formulas have been presented here, to minimize the prerequisite mathematics. Overall, the author finds Hobby’s idea for connecting Bézier curves a real gem that deserves a careful and more widely accessible description than the ones presented in [2, 5].

2 Determinants

Let us start by recalling the notion of a determinant: a second order determinant, i.e., the determinant for a pair of planar vectors, $\mathbf{v}_1 = (x_1, y_1)$ and $\mathbf{v}_2 = (x_2, y_2)$, is defined as

$$\det(\mathbf{v}_1, \mathbf{v}_2) \equiv \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} \stackrel{\text{def}}{=} x_1 y_2 - x_2 y_1. \quad (2)$$

The following determinant property will be useful later in this paper: it is easy to check with direct calculations for planar determinants that

$$\det(\mathbf{v}_1 + p\mathbf{v}_2, \mathbf{v}_2) = \det(\mathbf{v}_1, \mathbf{v}_2 + q\mathbf{v}_1) = \det(\mathbf{v}_1, \mathbf{v}_2), \quad (3)$$

where p and q are arbitrary real numbers. The geometric interpretation of a determinant will be useful as well: the absolute value of a planar determinant represents the area of the parallelogram spanned on the vectors in question and its sign depends on the orientation of the vectors—for the counterclockwise orientation of the vectors \mathbf{v}_1 and \mathbf{v}_2 the respective determinant is positive.

A small digression: the above interpretation is, in general, so important that the outstanding Russian mathematician Vladimir I. Arnold in his article on the teaching of mathematics [1] emphasizes: *The determinant of a matrix is an (oriented) volume of the parallelepiped whose edges are its columns. If the students are told this secret (which is carefully hidden in a pure algebraic education), then the whole theory of determinants becomes easy to understand. If determinants are defined otherwise, then any sensible person will forever hate all determinants.*

3 Differentiating Bernstein polynomials

The derivative of a parametrically defined curve is determined by differentiating components of the vector defining the curve. Let us imagine a vehicle (point) whose location at time t is defined by $(x(t), y(t))$; then $(x'(t), y'(t))$ simply defines the direction and value of the velocity of this vehicle and $(x''(t), y''(t))$ defines the direction and value of vehicle’s acceleration.

Let us now return to formula (1) in its general form in order to determine the value of the first and second derivative of the function $\mathbf{B}(t)$ at points $t = 0$ and $t = 1$, i.e., at the nodes

$\mathbf{B}(0) = \mathbf{B}_0$ and $\mathbf{B}(1) = \mathbf{B}_n$. Differentiating the polynomial once and twice produces results of the form

$$\begin{aligned}\mathbf{B}'(t) &= n(1-t)^{n-1}(\mathbf{B}_1 - \mathbf{B}_0) + \mathbf{V}(t), \\ \mathbf{B}''(t) &= (n-1)n(1-t)^{n-2}((\mathbf{B}_0 - \mathbf{B}_1) + (\mathbf{B}_2 - \mathbf{B}_1)) + \mathbf{W}(t),\end{aligned}\quad (4)$$

where $\mathbf{V}(t)$ and $\mathbf{W}(t)$ are certain polynomials (the exact formulas are unimportant here) such that $\mathbf{V}(0) = \mathbf{W}(0) = \mathbf{V}(1) = \mathbf{W}(1) = 0$, hence

$$\mathbf{B}'(0) = n(\mathbf{B}_1 - \mathbf{B}_0), \quad \mathbf{B}''(0) = (n-1)n((\mathbf{B}_0 - \mathbf{B}_1) + (\mathbf{B}_2 - \mathbf{B}_1)). \quad (5)$$

For $t = 1$, similar relations can be found:

$$\mathbf{B}'(1) = n(\mathbf{B}_n - \mathbf{B}_{n-1}), \quad \mathbf{B}''(1) = (n-1)n((\mathbf{B}_{n-2} - \mathbf{B}_{n-1}) + (\mathbf{B}_n - \mathbf{B}_{n-1})). \quad (6)$$

Note that defining “normalized” derivatives as

$$\bar{\mathbf{B}}'(t) \equiv (\bar{B}_x'(t), \bar{B}_y'(t)) \stackrel{\text{def}}{=} \frac{1}{n}\mathbf{B}'(t), \quad \bar{\mathbf{B}}''(t) \equiv (\bar{B}_x''(t), \bar{B}_y''(t)) \stackrel{\text{def}}{=} \frac{1}{(n-1)n}\mathbf{B}''(t), \quad (7)$$

we can rewrite dependencies (5) and (6) as

$$\bar{\mathbf{B}}'(0) = \mathbf{B}_1 - \mathbf{B}_0, \quad \bar{\mathbf{B}}''(0) = (\mathbf{B}_0 - \mathbf{B}_1) + (\mathbf{B}_2 - \mathbf{B}_1), \quad (8)$$

$$\bar{\mathbf{B}}'(1) = \mathbf{B}_n - \mathbf{B}_{n-1}, \quad \bar{\mathbf{B}}''(1) = (\mathbf{B}_{n-2} - \mathbf{B}_{n-1}) + (\mathbf{B}_n - \mathbf{B}_{n-1}). \quad (9)$$

The geometric interpretation, shown in figure 2, should help to understand the formulas (8) and (9), especially the expression for the second derivative. For example, vectors $\bar{\mathbf{B}}''(0)$ and $\bar{\mathbf{B}}''(1)$ shown in figure 2, indicate the direction of the centripetal force acting on a fictive point passenger of the point vehicle (mentioned at the beginning of this section) at the endpoints \mathbf{B}_0 and \mathbf{B}_n , respectively.

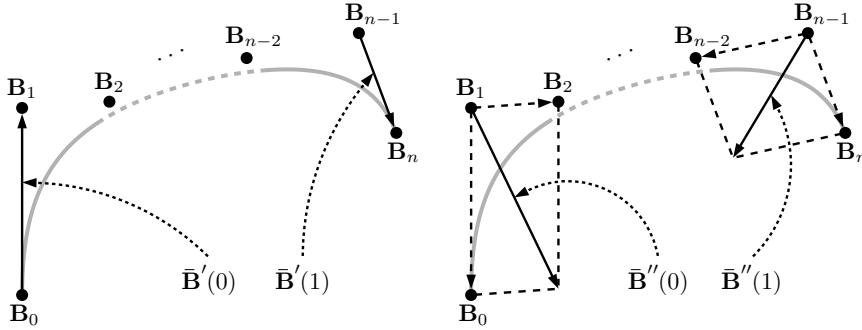


Figure 2: Geometric interpretation of the first and second derivatives of a Bernstein polynomial.

4 Curvature

In what follows, the precise definition of the curvature of a planar curve will be needed. For the purpose of this paper, the most commonly accepted definition which is simple and is (the author believes) the most natural one will be used. According to this definition, the curvature of a flat curve is simply the measure of the change of curve direction; more precisely, the change of the angle counted with respect to the curve length for an infinitesimal change of the “time” parameter — see figure 3.

Applying elementary transformations of the curvature formula presented in figure 3, one obtains

$$\begin{aligned}\kappa(t) &= \lim_{\Delta t \rightarrow 0} \frac{\Delta\varphi}{\Delta t} \frac{1}{\Delta s/\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{\Delta\varphi}{\Delta t} \frac{1}{\sqrt{(\Delta x/\Delta t)^2 + (\Delta y/\Delta t)^2}} = \\ &= \frac{d\varphi}{dt} \frac{1}{\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}}.\end{aligned}\quad (10)$$

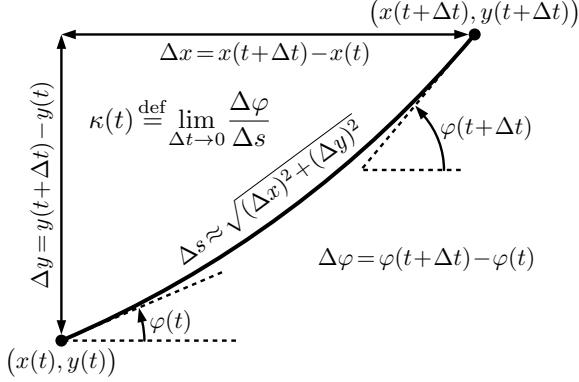


Figure 3: Definition of the curvature $\kappa(t)$ of a planar curve defined parametrically as $(x(t), y(t))$; for purposes of consistency with the definition of a determinant (2), counterclockwise angles will be considered positive (cf. figure 4).

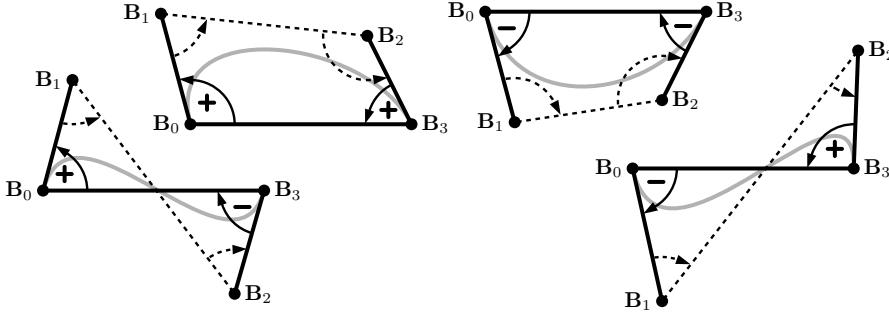


Figure 4: Convention of determining of the angle $\angle B_a B_b B_c$ with absolute value less than π : the sign of the directed angle (from the arm $B_b B_a$ to the arm $B_b B_c$) at the vertex B_b will be considered consistent with the sign of the determinant $\det(B_a - B_b, B_c - B_b)$; arrows denote the assumed orientation of the angles, gray lines mark the shape of the Bézier arcs defined by the respective quadrilaterals $B_0 B_1 B_2 B_3$, according to equation (1) for $k = 2$ and $n = 3$.

Using a more convenient notation already used in the previous point, namely, $x'(t) \equiv \frac{dx}{dt}$, $y'(t) \equiv \frac{dy}{dt}$, formula $\varphi(t) = \arctan\left(\frac{y'(t)}{x'(t)}\right)$, and applying basic differential calculus identities, one obtains

$$\begin{aligned} \frac{d\varphi}{dt} \equiv \varphi'(t) &= \left(\arctan\left(\frac{y'(t)}{x'(t)}\right) \right)' = \frac{1}{1 + \left(\frac{y'(t)}{x'(t)}\right)^2} \left(\frac{y'(t)}{x'(t)} \right)' \\ &= \frac{1}{1 + \left(\frac{y'(t)}{x'(t)}\right)^2} \frac{x'(t)y''(t) - y'(t)x''(t)}{x'(t)^2} \\ &= \frac{x'(t)y''(t) - y'(t)x''(t)}{x'(t)^2 + y'(t)^2}. \end{aligned} \quad (11)$$

Observe that the numerator of formula (11) is nothing other than the determinant, namely, $\det((x'(t), y'(t)), (x''(t), y''(t)))$, which explains our interest in determinants. We will come back to determinants later.

Combining equations (10) and (11) eventually yields

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{(x'(t)^2 + y'(t)^2)^{3/2}}. \quad (12)$$

The above derivation is far from complete mathematical precision but making it more rigorous does not seem difficult (e.g., it suffices to take arccot instead of arctan in order to handle the case $x'(t) = 0$).

The resulting formula (12) can be assigned a simple geometrical meaning.

The simplest case is a straight line defined by $z(t) = (a_x t + b_x, a_y t + b_y)$, where a_x, b_x, a_y and b_y are real numbers such that $a_x^2 + a_y^2 \neq 0$. Obviously, it has curvature equal to zero which is no surprise. Consider thus a less trivial example, namely, a circle of radius R with its center in the origin of the coordinate system, oriented counterclockwise: $z(t) = (R \cos(t), R \sin(t))$, $0 \leq t < 2\pi$. Obviously, $z'(t) = (-R \sin(t), R \cos(t))$ and $z''(t) = (-R \cos(t), -R \sin(t))$; from formula (12), it can immediately be derived that

$$\kappa = \frac{R^2}{(R^2)^{3/2}} = \frac{1}{|R|} = \text{const.}$$

Note that changing the curve orientation to clockwise, $z(t) = (R \cos(t), -R \sin(t))$, results in the change of the sign of curvature:

$$\kappa = -\frac{1}{|R|}.$$

This example explains why the entity κ^{-1} is called the *radius of curvature* and why clockwise planar curves are called *negatively oriented* while counterclockwise planar curves are called *positively oriented*.

The sign of curvature is, obviously, related to the accepted convention for determining the sign of an angle (cf. figures 3 and 4). The curvature is constant in the case of a straight line and a circle. In general, it is a local quantity. Positive curvature means that the curve (locally) turns to the left, negative curvature that it turns to the right. The points at which the curvature changes sign are called inflection points.

4.1 Curvature of Bernstein polynomials

The formula for curvature for Bernstein polynomials (12) expressed with “normalized” derivatives (7) takes the form

$$\kappa(t) = \frac{B'_x(t)B''_y(t) - B'_y(t)B''_x(t)}{(B'_x(t)^2 + B'_y(t)^2)^{3/2}} = \frac{n-1}{n} \frac{\bar{B}'_x(t)\bar{B}''_y(t) - \bar{B}'_y(t)\bar{B}''_x(t)}{(\bar{B}'_x(t)^2 + \bar{B}'_y(t)^2)^{3/2}}. \quad (13)$$

Hence, taking into account formulas (2), (8) and (9), it is easy to determine the curvature for $t = 0$ and $t = 1$

$$\begin{aligned} \kappa(0) &= \frac{n-1}{n} \frac{\det((\mathbf{B}_1 - \mathbf{B}_0), (\mathbf{B}_2 - \mathbf{B}_1) + (\mathbf{B}_0 - \mathbf{B}_1))}{|\mathbf{B}_1 - \mathbf{B}_0|^3}, \\ \kappa(1) &= \frac{n-1}{n} \frac{\det((\mathbf{B}_n - \mathbf{B}_{n-1}), (\mathbf{B}_{n-2} - \mathbf{B}_{n-1}) + (\mathbf{B}_n - \mathbf{B}_{n-1}))}{|\mathbf{B}_{n-1} - \mathbf{B}_n|^3}, \end{aligned} \quad (14)$$

where the notation $|\mathbf{V}|$ means the length of a vector \mathbf{V} . Next, using property (3), one obtains

$$\begin{aligned} \kappa(0) &= \frac{n-1}{n} \frac{\det((\mathbf{B}_1 - \mathbf{B}_0), (\mathbf{B}_2 - \mathbf{B}_1))}{|\mathbf{B}_1 - \mathbf{B}_0|^3}, \\ \kappa(1) &= \frac{n-1}{n} \frac{\det((\mathbf{B}_n - \mathbf{B}_{n-1}), (\mathbf{B}_{n-2} - \mathbf{B}_{n-1}))}{|\mathbf{B}_{n-1} - \mathbf{B}_n|^3}. \end{aligned} \quad (15)$$

Let $h_2^{0,1}$ be the distance (directed) of the point \mathbf{B}_2 to the straight line connecting the points \mathbf{B}_0 and \mathbf{B}_1 and, similarly, let $h_{n-2}^{n-1,n}$ be the distance of the point \mathbf{B}_{n-2} to the straight line connecting the points \mathbf{B}_{n-1} and \mathbf{B}_n ; note that $h_2^{0,1}$ and $h_{n-2}^{n-1,n}$ have the same signs as the determinants $\det((\mathbf{B}_1 - \mathbf{B}_0), (\mathbf{B}_2 - \mathbf{B}_1))$ and $\det((\mathbf{B}_n - \mathbf{B}_{n-1}), (\mathbf{B}_{n-2} - \mathbf{B}_{n-1}))$, respectively—cf. figure 5: h_a corresponds to $h_2^{0,1}$ and h_b corresponds to $h_{n-2}^{n-1,n}$, i.e., $h_1^{2,3}$.

In other words, if the point \mathbf{B}_2 is located to the right of the straight line directed from \mathbf{B}_0 to \mathbf{B}_1 , then $h_2^{0,1} < 0$; if it is located to the left, then $h_2^{0,1} > 0$. Similarly, if the point \mathbf{B}_{n-2}

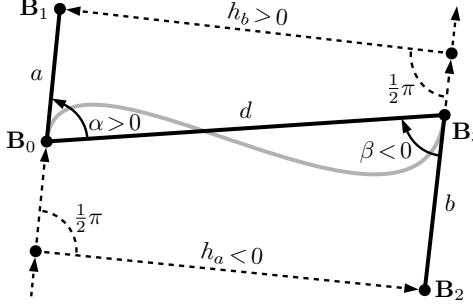


Figure 5: Geometrical interpretation of formulas (17) for a Bézier arc ($n = 3$); the curvature at the points \mathbf{B}_0 and \mathbf{B}_3 is given by (18).

is located to the right of the straight line directed from \mathbf{B}_{n-1} to \mathbf{B}_n , then $h_{n-2}^{n-1,n} < 0$; if it is located to the left, then $h_{n-2}^{n-1,n} > 0$.

As was said in section 2, the absolute value of a determinant of a pair of vectors on a plane is equal to the area of the relevant parallelogram, thus

$$\begin{aligned}\det((\mathbf{B}_1 - \mathbf{B}_0), (\mathbf{B}_2 - \mathbf{B}_1)) &= h_2^{0,1} |\mathbf{B}_1 - \mathbf{B}_0|, \\ \det((\mathbf{B}_{n-1} - \mathbf{B}_n), (\mathbf{B}_{n-2} - \mathbf{B}_{n-1})) &= h_{n-2}^{n,n-1} |\mathbf{B}_{n-1} - \mathbf{B}_n|.\end{aligned}\quad (16)$$

Hence eventually

$$\kappa(0) = \frac{n-1}{n} \frac{h_2^{0,1}}{|\mathbf{B}_1 - \mathbf{B}_0|^2}, \quad \kappa(1) = \frac{n-1}{n} \frac{h_{n-2}^{n,n-1}}{|\mathbf{B}_{n-1} - \mathbf{B}_n|^2}. \quad (17)$$

The derivation of formulas (17) completes our considerations for the general $(n+1)$ -node case.

4.2 Curvature of Bézier arcs

Now we can proceed to deriving formulas for the curvature of Bézier arcs, i.e., for the 4-node case. The resulting formulas will be used for the smooth connection of arcs.

As was already mentioned, curvature sign depends on the convention for determining the sign of an angle. In the following, we will determine the sign of an angle (less than π) in a $\mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$ quadrilateral according to the convention accepted so far, i.e., we will consider each angle as directed from the previous to the next edge (see figures 3 and 4). Let us examine the situation presented in figure 5: obviously $\alpha > 0$ whereas $\beta < 0$. According to the definition of the directed distance introduced in section 4.1, $h_a < 0$ and $h_b > 0$.

Using the same notation as in figure 5, the curvature formulas for the nodes (endpoints) can be expressed as follows (regardless of the vertex configuration of the relevant quadrilateral):

$$\kappa(0) = \frac{2}{3} \frac{h_a}{a^2}, \quad \kappa(1) = \frac{2}{3} \frac{h_b}{b^2}. \quad (18)$$

Now, we will eliminate the entities h_a and h_b from formulas (18) in order to express curvature as a function of length (which is always non-negative) of the handles (a and b), the chord (d), and the (directed) angles α and β .

It follows from figure 6 that $c = d - b \sin(\beta) \cot(\alpha) - b \cos(\beta)$ and $h_a = -c \sin(\alpha)$, hence

$$h_a = -(d \sin(\alpha) - b \sin(\beta) \cos(\alpha) - b \sin(\alpha) \cos(\beta)) = b \sin(\alpha + \beta) - d \sin(\alpha). \quad (19)$$

Similarly

$$h_b = a \sin(\alpha + \beta) - d \sin(\beta). \quad (20)$$

Making use of formulas (19) and (20), we arrive eventually at the expression for Bézier curvature at the endpoints, expressed as a function of length of the handles and angles between the respective handles and the chord

$$\kappa(0) = \frac{2}{3} \frac{b \sin(\alpha + \beta) - d \sin(\alpha)}{a^2}, \quad \kappa(1) = \frac{2}{3} \frac{a \sin(\alpha + \beta) - d \sin(\beta)}{b^2}. \quad (21)$$

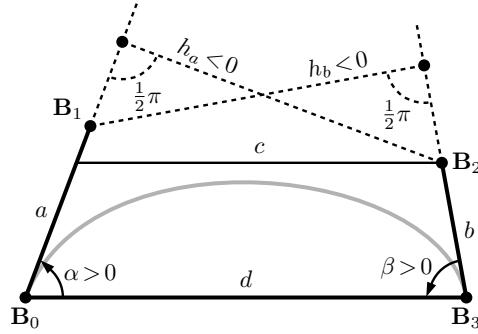


Figure 6: Auxiliary sketch for deriving formula (21) which describes the relationship between curvature and the relevant angles (here α and β) and the length of the handles and the chord (a , b and d , respectively).

Let us emphasize once again that the resulting formulas are invariant with respect to the vertex configuration of the quadrilateral $\mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$, i.e., they do not depend on the signs of the angles α and β (cf. figure 4).

4.3 Curvature peculiarities

As figure 7 shows, the mathematical notion of curvature usually reflects what the human eye can see; curves with slightly changing curvature are likely to be called smooth or “neatly formed”. If curvature changes rapidly, then we are usually able to point to the place where such a change occurs without mathematical considerations.

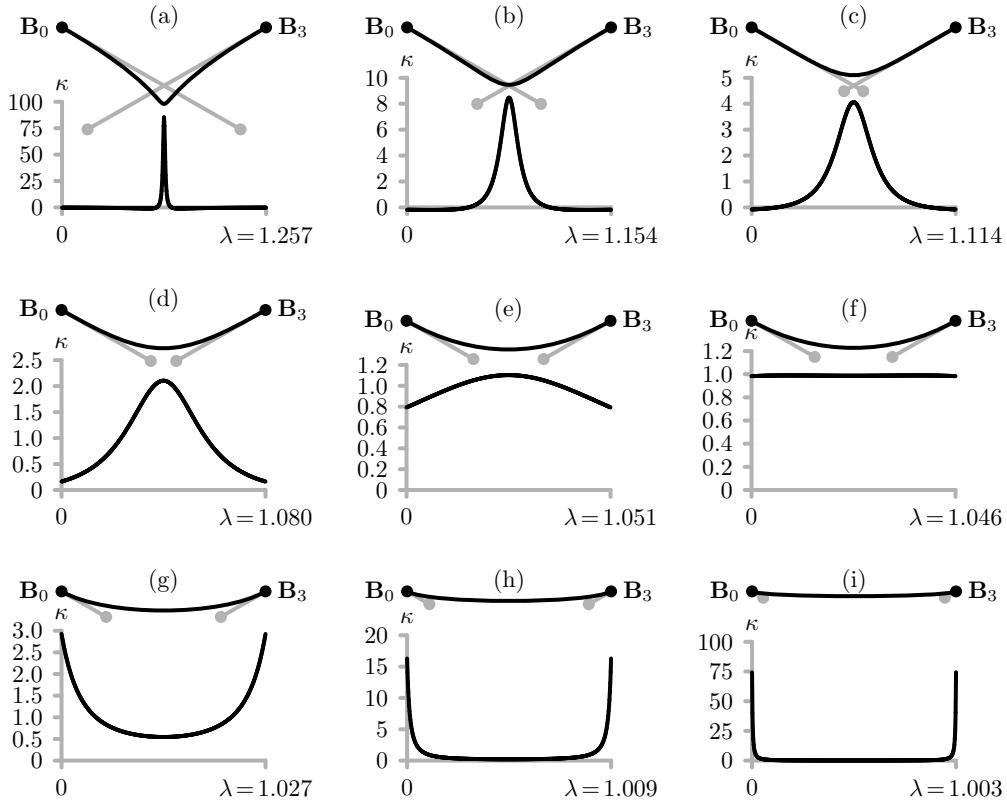


Figure 7: Curvature κ of the Bézier arc having a unit chord ($|\mathbf{B}_0 - \mathbf{B}_3| = 1$); case (f) results from applying formula (29).

It follows from equation (21), however, that curvature can achieve large values when the respective handles are short. It can even approach infinity if control points are very close to the respective endpoints. Even worse, infinite curvature can be imperceptible to the human eye, as figure 7 shows: for cases (a)–(c) the change of curvature is clearly visible in the middle of the diagram; cases (d)–(f) are perceived as fragments of a circle or “circle-like” shapes; cases (g)–(i) illustrate a transformation from a Bézier arc to a chord as the control nodes approach the respective endpoints; in the latter case, paradoxically, the curvature near the endpoints suddenly increases despite the eye not noticing it.

5 Bézier arc and a circle

Let us assume that for given angles α and β , $-\pi < \alpha, \beta \leq \pi$, and for chord length $d \neq 0$, the length of the handles can be calculated from the following formulas

$$a = \frac{1}{3} d \frac{\rho(\alpha, \beta)}{\tau_a}, \quad b = \frac{1}{3} d \frac{\sigma(\alpha, \beta)}{\tau_b}, \quad (22)$$

where $\rho(\alpha, \beta)$ and $\sigma(\alpha, \beta)$ are certain functions, not necessarily given explicitly (we will discuss them in a moment) and τ_a and τ_b are given real (positive) numbers; we will call these numbers *tensions*.

In the METAFONT and METAPOST programs, tensions can be specified explicitly using the ‘tension’ operator ([6, pp. 129–132 and p. 136, ex. 14.15]). Usually, in practical applications, $\tau_a = \tau_b = 1$ which is the default tension value in METAFONT and METAPOST.

Let us assume for a moment that $\tau_a = \tau_b = 1$; moreover, assume that $d = 1$ (in other words, adjust units in such a way that $d = 1$). Then, the first derivative vectors at the endpoints ($t = 0$ and $t = 1$; cf. (5) and (6)) have the length $\rho(\alpha, \beta)$ and $\sigma(\alpha, \beta)$, respectively. This explains why the functions ρ and σ are called *velocity functions*.

In applications under consideration, it is natural to assume a basic symmetry property: an exchange of variables should return a geometrically congruent (mirrored) figure, precisely

$$\rho(\alpha, \beta) = \sigma(\beta, \alpha). \quad (23)$$

Therefore, we are actually dealing with one function, but it is more convenient to distinguish between the velocity functions at the points $t = 0$ and $t = 1$.

In the METAFONT and METAPOST programs, the velocity function is defined with a relatively complex heuristic formula (due to Hobby):

$$\rho(\alpha, \beta) = \sigma(\beta, \alpha) = \frac{2 + \sqrt{2}(\sin \alpha - \frac{1}{16} \sin \beta)(\sin \beta - \frac{1}{16} \sin \alpha)(\cos \alpha - \cos \beta)}{(1 + \frac{1}{2}(\sqrt{5} - 1) \cos \alpha + \frac{1}{2}(3 - \sqrt{5}) \cos \beta)}, \quad (24)$$

substantiated in his paper [2]. An alternative form is also given (defined for $0 < |\alpha| \leq \beta < \pi$), which supposedly works better in asymmetrical cases but is far more complex computationally and more difficult to analyze theoretically:

$$\rho(\alpha, \beta) = f(\alpha, \beta) + \gamma(\beta) \sin\left(\psi_\beta\left(\frac{\alpha}{\beta}\right)\right), \quad \sigma(\alpha, \beta) = f(\alpha, \beta) - \gamma(\beta) \sin\left(\psi_\beta\left(\frac{\alpha}{\beta}\right)\right), \quad (25)$$

where

$$f(\alpha, \beta) = \frac{m\mu^2 + \mu + 2n}{\mu + n \cos(\nu) + n}, \quad m = 0.2678306, \quad n = 0.2638750,$$

$$\mu = (\beta - \alpha) \left(\frac{\beta - \alpha}{2\beta} \right)^{1.402539}, \quad \nu = \frac{\alpha + \beta}{2} \left(\frac{2\beta}{\alpha + \beta} \right)^{0.7539063}, \quad (26)$$

$$\gamma(\beta) = \frac{1.17}{\pi} \beta - 0.15 \sin(2\beta), \quad \psi_\beta(x) = \pi \left(x + (x^2 - 1) \left(\left(0.32 - \frac{\beta}{2\pi} \right) x + 0.5 - \frac{\beta}{2\pi} \right) \right).$$

Knuth, in his collection of essays entitled *Digital Typography* [4], devotes some attention to the choice of functions ρ and σ ; he proposes, among others, the following formula

$$\rho(\alpha, \beta) = \sigma(\beta, \alpha) = \frac{2 \sin(\beta)}{\left(1 + \cos\left(\frac{\alpha + \beta}{2}\right)\right) \sin\left(\frac{\alpha + \beta}{2}\right)} \quad (27)$$

and suggests how to improve it.

A formula of this kind, but much simpler, was first proposed by J. R. Manning in his already cited paper [7] (Hobby's formula (24) is in fact a sophisticated modification of Manning's formula):

$$\rho(\alpha, \beta) = \sigma(\beta, \alpha) = \frac{2}{1 + c \cos(\beta) + (1 - c) \cos(\alpha)}. \quad (28)$$

Manning suggests setting $c = \frac{2}{3}$. The idea behind Manning's formula stems from a simple observation: if $\alpha = \beta$, then the formula $\rho(\alpha) = 2/(1 + \cos(\alpha))$ provides a good approximation of a circle by a Bézier arc. Namely, if both (symmetric) handles of the Bézier arc $\mathbf{B}(t)$ have the length given by

$$\rho(\alpha) = \frac{d}{3} \frac{2}{1 + \cos(\alpha)}, \quad (29)$$

where d is the length of the chord of the arc and α is the angle between handles and the chord, then the point $\mathbf{B}(\frac{1}{2})$ coincides with the center of the segment of a circle going through the points $\mathbf{B}(0)$ and $\mathbf{B}(1)$ and tangent to the Bézier arc at these points. Such a Bézier arc, especially for small angles, is visually indistinguishable from a circle—more information on the precision of such an approximation can be found in [4]; cf. also figure 7f.

All in all, we may note that the velocity function is intended to be a generalization, heuristic of course, to two parameters of the function $\rho(\alpha)$ guaranteeing a good approximation (equation (29)) of a circle by a Bézier arc. This remains, of course, related to our primary goal—striving to obtain a possibly smooth joining of Bézier arcs. Bézier arcs computed using velocity functions as defined above are expected to imitate circles optically which, because of their constant curvature, are supposed to be an ideal candidate for an “æsthetic model”. This observation provides a clue for our final results.

6 Smooth joining of Bézier arcs—J. D. Hobby's method

Finally, we have all the necessary tools and measures necessary to face the following task: given a series of $n + 1$ points on a plane $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$, $n > 1$, connect these points with Bézier arcs in such a way that the result forms a possibly “elegant” (smooth, neat) curve.

The *sine qua non* condition is obvious: at a connection point the curve should not change its direction abruptly, meaning that the handles at the connection points must be collinear. (Collinearity of handles is a weaker condition than equality of derivatives; the equality of derivatives, as defined by formulas (5) and (6), implies the equality of the lengths of adjacent handles.)

This condition does not guarantee a unique solution. We might impose an additional condition that the curvature at a junction point is the same on both sides of this point. However, this leads to a system of trigonometrical equations that is hard to solve.

In such cases, mathematicians and physicist often replace a function with its linear approximation (not always justifiably); in this particular case, the sine function would be replaced by a linear function of the argument and the relevant velocity function—with a function identically equal to 1.

$$\sin(\alpha) \approx \alpha, \quad \rho(\alpha, \beta) \approx \sigma(\alpha, \beta) \approx 1. \quad (30)$$

Such an approximation can be justified mathematically for small values of arguments (with respect to their absolute values) for (24) and (28) (formulas (25) and (27) are undefined for $\alpha = \beta = 0$), but it will also be used for values significantly different from zero. Therefore, we can consider (30) to be a heuristic simplifying assumption.

Applying relation (30) to the formula for Bézier curvature at endpoints, derived from combining formulas (21) and (22), yields

$$\begin{aligned}\kappa(0) &= \frac{2}{3} \frac{\frac{1}{3}d\frac{\sigma(\alpha, \beta)}{\tau_b} \sin(\alpha + \beta) - d \sin(\alpha)}{\left(\frac{1}{3}d\frac{\rho(\alpha, \beta)}{\tau_a}\right)^2} = \frac{2\tau_b^{-1}\sigma(\alpha, \beta) \sin(\alpha + \beta) - 6 \sin(\alpha)}{d\tau_a^{-2}\rho^2(\alpha, \beta)}, \\ \kappa(1) &= \frac{2}{3} \frac{\frac{1}{3}d\frac{\rho(\alpha, \beta)}{\tau_a} \sin(\alpha + \beta) - d \sin(\beta)}{\left(\frac{1}{3}d\frac{\sigma(\alpha, \beta)}{\tau_b}\right)^2} = \frac{2\tau_a^{-1}\rho(\alpha, \beta) \sin(\alpha + \beta) - 6 \sin(\beta)}{d\tau_b^{-2}\sigma^2(\alpha, \beta)},\end{aligned}\quad (31)$$

and we have

$$\kappa(0) \approx \bar{\kappa}(0) \stackrel{\text{def}}{=} \frac{2\tau_b^{-1}(\alpha + \beta) - 6\alpha}{d\tau_a^{-2}}, \quad \kappa(1) \approx \bar{\kappa}(1) \stackrel{\text{def}}{=} \frac{2\tau_a^{-1}(\alpha + \beta) - 6\beta}{d\tau_b^{-2}}. \quad (32)$$

The function $\bar{\kappa}$ defined above is the *mock curvature* introduced by Hobby in [2], mentioned by Knuth in [6] and discussed in detail in [5, § 274–277]. It constitutes a key to joining Bézier arcs smoothly: first we solve a set of linear equations, derived from (32), obtaining the values of angles between chords and the respective handles, and then we compute the length of the relevant handles from (22).

6.1 Equations

At the junctions of Bézier arcs, we require that mock curvature be preserved, bringing the task to solving a set of linear equations. The formulation of these equations consists of a series of tedious but elementary calculations.

First, let us adopt the notation and conventions presented in figure 8.

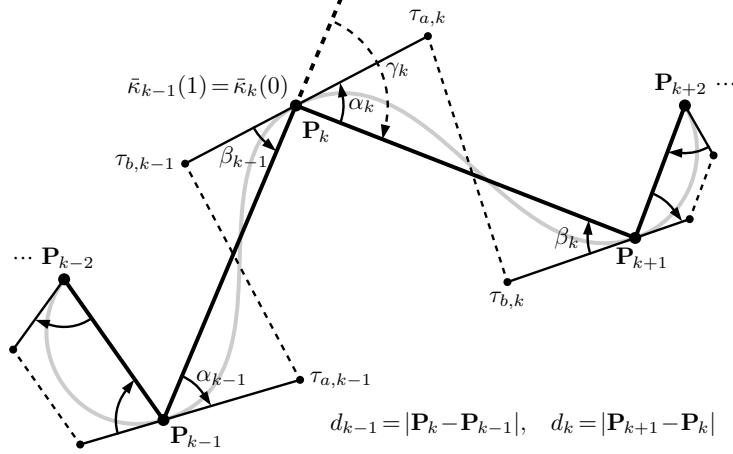


Figure 8: Notation assumed for formulating linear equations (38); index k refers (unlike in [5]) to the quantities related to the Bézier arc based on the endpoints \mathbf{P}_k and \mathbf{P}_{k+1} .

Second, observe that the collinearity of handles at junction points implies that

$$\alpha_k + \beta_{k-1} + \gamma_k = 0 \text{ for } k = 1, 2, \dots, n-1, \quad (33)$$

where γ_k is the turning angle of the broken line $\mathbf{P}_0 \mathbf{P}_1 \dots \mathbf{P}_n$ at the point \mathbf{P}_k .

Third, observe moreover that the preservation of the mock curvature at the point \mathbf{P}_k results in a simple linear equation

$$\bar{\kappa}_{k-1}(1) \equiv \frac{2\tau_{a,k-1}^{-1}(\alpha_{k-1} + \beta_{k-1}) - 6\beta_{k-1}}{d_{k-1}\tau_{b,k-1}^{-2}} = \frac{2\tau_{b,k}^{-1}(\alpha_k + \beta_k) - 6\alpha_k}{d_k\tau_{a,k}^{-2}} \equiv \bar{\kappa}_k(0). \quad (34)$$

By making use of (33), we can eliminate β_{k-1} and β_k from (34) obtaining

$$\frac{\tau_{a,k-1}^{-1}(\alpha_{k-1} - \alpha_k - \gamma_k) - 3(-\alpha_k - \gamma_k)}{d_{k-1}\tau_{b,k-1}^{-2}} - \frac{\tau_{b,k}^{-1}(\alpha_k - \alpha_{k+1} - \gamma_{k+1}) - 3\alpha_k}{d_k\tau_{a,k}^{-2}} = 0, \quad (35)$$

i.e.,

$$\begin{aligned} & \frac{\tau_{a,k-1}^{-1}}{d_{k-1}\tau_{b,k-1}^{-2}}\alpha_{k-1} + \left(\frac{3 - \tau_{a,k-1}^{-1}}{d_{k-1}\tau_{b,k-1}^{-2}} + \frac{3 - \tau_{b,k}^{-1}}{d_k\tau_{a,k}^{-2}} \right)\alpha_k + \frac{\tau_{b,k}^{-1}}{d_k\tau_{a,k}^{-2}}\alpha_{k+1} \\ &= -\frac{3 - \tau_{a,k-1}^{-1}}{d_{k-1}\tau_{b,k-1}^{-2}}\gamma_k - \frac{\tau_{b,k}^{-1}}{d_k\tau_{a,k}^{-2}}\gamma_{k+1}. \end{aligned} \quad (36)$$

Now, by introducing one-letter symbols for known values (coefficients)

$$\begin{aligned} A_k &\stackrel{\text{def}}{=} \frac{\tau_{a,k-1}^{-1}}{d_{k-1}\tau_{b,k-1}^{-2}}, & B_k &\stackrel{\text{def}}{=} \frac{3 - \tau_{a,k-1}^{-1}}{d_{k-1}\tau_{b,k-1}^{-2}}, & C_k &\stackrel{\text{def}}{=} \frac{3 - \tau_{b,k}^{-1}}{d_k\tau_{a,k}^{-2}}, \\ D_k &\stackrel{\text{def}}{=} \frac{\tau_{b,k}^{-1}}{d_k\tau_{a,k}^{-2}}, & E_k &\stackrel{\text{def}}{=} -B_k\gamma_k - D_k\gamma_{k+1}, \end{aligned} \quad (37)$$

we obtain a set of $n - 1$ linear equations with $n + 1$ unknowns:

$$\begin{aligned} A_1\alpha_0 + (B_1 + C_1)\alpha_1 + D_1\alpha_2 &= E_1, \\ A_2\alpha_1 + (B_2 + C_2)\alpha_2 + D_2\alpha_3 &= E_2, \\ A_3\alpha_2 + (B_3 + C_3)\alpha_3 + D_3\alpha_4 &= E_3, \\ &\vdots \\ A_{n-1}\alpha_{n-2} + (B_{n-1} + C_{n-1})\alpha_{n-1} + D_{n-1}\alpha_n &= E_{n-1}. \end{aligned} \quad (38)$$

Thus, two more equations are needed.

If a closed curve is to be obtained, then $\alpha_n = \alpha_0$ and the problem reduces to n unknowns. The missing equation can be obtained by assuming that mock curvature is preserved at the point $\mathbf{P}_0 = \mathbf{P}_n$,

$$A_0\alpha_{n-1} + (B_0 + C_0)\alpha_0 + D_0\alpha_1 = E_0. \quad (39)$$

If an open curve is to be obtained, then either the angles α_0 and α_n should be given explicitly which reduces the number of unknowns by 2, or another condition must be found.

Before we proceed to this issue, let us discuss a purely technical detail; the angle that we want to find is, in fact, β_{n-1} because α_n is located outside the broken line $\mathbf{P}_0\mathbf{P}_1\dots\mathbf{P}_n$. It is, however, more convenient (because of the symmetry of formulas), to use the unknown α_n ; therefore, we accept an artificial condition

$$\gamma_n = 0. \quad (40)$$

This means that relation (33) for the angles β_0 and β_{n-1} , in the case of an open curve, takes the form

$$\beta_0 = -\alpha_1 - \gamma_1, \quad \beta_{n-1} = -\alpha_n. \quad (41)$$

We may also assume that mock curvature at the endpoints is given explicitly or implicitly or, alternatively, we may impose the requirement of the equality of mock curvature at the endpoints and at the respective neighbouring nodes, namely, $\bar{\kappa}_0(0) = \bar{\kappa}_0(1)$ and $\bar{\kappa}_{n-1}(0) = \bar{\kappa}_{n-1}(1)$. The latter condition can be generalized in a natural way as follows:

$$\bar{\kappa}_0(0) = \omega_0\bar{\kappa}_0(1), \quad \omega_n\bar{\kappa}_{n-1}(0) = \bar{\kappa}_{n-1}(1), \quad \omega_0, \omega_n \geq 0. \quad (42)$$

It is exactly this idea that has been used in the METAFONT program (there is, however, no possibility of explicitly setting the mock curvature at a given point). The additional parameters ω_0 and ω_n are called *curls*.

Both METAFONT and METAPOST allow to explicitly set curls at the endpoints of a path using the ‘curl’ operator ([6, pp. 128ff.]); by default, curl=1. Conditions (42) can be

transformed to

$$\begin{aligned} \frac{2\tau_{b,0}^{-1}(\alpha_0 - \alpha_1 - \gamma_1) - 6\alpha_0}{d_0\tau_{a,0}^{-2}} &= \omega_0 \frac{2\tau_{a,0}^{-1}(\alpha_0 - \alpha_1 - \gamma_1) + 6(\alpha_1 + \gamma_1)}{d_0\tau_{b,0}^{-2}}, \\ \omega_n \frac{2\tau_{b,n-1}^{-1}(\alpha_{n-1} - \alpha_n) - 6\alpha_{n-1}}{d_{n-1}\tau_{a,n-1}^{-2}} &= \frac{2\tau_{a,n-1}^{-1}(\alpha_{n-1} - \alpha_n) + 6\alpha_n}{d_{n-1}\tau_{b,n-1}^{-2}}. \end{aligned} \quad (43)$$

by using the definition of mock curvature (32) to develop $\bar{\kappa}_0(0)$, $\bar{\kappa}_0(1)$, $\bar{\kappa}_{n-1}(0)$, and $\bar{\kappa}_{n-1}(1)$.

Using relation (41) to order equations (43) yields

$$\begin{aligned} \left(\frac{\tau_{b,0}^{-1}}{\tau_{a,0}^{-2}} - \frac{3}{\tau_{a,0}^{-2}} - \frac{\tau_{a,0}^{-1}\omega_0}{\tau_{b,0}^{-2}} \right) \alpha_0 - \left(\frac{\tau_{b,0}^{-1}}{\tau_{a,0}^{-2}} - \frac{\tau_{a,0}^{-1}\omega_0}{\tau_{b,0}^{-2}} + \frac{3\omega_0}{\tau_{b,0}^{-2}} \right) \alpha_1 &= \left(\frac{\tau_{b,0}^{-1}}{\tau_{a,0}^{-2}} - \frac{\tau_{a,0}^{-1}\omega_0}{\tau_{b,0}^{-2}} + \frac{3\omega_0}{\tau_{b,0}^{-2}} \right) \gamma_1, \\ \left(\frac{\tau_{b,n-1}^{-1}\omega_n}{\tau_{a,n-1}^{-2}} - \frac{3\omega_n}{\tau_{a,n-1}^{-2}} - \frac{\tau_{a,n-1}^{-1}}{\tau_{b,n-1}^{-2}} \right) \alpha_{n-1} - \left(\frac{\omega_n\tau_{b,n-1}^{-1}}{\tau_{a,n-1}^{-2}} - \frac{\tau_{a,n-1}^{-1}}{\tau_{b,n-1}^{-2}} + \frac{3}{\tau_{b,n-1}^{-2}} \right) \alpha_n &= 0, \end{aligned} \quad (44)$$

By multiplying the first equation (44) by $-\tau_{a,0}^{-2}$, and the second by $-\tau_{b,n-1}^{-2}$ (thanks to which the obtained formulas are easier to compare with those given in [5]), we finally obtain the two missing equations:

$$\begin{aligned} C_0\alpha_0 + D_0\alpha_1 &= E_0, \\ A_n\alpha_{n-1} + B_n\alpha_n &= 0, \end{aligned} \quad (45)$$

where

$$\begin{aligned} C_0 &= \omega_0 \frac{\tau_{a,0}^{-3}}{\tau_{b,0}^{-2}} + 3 - \tau_{b,0}^{-1}, \quad D_0 = \omega_0 \frac{\tau_{a,0}^{-2}}{\tau_{b,0}^{-2}} (3 - \tau_{a,0}^{-1}) + \tau_{b,0}^{-1}, \quad E_0 = -D_0\gamma_1, \\ A_n &= \omega_n \frac{\tau_{b,n-1}^{-2}}{\tau_{a,n-1}^{-2}} (3 - \tau_{b,n-1}^{-1}) + \tau_{a,n-1}^{-1}, \quad B_n = \omega_n \frac{\tau_{b,n-1}^{-3}}{\tau_{a,n-1}^{-2}} + 3 - \tau_{a,n-1}^{-1}. \end{aligned} \quad (46)$$

One may of course ask about the solvability of the set of linear equations thus formulated. In this way, we go back to determinants as those are exactly the determinants (of degree equal to the number of unknowns) that mathematicians use for examining this problem; columns (or rows) of coefficients of the set of equations constitute relevant “vectors”. If a determinant for a given set of equations is non-zero, then there exists a unique solution.

We will not go into mathematically advanced analysis of this issue (readers interested in details, please refer to [2]). We content ourselves by adducing without proof the theorem given in [2] (see also [5, § 276]): if for $0 \leq k \leq n-1$ we assume $\tau_{a,k} \geq \frac{3}{4}$, $\tau_{b,k} \geq \frac{3}{4}$ (a limitation built into METAFONT and METAPOST), then the equation sets for both a closed path (38)+(39) and an open path (38)+(45) have unique solutions and, in general, any disturbance introduced at a given node, caused, e.g., by the change of a node location, disappears exponentially as the distance from this node grows (more precisely: the change of conditions at node k results in an angle change at node j proportional to $2^{-|k-j|}$). It should be noted, however, that there are cases for which a tiny perturbation at a certain node may cause the global change of the shape of the resulting curve (see subsection 6.4, figures 10 and 11).

6.2 Two-point case

So far we have assumed that $n > 1$. The discussion can be extended to the $n = 1$ case, i.e., to the two-point case, but this requires additional assumptions.

If $n = 1$ and we are dealing with a closed curve, then it is reasonable to assume that this is simply a degenerate case, i.e., $\mathbf{P}_0 = \mathbf{P}_1$ and, moreover, lengths of both handles are equal to zero—cf. equation (22). Note that the METAFONT path expression ‘ $(0,0)\dots(1,0)\dots\text{cycle}$ ’ corresponds, in fact, to the case $n = 2$: $\mathbf{P}_0 = (0,0)$, $\mathbf{P}_1 = (1,0)$, $\mathbf{P}_2 = (0,0) = \mathbf{P}_0$.

If $n = 1$ and we are dealing with an open curve, then the degenerate case $\mathbf{P}_0 = \mathbf{P}_1$ can be treated as above. In other cases, the set of equations reduces to two equations (45). In this case, $E_0 = 0$ (because $\gamma_1 = 0$ by (40)) which means that $\alpha_0 = \alpha_1 = 0$, provided the relevant matrix determinant is non-zero. However, for default values of coefficients defining curl and tension, i.e., 1, we have $C_0 = D_0 = A_1 = B_1 = 3$, thus the determinant equals 0

which means that the set of equations (45) has infinitely many solutions and therefore again an additional assumption is needed. The simplest way would be to assume that if no angles are given explicitly, then $\alpha_0 = \alpha_1 = 0$.

If the angle α_0 is given, then α_1 can be computed from the second equation of system (45); and vice versa, if the angle α_1 is given, then α_0 can be computed from the first equation of system (45); the asymmetry is due to the asymmetry of the curl boundary conditions (equations (45) and (46)) and to the fact that curl and angle cannot both be specified at a given endpoint.

At this stage, we have all the angles in question either given explicitly or calculated. The only remaining task is to calculate the length of the handles from formula (22).

6.3 Operator ‘atleast’

The above discussion explains how all the METAFONT path constructors work except for the ‘tension atleast’ operator ([6, p. 129, 132, 136, and ex. 14.15]) which is a variant of the ‘tension’ operator (section 5, equation (22)). It is a primitive operator in the METAFONT and METAPOST engines but, in principle, it could be implemented using METAFONT/METAPOST macros. The recipe is as follows: first skip the ‘atleast’ operator (modifier), i.e., use the standard ‘tension’ operator thereby reducing the problem to a known issue.

Figure 9 illustrates the further procedure. Assume that we have computed all the handles by using the algorithm described above. Then we check whether the nodes defining the relevant Bézier segments form a concave quadrilateral (case (d) in figure 9). If so, then the extension of one of the handles bisects the other—the handle being bisected should be shortened by moving its endpoint to the intersection point (which, on the one hand, results in increasing tension and, on the other hand, introduces a discontinuity of mock curvature). More precisely, the length of the shortened handle (\bar{a} in figure 9) can be determined using the law of sines. In such a way, inflection points can be avoided. Sometimes, however, surprising results are obtained, namely, when the “shortening handle” is very short.

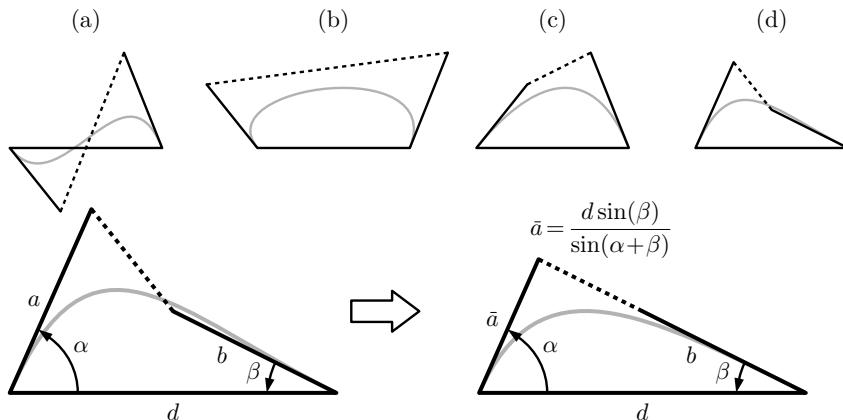


Figure 9: Implementation of the ‘atleast’ operator: only the configuration of the control points presented in figure (d) is to be taken into account; (a), (b) and (c) configurations are ignored.

Let us note that the ‘tension atleast’ operator is a little like increasing the tension just enough to avoid concave quadrilaterals such as shown in figure 9d, except that increasing the tension might affect the direction of angles α and β .

6.4 Instability of the interpolation algorithm — the straight angle issue

One of the crucial steps of Hobby’s algorithm is determining angles γ_k between subsequent segments of the broken line $\mathbf{P}_0\mathbf{P}_1\dots\mathbf{P}_n$ —see figure 8 and equation (33). If the broken line turns by an angle significantly different from the straight angle (its absolute is much less than π), then the algorithm behaves stably, i.e., small changes in the location of the points $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$ yield small changes in the shape of the resulting curve. Unfortunately, an essential

problem arises for angles close to the straight angle. This problem is not in the least specific to METAFONT and METAPOST. It seems to be an intrinsic quandary in the realm of discrete graphics (in general, numerical methods), as small perturbations are unavoidable there due to rounding errors.

In the case of METAFONT and METAPOST, it is a convention adopted by the authors that is the immediate source of problems, namely, both programs operate on angles less than or equal (with respect to the absolute value) to π . If a temporary value occurs which does not meet this limitation, then it is reduced to the appropriate numerical range by adding or subtracting a multiple of 2π . For example, an angle $\pi + \varepsilon$, where $\pi > \varepsilon > 0$, will be replaced by the angle $\pi + \varepsilon - 2\pi = -\pi + \varepsilon$. If $\varepsilon \approx 0$, then unstable behavior can be expected (we omit here the issue of the precision of the computer representation of the number π).

In order to see how instability can manifest, let us consider a trivial example, namely, a cyclic path built from two nodes, \mathbf{P}_0 and \mathbf{P}_1 ; as mentioned in subsection 6.2, it is convenient to regard this example as a three-point case, i.e., to apply the interpolation algorithm to the points \mathbf{P}_0 , \mathbf{P}_1 and \mathbf{P}_2 , where $\mathbf{P}_0 = \mathbf{P}_2$ which means that the respective broken line turns by the straight angle at the points $\mathbf{P}_0 = \mathbf{P}_2$ and \mathbf{P}_2 . The question is: to the right or to the left?

For this case, $\alpha_0 = \alpha_2$ and equations (38) and (39) boil down to the system of two linear equation with two unknowns

$$\begin{aligned} (B_0 + C_0)\alpha_0 + (A_0 + D_0)\alpha_1 &= E_0, \\ (A_1 + D_1)\alpha_0 + (B_1 + C_1)\alpha_1 &= E_1. \end{aligned} \quad (47)$$

Coefficients on the left side of system (47) are given by simple formulas, namely

$$A_0 = A_1 = D_0 = D_1 = \frac{1}{d}, \quad B_0 = B_1 = C_0 = C_1 = \frac{2}{d}, \quad (48)$$

where $d = |\mathbf{P}_1 - \mathbf{P}_0|$, provided the default values of the curl and tension parameters are assumed. Coefficients E_0 and E_1 depend on the turning angles γ_0 and γ_1 of the broken line $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$ (cf. equation (33))

$$E_0 = -\frac{2\gamma_0 + \gamma_1}{d}, \quad E_1 = -\frac{\gamma_0 + 2\gamma_1}{d}. \quad (49)$$

The above conditions can be rewritten as

$$4\alpha_0 + 2\alpha_1 = -(2\gamma_0 + \gamma_1), \quad 2\alpha_0 + 4\alpha_1 = -(\gamma_0 + 2\gamma_1). \quad (50)$$

If we assume turning to the left, i.e., $\gamma_0 \approx \pi$ and $\gamma_1 \approx \pi$, then $\alpha_0 \approx \frac{1}{2}\pi$ and $\alpha_1 \approx \frac{1}{2}\pi$; but if we assume turning to the right, i.e., $\gamma_0 \approx -\pi$ and $\gamma_1 \approx -\pi$, then $\alpha_0 \approx -\frac{1}{2}\pi$ and $\alpha_1 \approx -\frac{1}{2}\pi$. In other words, using the angle $\approx \pi$ instead of $\approx -\pi$ reverses the orientation of the resulting path. Theoretically, two other cases might occur: $\gamma_0 \approx -\pi$, $\gamma_1 \approx \pi$ and $\gamma_0 \approx \pi$, $\gamma_1 \approx -\pi$, which would produce figure-eight shaped curves. It turns out, however, that the METAFONT and METAPOST path expression ‘ $\mathbf{P}_0..P_1..cycle$ ’ usually creates positively oriented nearly circular paths, occasionally creates negatively oriented nearly circular paths (see figure 10), but never figure-eight shaped curves; however, if the direction at one node is specified explicitly, figure-eight shapes may occur: the expression ‘ $(0,0)\{down}\..(100,0)..cycle$ ’ creates an oval path while the expression ‘ $(0,0)\{up}\..(100,0)..cycle$ ’ creates an figure-eight shape.

If the broken line “turns back”, i.e., turns by a nearly straight angle, the algorithm implemented in METAFONT and METAPOST tries to detect this situation and, in most cases, chooses the positive angle (see [5, § 454]). Sometimes, however, despite the precautions taken, the algorithm fails to decide properly whether the angle in question is a positively oriented nearly straight angle, or a distorted by rounding errors negatively oriented (on purpose) nearly straight angle.

Based on the foregoing considerations, it is not difficult to predict that if input data contains several (nearly) collinear adjacent points, instability is bound to emerge.

Without a detailed analysis, we present one more example, a little bit less trivial: we apply the interpolation algorithm to creating a cyclic path (nota bene, cyclicity is not crucial) for input data consisting of five nearly collinear points, arranged horizontally. It turns out that shifting one of the nodes (vertically) by 2^{-16} , i.e., by the smallest non-zero value accepted by

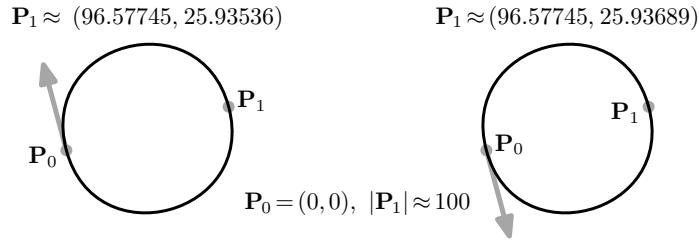


Figure 10: A very small location change of the point \mathbf{P}_1 reverses the orientation of the resulting curve (gray arrow); the curve on the left side of the figure was created with the following METAFONT/METAPOST path expression ‘ $(0,0) .. ((100,0) rotated 15.03189) .. cycle$ ’, the curve on the right side with the slightly modified path expression ‘ $(0,0) .. ((100,0) rotated (15.03189+.0003)) .. cycle$ ’.

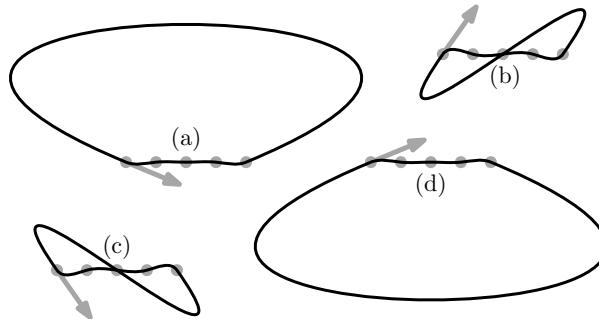


Figure 11: Even a minimal location change of a single point can result not only in the change of orientation, but also in a change of the shape of the resulting curve. In case (a), points are distributed uniformly, i.e., $\mathbf{P}_i = (20i, 0)$, for $i = 0, 1, \dots, 4$; in case (b), \mathbf{P}_1 is shifted up by 2^{-16} ; in case (c), \mathbf{P}_4 is shifted down by 2^{-16} ; in case (d), \mathbf{P}_0 is shifted down by 2^{-16} ; in all cases, the METAFONT/METAPOST path expression ‘ $\mathbf{P}0 .. \mathbf{P}1 .. \mathbf{P}2 .. \mathbf{P}3 .. \mathbf{P}4 .. cycle$ ’ was used.

METAFONT and “canonical” implementations of METAPOST (without double precision), can change not only the orientation of the resulting curve, but also its shape—see figure 11.

Although the problem cannot, by its nature, be cured, it is rather innocuous in practice. Nevertheless, as suggested by John D. Hobby (in private communication), the following fixes to METAFONT/METAPOST implementations can be proposed:

- One natural idea is that users should avoid approaching this situation. METAFONT and METAPOST already have internal variables that can be used to enable and disable certain error messages and an appropriate error message could be helpful.
- Since the discontinuities are caused by adding or subtracting 2π from the γ_k angles, users should perhaps have a way of specifying whether or not to do this.

6.5 Other technical details

It is clear that the implementation of such a complex algorithm as the one described above abounds in technical challenges. Not all of them can be discussed in a paper like this one. Information on specific implementation-related solutions, such as, for example, the fact that METAFONT imposes a limit ≤ 12 (for default tension values) on Hobby’s velocity function (24) or that the curl value can be altered in some cases, can be found in [5] (§ 116 and § 296, respectively). A word of warning is needed, however: the program source is not an easy read.

7 Comparison of selected interpolation methods

The interpolation algorithm described in section 6 works well in most applications, except for the (rarely encountered in practice) cases of instability. This fact is well-known to METAFONT and METAPOST users. It does not mean, however, that the algorithm is recommended in every situation. Sometimes it is better to use other algorithms and this issue will be briefly surveyed at the end of our discussion.

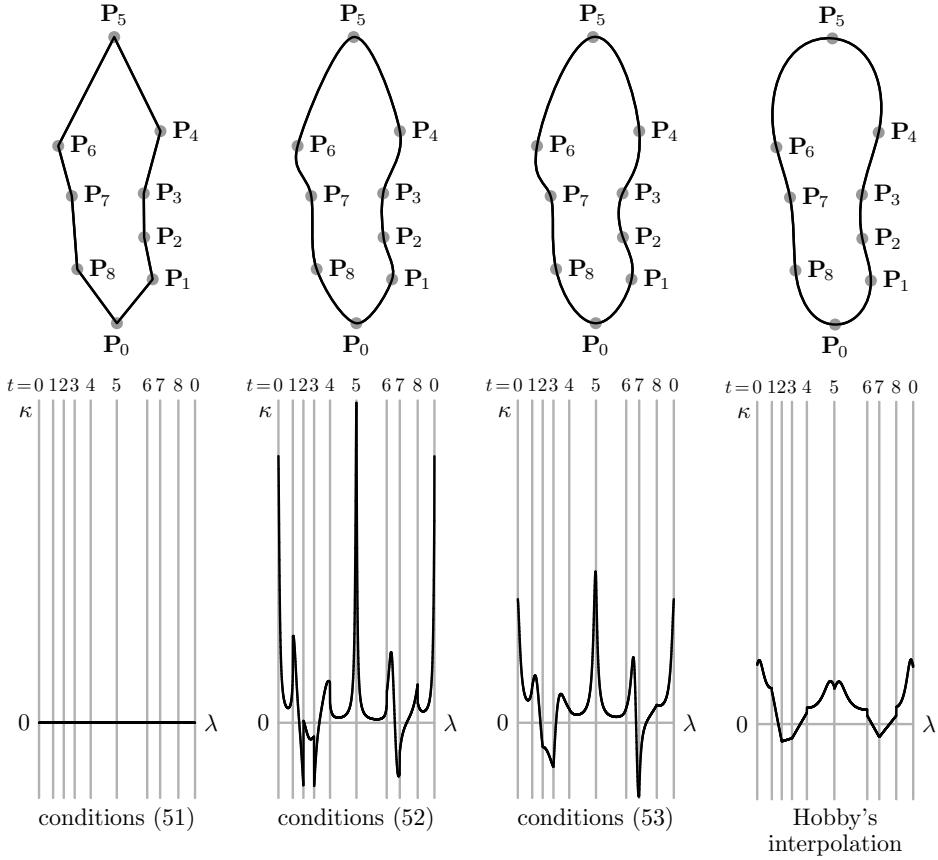


Figure 12: Comparison of interpolation methods for a closed curve; curvature scale κ is common for all the cases; parameter λ , as in figure 7, refers to the length of a path.

Let a series of points $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$, $n > 1$, be given as above and let $\mathbf{P}_k, \mathbf{P}_k^a, \mathbf{P}_k^b$, and \mathbf{P}_{k+1} (a —after, b —before) be the nodes of a Bézier arc based on the chord $\mathbf{P}_k \mathbf{P}_{k+1}$.

The easiest way to connect the points is to use a broken line:

$$\mathbf{P}_k^a = \mathbf{P}_k + \frac{1}{3}(\mathbf{P}_{k+1} - \mathbf{P}_k), \quad \mathbf{P}_k^b = \mathbf{P}_k + \frac{1}{3}(\mathbf{P}_{k-1} - \mathbf{P}_k). \quad (51)$$

METAFONT and METAPOST users do not need to calculate the handles explicitly. Such a connection of points can be obtained by using the macro ‘`--’’, which expands to the following path expression ‘`{curl 1} .. {curl 1}`’. A similar optical effect could be achieved by superimposing conditions $\mathbf{P}_k^a = \mathbf{P}_k$, $\mathbf{P}_k^b = \mathbf{P}_k$; the macro ‘`---’’, which expands to ‘`... tension infinity ...`’, yields a similar result, i.e., $\mathbf{P}_k^a \approx \mathbf{P}_k$, $\mathbf{P}_k^b \approx \mathbf{P}_k$, but the latter two methods, seemingly equivalent, can yield perceptibly different curves.

In order to smooth corners, setting the first derivative explicitly (cf. equation (5)), e.g.,

$$\mathbf{P}_k - \mathbf{P}_k^b = \mathbf{P}_k^a - \mathbf{P}_k = \frac{1}{3} \frac{(\mathbf{P}_{k+1} - \mathbf{P}_{k-1})}{2}, \quad (52)$$

may yield fairly satisfactory results. This is a local method, i.e., a change of coordinates of point \mathbf{P}_k has an impact on the resulting curve only at points \mathbf{P}_{k-2} , \mathbf{P}_{k-1} , \mathbf{P}_{k+1} , and \mathbf{P}_{k+2} .

A non-local method, far less complex than the method described in section 6, involves the assumption of the equality (continuity) of the first and second derivatives at junction points (see formulas (5) and (6)):

$$\mathbf{P}_k - \mathbf{P}_k^b = \mathbf{P}_k^a - \mathbf{P}_k, \quad (53)$$

$$(\mathbf{P}_{k-1}^a - \mathbf{P}_k^b) + (\mathbf{P}_k - \mathbf{P}_k^b) = (\mathbf{P}_k - \mathbf{P}_k^a) + (\mathbf{P}_{k+1}^b - \mathbf{P}_k^a).$$

As already mentioned, Hobby's interpolation usually produces satisfactory results. Figure 12, an example excerpted from [2] and [7], provides a convincing argument. Let us note that the continuity of the first and second derivative (conditions (53)) implies the continuity of curvature (equation (12)) while Hobby's interpolation does not guarantee the continuity of curvature (figure 12, $t = 1, 4, 6, 8$).

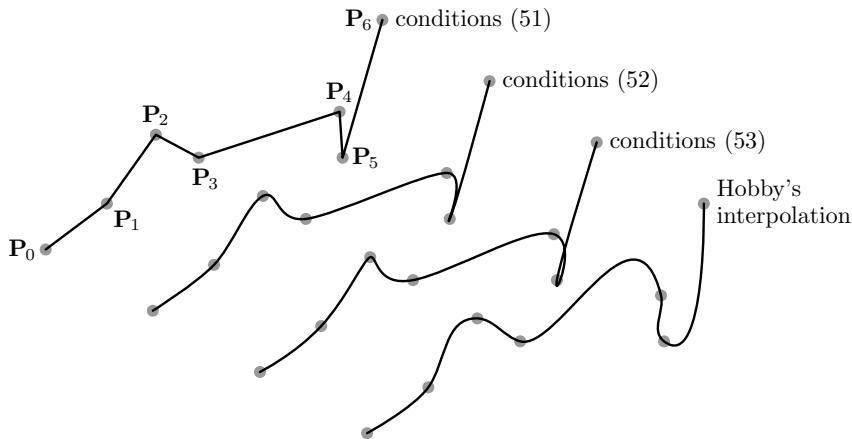


Figure 13: Comparison of interpolation methods for a sample 7-point set of data; an additional boundary condition $\mathbf{P}_0 = \mathbf{P}_0^a$, $\mathbf{P}_6 = \mathbf{P}_6^b$ has been imposed in each case except the broken line interpolation.

Hobby's interpolation exhibits the smallest curvature fluctuation and produces the most regular optical curves in comparison with the other methods. In particular, curves obtained by applying condition (53) do not look as smooth. It turns out that curvature discontinuity is nearly imperceptible unless it is accompanied by a noticeable change of curve direction. (cf. remarks in subsection 4.3). And vice versa, even though curvature is constant for a broken line, namely, equal to zero except nodes where its value is undefined, the eye immediately catches these points because curve direction is not preserved there.

A typical task for which Hobby's method cannot be recommended is the visualisation of empirical data—**METAFONT** and **METAPOST** generate too “rotund” shapes (figure 13). Condition (53) also produces hardly acceptable results in this particular case—the diagram of a function should rather not reveal loops. In such cases, the best approach seems to be using as simple a method as possible, e.g., (51) or (52).

Of course, there are many variants of the methods briefly reviewed in this section. We have not discussed them here at length not because they are not worthy of being applied. On the contrary, it is good to know that the unquestionably excellent Hobby's interpolation algorithm can sometimes successfully be replaced by a simpler one.

8 Acknowledgements

The author very gratefully thanks Alan Hoenig, Daniel Luecking, and John Hobby for reviewing an earlier version of this paper and suggesting several significant improvements, as well as correcting drafting errors.

References

- [1] Arnold, Vladimir I., *On teaching mathematics*, 1997, <http://pauli.uni-muenster.de/~munsteg/arnold.html>, accessed 14.04.2013.

- [2] Hobby, John. D., “Smooth, Easy to Compute Interpolating Splines”, *Discrete and Computational Geometry*, 1986, vol. 1(2), <ftp://db.stanford.edu/pub/cstr/reports/cs/tr/85/1047/CS-TR-85-1047.pdf>, accessed 14.04.2013.
- [3] Joy, Kenneth I., *Bernstein Polynomials*, 2000, <http://www.idav.ucdavis.edu/education/CAGDNotes/CAGDNotes/Bernstein-Polynomials.pdf>, accessed 14.04.2013.
- [4] Knuth, Donald E., *Digital Typography*, CSLI Publications, Stanford, California, 1999.
- [5] Knuth, Donald E., *METAFONT: The Program, Computers & Typesetting, vol. D*, Addison-Wesley, Reading, Massachusetts, 1986.
- [6] Knuth, Donald E., *The METAFONTbook, Computers & Typesetting, vol. C*, Addison-Wesley, Reading, Massachusetts, 1986.
- [7] Manning, J. R., “Continuity Conditions for Spline Curves”, *Computer Journal*, 1974, vol. 17(2), p. 181–186, <http://comjnl.oxfordjournals.org/content/17/2/181.full.pdf>, accessed 14.04.2013.
- [8] http://en.wikipedia.org/wiki/Bernstein_polynomial, accessed 03.06.2013.

◊ Bogusław Jackowski
GUST
Gdańsk, Poland
`b_jackowski (at) gust dot org dot pl`

Representing linguistic pitch in (X)LaTeX

Kevin Donnelly

Abstract

Linguists, especially those working with tone languages, may need to depict pitch levels in the language examples they use. This article looks at some options in LaTeX and XeTeX for representing pitch and tone. The emphasis is on Africanist linguistics, since that is the area with which I am most familiar.

1 Introduction

For extended linguistic work XeTeX is preferable, since it uses UTF-8 natively, and allows fonts to be chosen that will represent all aspects of the language's orthography (whether that is already standardised, or is one of the things that you are working on). However, it is still possible to represent pitch and tone effectively even if you need to use LaTeX, though a different set of tools is required. Notes for both TeX flavours are given here, and sample documents for each are attached—*pitch_latex.tex* and *pitch_xetex.tex*. Everything has been tested on Ubuntu 12.04 running TeX Live 2012 (note that older TeX Live versions may not produce the same output).

2 Setting up the document

Both sample documents assume that you are writing a book, and additional packages or code to enable pitch and tone to be represented are then added.

For LaTeX, the package *tipa* [8] is used, with additional code to allow pitch marks to be printed without sidebars (this is common for East Asian languages, but not for African languages):

```
\makeatletter
\renewcommand{\tonestembar}{%
  \setbox0\hbox{\tipaencoding\char'277}%
  \hbox{\vrule height \ht0 depth \dp0
    width \pt0}}
\makeatother
```

Times is set as the default font:

```
\usepackage{mathptmx}
```

For XeTeX, the package *fontspec* [9] is used. Most (but apparently not all) of the features in *tipa* are now part of *xunicode*, so there is no need to activate *tipa* (doing so produces warnings such as: *Command \supsup already defined*). Unfortunately, without *tipa*, commands like *\textupstep* in XeTeX do not work properly (they appear after the letter they refer to rather than before it), so some workaround code is needed—see Section 3. Moreover, the LaTeX code above to print pitch marks without sidebars will not work (thanks to Alan Munn for pointing this out), since the equivalent glyphs in Unicode can

not be segmented—an alternative will be presented in Section 4.

Charis SIL [10] is selected as the main font. This is a normal text font, like Times, but includes a vast range of glyphs to meet a range of phonetic and orthographic requirements. (This article is typeset in Charis SIL instead of the normal *TUGboat* font in order to enable easier display of text being discussed.)

```
\defaultfontfeatures{Mapping=tex-text,
  Scale=MatchLowercase}
\setmainfont{Charis SIL}
```

The *\defaultfontfeatures* command must precede the actual font selection, and tells *fontspec* to use common TeX aliases (e.g. *--* will produce *–*, and *|* will produce *|* instead of *—*), and to scale all fonts to match the main font's lower-case size. This ensures that any additional fonts selected (see next paragraph) will better match the weight of the main font.

fontspec also allows fonts to be set up for particular purposes. For instance, if we are annotating the syntactic relationships of particular words, and wish to use a separate font for that, in the preamble we could add

```
\newfontfamily{\syntaxfont}{Liberation Sans}
\newcommand{\syn}[1]{\syntaxfont #1}
```

Here, the first line specifies Liberation Sans as the font for syntax annotations in example glosses, and the second line sets up a new command so that

```
\syn{annotation}
```

can be used instead of the more cumbersome

```
{\syntaxfont annotation}
```

For both files, the package *expex* [2] is optional, but recommended because it offers good typesetting of linguistic examples, and allows word-by-word glossing.

3 Tone-marking individual words

The first priority in representing a tone-language is the availability of sufficient glyphs (mostly diacritics) for the tone-marking. It is worth noting that the readability of diacritics (or even whether they are displayed at all) depends crucially on the font—not all are capable of showing all diacritics, or placing them in the right location.

Diacritics commonly used in representing tone include: á à â ã á à á 'á 'à 'â 'ã. This list is not exhaustive—many other diacritics are available.

Many of these diacritics may already be available via your keyboard (for instance, on the default UK keyboard in Ubuntu, á is produced using AltGr + ; then a; à by AltGr + # then a, and â by AltGr + ' then a). If diacritics are to be used frequently, it is worth setting up a keyboard layout to allow this.

In LaTeX, commands can be used to access diacritics—the *Comprehensive LaTeX Symbol List* [6] and Appen-

dix A.3 of the *TIPA Manual* [8] are valuable here. Most of the other diacritics above can be accessed as follows:

```

    à  \v{a}
    á  \={a}
    á  \H{a}
    à  \H*{a}
    à  \textvbaraccent{a}
    'a  \textupstep{a}
    'a  \textdownstep{a}

```

The exceptions are the superscript exclamation marks [!] and [!] (used in African-language linguistics to represent upstep and downstep respectively).

X_ET_EX offers more scope for defining diacritics, since UTF-8 allows diacritics to be combined (rather than having to depend on precomposed glyphs). This means that writing a\char"030D will produce à, because the number for the UTF-8 combining diacritic for the vertical diacritic is 030D—the relevant numbers for each glyph can be found by using a utility like KCharSelect. [3]

However, since it is difficult to remember these character numbers, and they are tedious to type, it is best to set up commands to produce them. Setting up the following commands in the preamble:

```

\newcommand \aup[1]{\char"F19E#1}
\newcommand \adown[1]{\char"F19F#1}

```

will allow us to get the Africanist up/downstep characters by using \aup{a} and \adown{a}: 'a, `a.

In those cases where the L_AT_EX tipa commands do not produce the desired results in X_ET_EX, you can override them—this may also be useful if you have an older text which you are converting, where you wish to minimise the number of textual changes that need to be made:

```

\renewcommand \textupstep[1]{\char"A71B#1}
\renewcommand \textdownstep[1]{\char"A71C#1}

```

This allows \textupstep{a} and \textdownstep{a} to continue to be used to give 'a and `a respectively.

The greater flexibility offered by combining diacritics can also be used to produce new diacritics. For instance, adding an acute and then a macron:

```
\newcommand \hbend[1]{#1\char"0301\char"0304}
```

can yield a diacritic to represent the end of a high-tone bridge: \hbend{a} gives á.

However, not all combining diacritics may be visible to X_ET_EX—in Charis SIL, for instance, glyphs in the ranges 07nn, 1Dnn and 20nn do not give the expected output.

4 Pitch representation for individual words

Inline pitch representation of individual words has been used in the past in African-language linguistics to draw attention to the pitch contours of individual words, so that, for example, the pitches of the kiKongo word **ibuuna** [_ — _] (so) can be shown without drawing premature con-

clusions as to how these pitches should be represented in the tone-marking.

The \tone command can be used for this in L_AT_EX: ibuuna [~\tone{11}\, \tone{5555}\, \tone{11}~] Here, the individual pitches are represented by the number of the tone letter (see A.2.1 of the *TIPA Manual* [8])—five levels are available. Longer pitches can be depicted by repeating the tone numbers, as in \tone{5555}. Glides can be represented by sequences of numbers, and some variation is possible here; as shown in the accompanying file *pitch_latex.tex*, a rising pitch on the last syllable of a variant pre-pausal form of **ibuuna** can be represented using any of \tone{13}, \tone{113}, \tone{1133} and \tone{133}.

The most reliable way of handling spacing between the pitchmarks is to use \hspace{\langle dimen \rangle}, giving measurements in em, mm or pt. For ease of use, this can be set up in a command such as:

```
\newcommand \gap[1]{\hspace{#1mm}}
```

so that a gap of 2mm length can then be inserted by using \gap{2}, and so on.

In X_ET_EX, Charis SIL provides five tone letters similar to those in TIPA, in each of four variants: line or dot with tonestembar on the right or the left (see section 1.4 of *Marking Tone* [7]). Issuing the command

```
\fontspec[Renderer=Graphite, RawFeature=
  {Special=Hide tone contour staves}]
  [Charis SIL]
```

after \begin{document} will remove the tonestembars on glides, but not on level pitches, so these glyphs are inappropriate for African-language linguistics.

However, Charis SIL also provides a set of nine pitch marks (glyphs F1F1–F1F9) specifically aimed at addressing this issue (see section 2 of *Marking Tone* [7]); these allow pitches to be shown inline. To simplify usage, the glyph numbers can be replaced by a \pitch command:

```
\newcommand \pitch[1]{\char"F1F#1}
```

and **ibuuna** [_ — _] can then be produced by writing:

```
ibuuna [~\pitch1\gap{1}\pitch9\pitch9
\gap{1}\pitch1-]
```

repeating the glyph number if longer pitches need to be shown. Inserting the command:

```
\fontspec[Renderer=Graphite]{Charis SIL}
```

after \begin{document} will again allow glides to be represented in a variety of ways: [_ — >] [_ — <] [_ — <—] [_ — >—].

Since these pitch-glyphs are in the Private Use Area of the font (in other words, they are not yet official Unicode glyphs) Charis SIL is the only font that they can be used with.

Mark Wibrow has presented a solution [1, 13] using TikZ [11] that works with both L_AT_EX and X_ET_EX, is

font-independent, and allows a greater variety of pitchmarks. Using this, marking such as *ibuuna* [- — —] can be produced by writing:

```
\tikz[baseline={(0,0.25ex)}]%
\contour[contour only, contour scale=2ex/6,
contour marks={1.55.1}]
{ibuuna};
```

The positioning and size of the inline marking is handled by the `baseline` and `contour scale` directives. The marking itself is input using numbers 1–5 to represent the levels, locating them above each letter that requires a pitchmark; letters not requiring pitchmarks must be designated by a full stop (.)—that is, the number of items in the pitchmark line must be the same as the number of letters, spaces, etc. in the text line. (To do this easily, your editor should be switched temporarily to a monospace font if it is not already using one.)

A variety of glides can be represented: [- — ↗] [- — ↗] [- — ↗] [- — ↗], along with the sort of shape which would be difficult to represent with either of the earlier options, e.g. [↗] [↘]

Since only the pitchmarks are shown, they can be lengthened by duplicating letters in the text. For instance, to get an extra-long mark for the first syllable, use:

```
contour marks={1111.55.,}
{iiiiibuuna};
```

giving [— — ↗]—see also the end of section 7.

5 Representing pitchlevels over word-sequences

Depicting pitchlevels over word-sequences rather than individual words is usually best done using a tiered format. The same general principles as for inline marking apply.

For L^AT_EX, the `tipa` pitchmarks look best when used in conjunction with `expex`, as shown in the accompanying file *pitch_latex.tex*, where the pitchmark line takes the place of a gloss. Note that sequences of more than one word in the text line need to be enclosed in braces, and items in the text line which do not need pitchmarks aligned to them should be marked by empty braces in the gloss (pitchmark) line.

The main drawback to this approach is that trial-and-error is required to place the pitchmarks, experimenting with different \gaps until the marks are aligned with the relevant syllable. The pitchmark line is also hard to edit, since it is unclear to which part of the text line it refers.

For X_ET_EX, the Charis SIL pitchmarks can be used in conjunction with `expex`—as with `tipa`, this requires the same trial-and-error to space the pitchmarks (in the following examples, a raised dot (.) denotes a short pause):

(1) ***ibuuna · basiidi kilumbu***

— — — — — ↗
so · they set aside a day ...

An approach that avoids this uses `pstricks` code by John Frampton, which sets up six pitchmarks (glides can also be specified). The marks are then applied simply by prefacing each vowel with the relevant level, so that writing:

```
\ib\5u\5un\1a · b\1as\5i\5id\4i
k\3il\2umb\pitchup u
```

will give:

(2) ***ibuuna · basiidi kilumbu***
— — — — — — ↗
so · they set aside a day

Mark Wibrow's TikZ solution [13][1] works on both L^AT_EX and X_ET_EX. The correspondence between letters and pitchmarks makes it easy to place and edit the marks while keeping the text easy to read:

```
contour marks={0.55.0....0.55.4..3.2..?}
{ibuuna · basiidi kilumbu};
```

The pitchmarks can be placed above or below the text, depending on the values used for `contour raise`:

ibuuna · basiidi kilumbu
— — — — — — ↗
(3) — — — — — — ↗
so · they set aside a day ...

ibuuna · basiidi kilumbu
— — — — — — ↗
(4) — — — — — — ↗
so · they set aside a day ...

Note that the TikZ solution will not work in L^AT_EX if the text contains characters such as raised dots (.), so this is another argument for using X_ET_EX. Note also that `expex` places its example number at the baseline of the TikZ graphic—there is currently no means of adjusting the vertical placement of the example number.

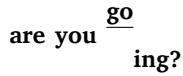
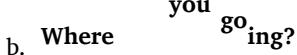
6 Representing pitch contours over phrases

In tone languages, the pitch domain is the syllable, and sections 3–5 have shown various means of representing pitch on the syllable. For pitch-accent and intonation languages, the pitch domain is the word, phrase, or sentence, and this section sets out a method for depicting pitch contours over these longer stretches.

Mark Wibrow's TikZ code [1, 12] allows contours to be represented in a variety of ways in both L^AT_EX and X_ET_EX. In the following examples, an English interrogative sentence (“*Where are you going?*”) will be used, on which two pitch contours will be represented. The first of these (a) is the neutral or default contour, in answer to “*I'm going out now*”—this has high pitch on *where* and *go*. The second contour (b) is a focussed one, in answer to “*I'm going out now, but not to the shops*”, and has high pitch on *are*.

One option is to place the words themselves on different levels, using the tokens follow contour directive.

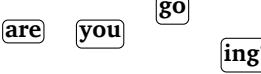
Where

- (5) a. 
are you go ing?
- b. 
 Where are you go ing?

An inline notation is used, where a defined character (by default, the pipe character |, but this can be changed to other characters, e.g. an asterisk) is used to specify the anchor points for the contour, and the height of the contour is specified by a following digit from 0–10 in square brackets:

```
\contour[tokens follow contour]
{|[10]_Where_|[3]are you |[6]_go_|[0]ing?};
```

The stress/high-pitch composite in example (5) has been marked by underlining (generated by a preceding and following underscore). It is also possible to place the words in boxes (box tokens), and expand the spaces between them (space token width), as in this example:

- (6) a. 

A more familiar option is to show the text on a line as usual, with a contour line above it. Example (7) shows this, and also different styles for the contour line (contour /.style)—*thick* and *gray* in (a), and *ultra thick* and *red* in (b):

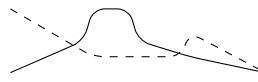
- (7) a. 
Where are you going?
- b. 
 Where are you going?

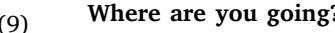
A contour can be displayed by itself, without showing the text line, by using *contour only*:

- (8) a. 

In some cases it may be useful to compare two contours directly. This can be done by drawing both contours, but setting one of them to *contour only*, as in example (9). In this case, the focussed contour is compared to the neutral contour, with the latter styled so that it is drawn with a dashed line. This example also shows how

the gap between the text line and the contour can be specified by using *contour raise*—the gap here is twice that in the previous examples.

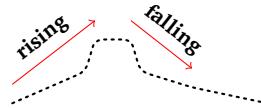


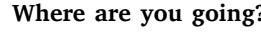
- (9) 

The contour can also be annotated by drawing additional \paths such as the following, part of example (10):

```
\path [draw=red, ->] ([yshift=0.25cm]
  mycontour-1) -- ([yshift=0.25cm]
  mycontour-3) node [midway, sloped, above]
  {\small rising};
```

contour mark prefix is used to give the contour a name (here *mycontour*), and then a red arrow (grayscale for the printed version) is drawn between the relevant anchor points of *mycontour*, counted sequentially (in this case, between anchor points 1 and 3). The *yshift* directive specifies how far above the contour the arrow should be, and the *node* clause specifies the location and size of the annotation “*rising*”. The contour is also styled to appear as a dotted line.



- (10) b. 

7 Replicating early typesetting of tone

Nowadays the conventions of describing tone and intonation are well-established, but at the beginning of the study of the role of pitch in languages even concepts such as “absolute pitch” and “relative pitch” were not well-understood. One of the earliest researchers in this field, Karl Laman, a Swedish missionary to the Kongo, produced the first dictionary of an African language to be tone-marked throughout [5]. His 1922 book, *The Musical Accent or Intonation in the Kongo Language* [4], was one of the first modern attempts to systematise and describe pitch linguistically.

When referring to early research, it may be useful to quote it in its original format, and this may include diacritics and pitchmarks that are no longer widely used, or were even invented solely for that particular piece of research. LATEX and XE^TEX offer possible ways to replicate this type of material.

Laman’s 1922 marking system for the six main pitch-levels he posits for kiKongo (there are a few others) are set out in Table 1. His marking is emulated here by using

stress diacritics (`\textprimstress` and `\textsecstress` from the `tipa` package for L^AT_EX, and Unicode characters 02C8 and 02CC for X_ET_EX, though the `tipa` characters will work in X_ET_EX too), and a super/subscript + sign.

Gradation	Designation	Diacritic
high	very high	a ["]
	high	a'
mid	semi-high	a ⁺
	semi-low	a ₊
low	low	a _l
	very low	a _{ll}

Table 1: Laman's pitch-marking system

For ease of use, commands can be set up to reproduce the marks (which can appear before or after the relevant vowel):

```
\newcommand{\hi}{\textprimstress}
\newcommand{\lo}{\char"02CC\char"02CC}
\newcommand{\shi}{$^{\{\scriptscriptstyle \text{+}\}}$}
```

and so on. This allows Laman's examples to be replicated by inserting the mark shortcut at the appropriate point in the word, as the following examples (from p. 12 of [4], but with inline pitchmarking added) show:

mfu⁺mu [- _] (*king*)
muk,a[']nda [- / _] (*book*)
zo'bongo₊ [- - -] (*antelope*)
ka₊ba [- -] (*to share*)
ba₊ka,la⁺ [- - -] (*man*)
tū⁺la [— —] (*to put*)
di₊nsu,su_l [- - -] (*herbal plant*)

Similar approaches can be used for other markings in [4], and for works by other early researchers, enabling such material, of intrinsic interest to the history of linguistics as well as to descriptive linguistics, to be distributed without great expense.

These examples incidentally demonstrate that more aesthetic placement of the inline pitchmarks discussed in section 4 can be achieved by manipulation of the text line in the `contour marks` directive. For instance, writing out **mukanda** in full

```
contour marks={.1.>..1}]
  {mukanda};]
```

gives **muk,a[']nda** [- / _], which contains unsightly gaps in the inline marking. To deal with this, simply remove excess letters, using:

```
contour marks={1.>.1}]
  {ukana};]
```

to give the more attractive **muk,a[']nda** [- / _].

8 Conclusions

T_EX is a powerful and versatile system for typesetting academic work dealing with pitch, whether relating to tonal

or intonational languages. A variety of diacritics is available, particularly if X_ET_EX is used to give access to a comprehensive font like Charis SIL. Add-on packages such as TikZ/PGF allow diagrammatic representation of pitch levels and contours. Intricate marking systems from early works on pitch can be replicated simply and inexpensively.

9 Acknowledgements

I am grateful to Christina Thiele for comments on earlier versions of this paper.

References

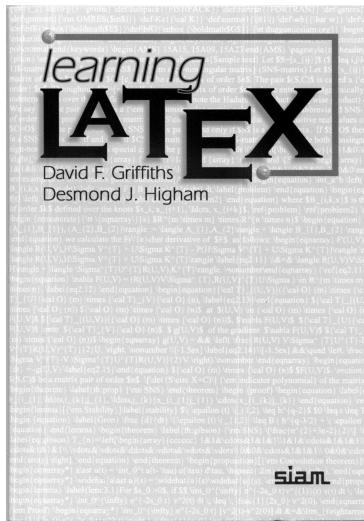
- [1] Kevin Donnelly and Mark Wibrow. TikZ pitch contour, 2013. <http://gitorious.org/tikz-pitch-contour>.
- [2] John Frampton. ExPex, 2012. <http://www.math.neu.edu/ling/tex/expex>.
- [3] KDE. KCharSelect, 2013. <http://utils.kde.org/projects/kcheselect>.
- [4] Karl E. Laman. *The Musical Accent or Intonation in the Kongo Language*. Svenska Missionsförbundet, Stockholm, 1922. Available from the Internet Archive at <http://archive.org/details/musicalaccent00lamarich>.
- [5] Karl E. Laman. *Dictionnaire Kikongo-Français*. IRCB, Brussels, 1936.
- [6] Scott Pakin. The Comprehensive L^AT_EX Symbol List, 2009. <http://www.ctan.org/pkg/comprehensive>.
- [7] Lorna Priest. Marking tone, 2007. http://scripts.sil.org/cms/scripts/render_download.php?format=file&media_id=PitchContours.v1.pdf&filename=PitchContours.v1.pdf via <http://scripts.sil.org/ipahome>.
- [8] Fukui Rei. tipa—Fonts and macros for IPA phonetics characters, 2002. <http://www.ctan.org/pkg/tipa>.
- [9] Will Robertson. fontspec—Advanced font selection in X_ET_EX and Lua_ET_EX, 2013. <http://www.ctan.org/pkg/fontspec>.
- [10] SIL. Charis SIL 4.112, 2011. <http://scripts.sil.org/CharisSIL>.
- [11] Till Tantau. TikZ and PGF, 2010. <http://www.ctan.org/pkg/pgf>.
- [12] Mark Wibrow. Using TikZ to depict intonation, 2013. <http://tex.stackexchange.com/questions/107941/using-tikz-to-depict-intonation>.
- [13] Mark Wibrow. Using TikZ to depict pitchlevel, 2013. <http://tex.stackexchange.com/questions/108530/using-tikz-to-depict-pitchlevel>.

◊ Kevin Donnelly
Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch
Wales
kevin (at) dotmon dot com
<http://kevindonnelly.org.uk>

Book review: *Learning L^AT_EX*

Boris Veytsman

David F. Griffiths and Desmond J. Higham,
Learning L^AT_EX. SIAM, 1997. x+84 pp. Paperback,
 US\$33.00. ISBN 978-0-898713-83-1.



When Karl Berry and I discussed the current situation with L^AT_EX books for beginners, he mentioned the old text by Griffiths and Higham as an example of the one made “just right”. I was surprised to find that the book is still in print. Thus I ordered a copy for myself and read it.

I must say Karl was right: this is indeed an incredibly good introduction to L^AT_EX. Even today, when many good books are available for beginners, this one stands out.

First, it is incredibly short. The great free book by Tobias Oetiker et al. used to be called *L^AT_EX 2 _{ϵ} in 120 minutes*, where 120 referred to the number of printed pages. On my machine `texdoc lshort` now gives me *L^AT_EX 2 _{ϵ} in 157 minutes*, and the number grows every time I update my copy of the T_EX Live distribution. Well, *Learning L^AT_EX* has 53 pages of main text (not counting preface, table of contents, appendices, index and bibliography), and 94 pages total. Of course, one cannot write a L^AT_EX book of this size without making white lies by omission, and the authors admit as much on page 1.

For example, here is how the authors introduce infinitely stretchable lengths. They start with the `\vspace{\langle length\rangle}` command and explain its usage. Then they say, oh, by the way, there is an infinite length `\fill`, so `\vspace{\fill}` will push your text down to the bottom of the page unless counterbalanced by another infinite spring below. Of course

an advanced user knows that there are three infinities in T_EX (`0pt plus 1 fil`, `0pt plus 1 fill` and `0pt plus 1 fill11`), but this might be confusing for a beginner, so the authors’ brief explanation seems to be a smart choice.

Another example of the authors’ unique style is the way floats are introduced. This is rather a “hard” topic, and many beginners have a problem grasping the concept (“I’ve put my table here—why did it travel to the next page?”). The authors explain `tabular`, show how to create the table, and then say, by the way, if you want a caption and number for the table, put everything inside a `table` environment. And by the way, the table will not interrupt your text, but will be placed in some proper place. Oh, and you can do the same with the figures too.

There are numerous short examples, each well written and to the point, often with a distinct sense of humor (e.g. the authors demonstrate spacing after abbreviations with the phrase, *Incomplete lists etc. are signs of lazy writing*).

This writing is easy to read, and may seem easy to write, but this latter impression is misleading. As any experienced author would agree, it is very hard to write “easy”. There is much work and thought behind the seemingly effortless passages. Thus I was not surprised when I read in the preface that this book is based on the lectures and courses the authors taught several times and polished over time.

The book is old, and this shows. It contains a section about L^AT_EX 2.09 and the “new” features of L^AT_EX 2 _{ϵ} ; PDF format is not mentioned at all, as well as such now-popular features as non-CM fonts, hyperlinks, colors, etc. However, one of the strengths of good software is that the knowledge of it does not become obsolete easily. Thus, it is a testament to (L^A)T_EX and the authors’ efforts that this book can still be used as a first L^AT_EX book or as a basis for a short practical course.

It would be also interesting to see a book using the same approach but updated with the new material. Making it as concise and easy to read as this one will be a challenge.

◊ Boris Veytsman
 Systems Biology School and
 Computational Materials
 Science Center, MS 6A2
 George Mason University
 Fairfax, VA 22030
 USA
 borisv (at) lk dot net
<http://borisv.lk.net>

**Book review: Zapf Exhibition:
The Calligraphy of Hermann & Gudrun Zapf**

Boris Veytsman

Minako Sando & Akira Kobayashi, *Zapf Exhibition: The Calligraphy of Hermann & Gudrun Zapf*. Japan Letter Arts Forum, 2011, 64 pp. Paperback, US\$29.95.



The Great East Japan Earthquake of 2011 lead to 15,883 deaths, 6,145 injured, and 2,671 people missing. It caused the meltdown at the Fukushima Daiichi nuclear disaster and incalculable damage to public health and the national economy. The way in which the consequences of the catastrophe were handled shows the amazing resilience of the Japanese people. The destroyed roads, buildings, and bridges are being restored, normal life is resumed. In particular, it is noteworthy that the exhibition of the calligraphy of Hermann and Gudrun Zapf, scheduled for March–April 2011 and postponed due to the catastrophe, opened as early as September of the same year. Only a small slip of paper, tucked in the catalog and correcting the dates, reminds us about the delay.

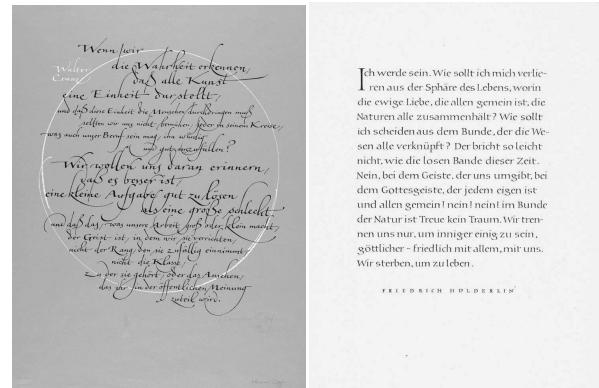
Several copies of this catalog found their way to the US. I bought one at RIT Press.

Of course reading a catalog is a pale substitute for actually visiting an exhibition. Still, the rich illustrations in this book are a real treasure for those who love beautiful letters.

The works of Hermann Zapf are well known to the TeX community. He is a permanent honorary board member of TUG; many of his fonts are included in TeX distributions, and his algorithms influenced the development of modern TeX. Several of Hermann Zapf's books demonstrate his calligraphy, including the beautiful *Alphabet Stories* (reviewed in *TUGboat* 28:2). Gudrun Zapf, a great artist in her own right, is perhaps less known among TeX people—except

for those who have read DEK's *3:16 Bible Texts Illuminated*, who may remember her beautiful rendering of 2 Thess. Thus some pages in the catalog are familiar to the aficionados of calligraphy and type, while others are refreshingly new.

Several illustrations from the catalog can be found at the web page <http://www.typetoken.net/typeface/zapf-exhibition/>. We reproduce below a quotation from Walter Crane by Hermann Zapf and a quotation from Hölderlin by Gudrun Zapf.



The book includes short introductory notes by Minako Sando & Akira Kobayashi and a biography of Hermann and Gudrun Zapf. Minako Sando is the director of the Japan Letter Arts Forum, while Akira Kobayashi is a renowned font designer who worked for many years with both Hermann and Gudrun Zapf.

As an appendix, the catalog reproduces some font design documents and photos of metal type, as if to remind us about the intimate connection between the written word and the printed word.

The book is well printed on beautiful paper. Fittingly, it is typeset in the fonts Optima Nova by Hermann Zapf and Akira Kobayashi, Diotima Classic by Gudrun Zapf and Akira Kobayashi, Hiragino Mincho by Akira Kobayashi and Aldus Nova by Hermann Zapf and Akira Kobayashi. The blending of Japanese and English type is perfect. The colors are reproduced carefully.

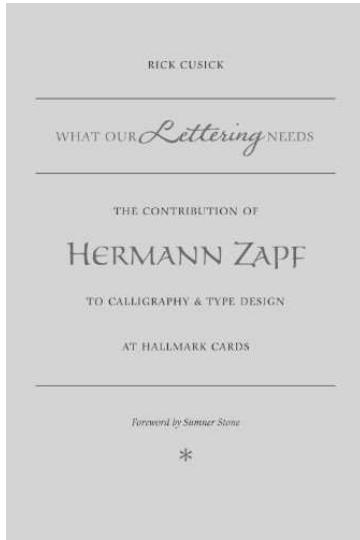
This is a beautiful book and a real gem for a collection of anybody interested in letters, calligraphy and book art.

◊ Boris Veytsman
Systems Biology School and
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
[borisv \(at\) lk dot net](mailto:borisv(at)lk dot net)
<http://borisv.lk.net>

Book review: *What Our Lettering Needs: The Contribution of Hermann Zapf to Calligraphy & Type Design at Hallmark Cards*

William Adams

Rick Cusick, *What Our Lettering Needs: The Contribution of Hermann Zapf to Calligraphy & Type Design at Hallmark Cards*. RIT Cary Graphics Art Press, 2011. 136 pp. Paperback, US\$24.95. ISBN 978-1-933360-55-3.



What Our Lettering Needs by Rick Cusick (with a foreword by Sumner Stone) covers a deceptively narrow topic which would at first glance seem too specific to warrant even this slim, elegant volume. Instead, it addresses a lacuna which has haunted books on graphic design and typography for years—the one line showings of proprietary Hallmark typeface designs which represent the iceberg-like tip of seven years of work by Prof. Hermann Zapf at Hallmark (1966–1973, pg. 95).

While such *Flying Dutchman*-like showings are too many to count, there is also a certain difficulty in enumerating the typefaces themselves due to the passage of time, the combination of one design with another to constitute a third and confused naming (pg. 38). There are all-too-brief samples shown on the two-page spread in the 2001 Zapfest exhibition catalog *Calligraphic Type Design in the Digital Age: An Exhibition in Honor of the Contributions of Hermann and Gudrun Zapf* (pgs. 26 and 27), the author's article which was the precursor to this work.

In this new book, a page or two, or more, is devoted to a dozen of these designs (an informal script, Jeannette; the contemporary Chancery Italic, Firenze; Hallmark Uncial; Hallmark Textura; the text face Crown Roman; an upright calligraphic font with civilité characteristics, Missouri; the unconnected script Scriptura; a calligraphic sans, Shakespeare, by Gudrun Zapf von

Hesse; the combinatorial Stratford, Charlemagne, and Winchester; as well as Constanze, a script alphabet designed by Joachim Romann), showing not just the finished alphabets, but also initial studies as well as examples of usage of the typefaces. None of these, however, are among the 87 typefaces which Hallmark has licensed to Monotype Imaging (<http://www.fonts.com/findfonts/searchresults.htm?kid=hallmark>). Readers however should be certain to compare the pages on Crown Roman with the book *Hunt Roman: The Birth of a Type* by Hermann Zapf and Jack Werner Stauffacher (available online at http://posner.library.cmu.edu/Posner/books/book.cgi?call=744.2_Z35H).

Just as important as the typeface designs is the coverage of some of the technical details of Prof. Zapf's design work, including the “clip and tip” where a swash is drawn on one sheet of paper and then cut out and attached to a letterform in the layout, and many pages from *The Hallmark Lettering Instruction Book* (or *The Manual* as it was called—an entire chapter is devoted to it) are shown in full color as are all the photographs and illustrations. Some such details are very humbling, such as the exquisitely beautiful design study for Zapfino-like capitals intended for use with Firenze shown with the 49-cent Deluxe Fine Point Bic ballpoint pen used to render the letters (pg. 43).

The chronography covers a time in which technology was rapidly changing and the breadth of material covered in each chapter reflects this, making it a wonderful introduction to what design work used to be like. More importantly, the cast of characters reads like a “Who's Who” of the design world, making the book a wonderful *vade mecum* for the designer who needs suggestions on what to read next (be sure to consult the Sources on pg. 109). The book's usefulness as research material is aided by having an index (presumably prepared by Molly Cort, who is credited with “Editorial and Production” in the book's colophon). It stands alone as a reference on the gift book line, Hallmark Editions, and also has a fascinating appendix on The Crabgrass Press and Phil Metzger.

The book has a few shortcomings—too small to show at actual size the larger originals (the pages from *The Manual* especially suffer from this); too short to show all the typefaces in the same detail as the dozen which were featured; some typographic infelicities such as allowing the last word on a page to break (e.g., “nevertheless” from the bottom of pg. 72 to the top of pg. 75); allowing as many as three hyphens in a row; and some odd choices in the formatting, e.g., the usage of a single master page for the page design for much of the book, one with the marginal notes column on the left and the page's folio on the right, which leaves us with the uncomfortable feeling of all compromises.

Nevertheless, all puns aside, this book is what every graphic designer, typographer and calligrapher needs.

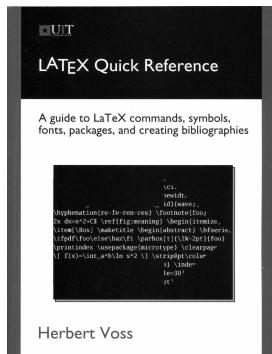
◇ William Adams
willadams (at) aol dot com

William Adams

Book review: *L^AT_EX Quick Reference*

Boris Veytsman

Herbert Voß, *L^AT_EX Quick Reference*.
UIT Cambridge, 2011. vi+234 pp. Paperback,
US\$29.95. ISBN 9781906860219.



A bookcase in my home is filled with dictionaries and reference books. Several times I promised myself to stop buying them: they take up too much space, and it is much easier to look up a definition on the Internet or in an electronic book using search functions. Still I just cannot resist. Thus if Herbert had not given me a copy of this book at the last *TeX* Users Group meeting, I probably would have ended up buying it anyway: this book is a perfect addition for this bookcase.

The author warns the readers this is *not* an introduction to *L^AT_EX*: you cannot use it for self-learning or as a textbook for *L^AT_EX*. Rather, it is a reference book. The difference between a textbook and a reference is the organization of material. In a textbook the topics are introduced in a pedagogical order. A reference book puts the material in the order which is easy to look it up. This is the difference between a phrase book with the chapters like “In a restaurant”, “On a bus” etc., and a dictionary with chapters “A”, “B”, “C”, etc.

While alphabetical order is natural for a dictionary (at least for alphabetic scripts), it is not so for reference books about computer languages like *L^AT_EX*. If the commands are explained alphabetically, then to find the command, say, `\DeclareSymbolFont`, I need first to know its spelling. However, when I need the information I usually do not know the spelling—or even whether such command exists. I probably have just a vague understanding of the task, for example, ‘I want to use the symbols from another font for my math’. Getting the name of the command may be the *result* of my query rather than its start. Therefore most reference books about *TeX* or *L^AT_EX* assume the logical order of organization. For exam-

ple, Victor Eijkhout’s *TeX by Topic* has chapters like *Characters*, *Boxes*, *Grouping*, *Spacing*, etc. *The L^AT_EX Companion* by Mittelbach, Goossens, et al., has chapters like *The Layout of the Page*, *Fonts and Encodings*, *Mastering Floats*, etc.

Unfortunately, *L^AT_EX Quick Reference* uses alphabetical order for most of its chapters. choosing simplicity over meaningful context. For example, pages 16–22 are occupied by a long list of all document classes currently existing on CTAN accompanied with one-line descriptions, sometimes rather cryptic; e.g. `qcm` is described as *multiple choice*. Another chapter is devoted to *L^AT_EX* commands; it consists of three lists, *Environments* (pp. 41–46), *Commands* (pp. 46–67) and *Special Commands* (pp. 67–70). Another chapter has lists of all counters and lengths in core *L^AT_EX*. The entries in the lists are short—sometimes too short. A chapter about fonts has a list of fonts with samples (I liked this list the most simply because I love to study font shapes).

Besides these chapters, reminding one of Umberto Eco’s *The Infinity of Lists*, there are some less dry ones, much better written. There is a chapter about *TeX* binaries with a description of useful auxiliary programs like `pdfcrop`, rarely discussed in *TeX* books. There is a chapter about several selected packages such as `hyperref` or `xcolor` with a nice introduction to these packages. There is a chapter about *BIBTeX* and the newer alternative *biblatex*. Even in these chapters one finds long alphabetical lists (for instance, four pages of `hyperref` options and six pages of `.bst` files), but also finds a useful discussion, covering topics usually not explained elsewhere. These chapters alone probably can justify buying the book.

The book is nicely typeset, printed and bound—although the margins might be a little more generous.

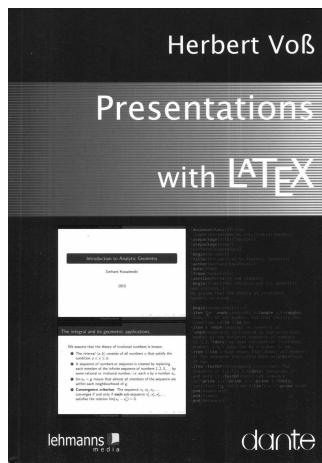
In conclusion, I can imagine a reader thumbing through the book, stopping at an interesting passage or enjoying a font sample. It is less clear to me how to use the book as a reference due to the way the material is organized. Maybe an electronic version of the book with the possibility of full text search would be a better idea. Otherwise I would much prefer a logical grouping of the topics.

◇ Boris Veytsman
Systems Biology School and
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
`borisv(at)lk dot net`
<http://borisv.lk.net>

Book review: *Presentations with L^AT_EX*

Boris Veytsman

Herbert Voß, *Presentations with L^AT_EX*. DANTE e.V., Lehmanns Media, 2012. vi+206 pp. Paperback, €19.95. ISBN 9783865414960.



The early period of L^AT_EX-based presentations may be called a Gothic era. The lack of versatile building materials (even color was not so easy to achieve in those times) led to relatively pristine design with few or no visual effects.

This does not mean that all slides created with the SliT_EX, *seminar* or *foils* classes were necessarily beautiful. Even among Gothic buildings one can find both masterpieces and ugly monsters. (By the way, the designation “Gothic” itself was originally a pejorative: Renaissance architects derisively stated that only uncouth Barbarians like Goths could make medieval buildings.) However, these slides were invariably simple. Presentations were just documents designed for a specific aspect ratio (usually 4:3) with large fonts (usually sans serif) and logos, often repeated on each page.

Meanwhile the purveyors of commercial programs—the ubiquitous PowerPoint being the pack leader—promoted a very different presentation style, with optical effects, complex page transitions and sophisticated overlays. At some point classical L^AT_EX presentations started to appear rather quaint. There was a distinct demand for a more “modern” presentation alternative.

The early efforts in this direction tried to copy PowerPoint’s approach and effects. Even their names, such as *texpower* or *powerdot*, sometimes suggested the connection. However, at some point it became clear that T_EX, being a programming language, can give one almost infinite possibilities, far beyond those provided by non-extensible, end-user-oriented pro-

grams. These possibilities are especially evident in such packages as the current version of *powerdot* and the snazzy *beamer*. Here a user can, for example, typeset a mathematical formula having different terms highlighted and explained on different overlays, animate a sequence of frames, etc.—not even mentioning such common tasks as inserting a movie, or creating complex page transitions. Of course these possibilities give a user plenty of rope to hang himself, overloading the presentation visuals in a way detrimental to the presentation *content*. But in the hands of a good designer with taste and experience they may improve the result. Thus these packages opened the Baroque period of T_EX-based presentations, with its indulgence and richness—and its tendency to over-emphasize effects.

One of the results of this development is the complexity of the interface for the modern presentation packages. The cheat sheet of *foils* commands can be put on a card-sized piece of paper, while the command `texdoc beamer` opens a 245-page document. While the packages have comprehensive manuals, for many people reading such a tome is a formidable task. Such users might prefer books written by somebody on the side of the user rather than the author. Thus books like the present one have a wide audience and an important task to perform.

Herbert Voß describes two packages: *powerdot*, based on PSTRicks, and *beamer*, based on PGF (and initially written by Till Tantau of TikZ/PGF fame). It opens with a short chapter of obligatory advice about slide layout. Most of the material here is more or less known—and some, like the (in)famous one about no more than ten lines per slide would cause protests from the fans of Edward Tufte’s designs.

The strength of the book is in the very well-written chapters about *powerdot* and *beamer*. The explanation here follows the logic of developing a presentation. This makes the book much easier to read. The level of description is detailed, but the overall discussion is not lost in the details, and the reader is always aware of the general plan. The explanation of each package follows more or less the same path: Voß does not try to make two completely different presentation sequences, but neither does he blindly follow the parallel structures when the material does not justify this.

The book has many color illustrations including the comparison tables of slides using different motifs provided by the package authors. This should prove very useful for those lost in the hundreds of different combinations of *beamer* “outer themes”, “inner themes”, “font themes” and color schemes.

Besides “themes”, both *powerdot* and *beamer* provide extensive means to customize the output. However, it seems that many users feel intimidated by this richness. Many presentations at physics meetings look like identical twins: the authors just take the default template. Unfortunately the same is often true for TUG meetings, where *death by PowerPoint* becomes *death by beamer*. Herbert Voß devotes two detailed chapters to customization of *powerdot* and *beamer* respectively. The chapters are well written and will hopefully help the users to make their presentations more original. On the other hand, I dread the day when I see the first L^AT_EX-produced slides typeset in *Comic Sans* (unfortunately the *fontspec* package makes this task too easy, as shown).

There are two bonus chapters in the book. The first one is a concise explanation of color manipulation in L^AT_EX. Of course the *xcolor* package has an extensive manual, but Herbert’s text is much shorter and easier to read, keeping the most used features and omitting the more advanced and exotic ones. The second one, with the slightly misleading title *Questions and Answers*, is devoted to various odds and ends related to the packages and programs used in the book, and contains several useful hints as well as workarounds for some known bugs.

The book was originally published in German. The translation is good, and the couple of remaining examples of German syntax are rather cute; they do not disturb the reading.

The co-publishers, as with all of Voß’s recent books, are the German-speaking T_EX users group DANTE, and Lehmanns Media. As is fitting for a T_EX group-sponsored book, the typesetting is very good. In fact, this is the first of Herbert’s books I’ve read where his name on the cover is spelled as “Voß” rather than “Voss”. The fonts selected for the book, Linux Libertine, Lucida Sans and Bera Mono, are a good match.

This is a useful book which should be of interest to the wide audience of people preparing presentations with T_EX.

- ◊ Boris Veytsman
Systems Biology School and
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
borisv (at) lk dot net
<http://borisv.lk.net>

T_EX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at <http://tug.org/consultants.html>. If you’d like to be listed, please see that web page.

Aicart Martinez, Mercè

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) ono.com](mailto:m.aicart(at)ono.com)
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, L^AT_EX or T_EX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve

PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting(at)dangerouscurve.org)
Web: <http://dangerouscurve.org/tex.html>

We are your macro specialists for T_EX or L^AT_EX fine typography specs beyond those of the average L^AT_EX macro package. If you use X_ET_EX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T_EX and L^AT_EX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T_EX book.

Latchman, David

4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: [technical.designs \(at\) gmail.com](mailto:technical.designs(at)gmail.com)
Web: <http://www.elance.com/s/dlatchman>

Proficient and experienced L^AT_EX typesetter for books, monographs, journals and papers allowing your documents and books to look their possible best especially with regards to technical documents. Graphics/data rendered either using TikZ or Gnuplot. Portfolio available on request.

Peter, Steve

295 N Bridge St.
Somerville, NJ 08876
+1 732 306-6309

Email: [speter \(at\) mac.com](mailto:speter(at)mac.com)

Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of \TeX , I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline \TeX -based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Shanmugam, R.

No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.
Choolaimedu, Choolaimedu, Chennai-600094,
Tamilnadu, India
+91 9841061058

Email: [rshanmugam92 \(at\) gmail.com](mailto:rshanmugam92(at)gmail.com)

As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices. I provide consultation in building business models & technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for nearly 20 years, and handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in \TeX , $\text{\LaTeX}2\epsilon$, \XeLaTeX , Quark, InDesign, XML, MathML, eBooks, ePub, Mobi, iBooks, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sievers, Martin

Klaus-Kordel-Str. 8, 54296 Trier, Germany
+49 651 4936567-0
Email: [info \(at\) schoenerpublizieren.com](mailto:info(at)schoenerpublizieren.com)

Web: <http://www.schoenerpublizieren.com>

As a mathematician with ten years of typesetting experience I offer \TeX and \LaTeX services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents.

From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (\BIBTeX , \biblatex) to typesetting your math, tables or graphics—just contact me with information on your project.

Sofka, Michael

8 Providence St.
Albany, NY 12203
+1 518 331-3457

Email: [michael.sofka \(at\) gmail.com](mailto:michael.sofka(at)gmail.com)

Skilled, personalized \TeX and \LaTeX consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in \TeX and \LaTeX : Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in \TeX or \LaTeX ; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized \TeX or \LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris

46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: [borisv \(at\) lk.net](mailto:borisv(at)lk.net)

Web: <http://www.borisv.lk.net>

\TeX and \LaTeX consulting, training and seminars. Integration with databases, automated document preparation, custom \LaTeX packages, conversions and much more. I have about seventeen years of experience in \TeX and thirty years of experience in teaching & training. I have authored several packages on CTAN, published papers in \TeX related journals, and conducted several workshops on \TeX and related subjects.

Young, Lee A.

127 Kingfisher Lane
Mills River, NC 28759
+1 828 435-0525
Email: [leeayoung \(at\) morrisbb.net](mailto:leeayoung(at)morrisbb.net)

Web: <http://www.latexcopyeditor.net>
<http://www.editingscience.net>

Copyediting your $.tex$ manuscript for readability and mathematical style by a Harvard Ph.D. Your $.tex$ file won't compile? Send it to me for repair. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, \LaTeX manuscripts for the Physical Review; career as professional, published physicist.

Die TeXnische Komödie 2/2013

Die TeXnische Komödie is the journal of DANTE e.V., the German-language TeX user group (<http://www.dante.de>). [Non-technical items are omitted.]

CHRISTINA MÖLLER, Beschlüsse der 48. Mitgliederversammlung [Spring Meeting and 48th General Meeting of DANTE e.V.]; pp. 6–13

After last year's Autumn meeting, jointly held as EuroTeX in Breskens, Netherlands, the Spring meeting took place in Gießen. From March 5–8 the TeX community came together to listen to talks from various areas, discuss problems and exchange ideas.

CHRISTINE RÖMER, Handouts setzen [Typesetting handouts]; pp. 27–36

This short article meets the wishes of TeX users for templates for specific kinds of text.

ROLF NIEPRASCHK, HERBERT VOSS, Test auf eine Zahl [Testing for a number]; pp. 37–39

In TeX applications it is sometimes necessary to check an argument for its properties. In this article we show how to test an argument for an integer value.

CHRISTINE RÖMER, Logikzeichen [Logic symbols]; pp. 40–55

With the formal markup language of mathematical logic, special symbols were introduced that are also used in linguistic formal semantics and philosophy of language. Since there is only one package available to typeset Frege's *Begriffsschrift* from 1879, this article shows a list of the special symbols with their LATEX syntax. For those working with these symbols this may be helpful.

[Received from Herbert Voß.]

ArsTeXnica #15 (April 2013)

ArsTeXnica is the journal of GJT, the Italian TeX user group (<http://www.guitex.org>).

GIANLUCA PIGNALBERI, Editoriale [From the editor]; p. 3

CLAUDIO FIANDRINO, Drawing ER diagrams with TikZ; pp. 5–14

The paper will illustrate some techniques to represent Entity-Relationship (ER) diagrams with TikZ. In particular, it will focus on the standard internal library `er`, on the external package `TikZ-er2`, on the external tool `Graphviz` and on the object-oriented approach provided by the `er-oo` library.

CLAUDIO FIANDRINO, Sa-TikZ: A library to draw switching architectures; pp. 15–24

The article illustrates how it is possible to draw

some types of switching architectures in a simple manner. The Sa-TikZ library provides not only the keys devoted to the drawing part, but also the keys devoted to customizing aspects of the architectures in the spirit of normal TikZ syntax.

GIANLUCA PIGNALBERI, Realizzazione di un layout complesso: la prima pagina de *La Settimana Enigmistica* [How to design a complex layout: The first page of *La Settimana Enigmistica*]; pp. 25–30

Making a complex page layout with LATEX involves the use of multiple specialized packages. We will see how to use TikZ, `shapepar` and `textpos` while attempting to reproduce part of the cover of an old issue of *La Settimana Enigmistica*.

MASSIMILIANO DOMINICI, Una panoramica su Pandoc [An overview of Pandoc]; pp. 31–38

This paper is an overview of Pandoc, a utility for the conversion of Markdown-formatted text in many output formats, including LATEX and HTML.

CLAUDIO BECCARI, LATEX e le lingue romanze alpine: il friulano [LATEX and Alpine Romance languages: Friulan]; pp. 39–45

LATEX is used to typeset in a variety of languages: at the time of writing, it can handle 74 different languages (plus many variants), some of them used by very few people, some by many millions. At the moment, there are no facilities for typesetting documents in the various more or less official Alpine Romance languages. This paper would be another step toward extending the LATEX language capabilities to the various Romance languages that are used by large communities in the European Alps. In this second article on the subject we deal with Friulan, spoken, read, and written by almost 800,000 people in northeastern Italy and within other Friulian communities around the world.

CLAUDIO BECCARI, Le filigrane e le figure di sfondo [Watermarks and background figures]; pp. 46–58

Our TeX system has several packages available for inserting watermarks or background images. But each one of these packages has specific features that make it more suitable in certain applications rather than others. In this article the mechanism for inserting such “decorations” is thoroughly examined so as to fully understand how it works.

LUIGI SCARSO, LuaJITTeX; pp. 59–69
[Published in *TUGboat* 34:1.]

[Received from Gianluca Pignalberi.]

EuroBachoTeX 2013 proceedings

The EuroBachoTeX 2013 proceedings was published by GUST, the Polish language TeX user group. Their web site is <http://www.gust.org.pl>.

BOGUSŁAW JACKOWSKI, and MAREK RYĆKO, Two typographic etudes for basic calculus and implementation; pp. 5–8

Two simple techniques for creating certain Bézier curves will be presented. Although mathematically nearly trivial, they are potentially useful, especially when implemented as an interactive option.

BOGUSŁAW JACKOWSKI, ŁUKASZ DZIEDZIC and MAREK RYĆKO, Joining two Bézier arcs smoothly; pp. 9–12

BOGUSŁAW JACKOWSKI, ŁUKASZ DZIEDZIC and MAREK RYĆKO, Constructing a family of constrained (metrically consistent) Bézier arcs; pp. 13–16

(Combined and expanded version published in this issue of *TUGboat*.)

HANS HAGEN, Speed performance in ConTeXt; pp. 17–24

In the ‘mk’ and *hybrid* progress reports I have written some words on speed. Why is speed still important today?

HANS HAGEN, Does TeX have a future?; pp. 25–29
(Published in this issue of *TUGboat*.)

HANS HAGEN, SwigLib basics; pp. 30–32

SwigLib aims to add portable library support to LuaTeX. It does not provide Lua code except where absolutely required, since different macro packages have different needs. It also fits in the spirit of TeX and Lua to minimize core components.

LUIGI SCARSO, The *swiglib* project; pp. 33–36

This project has the following objectives:

- Set up an infrastructure for building libraries for LuaTeX using Swig so that we stay close to the original APIs.
- Investigate libraries of different complexity. We also explore the impact of dependencies on other libraries.
- Provide a set of common helpers that can be integrated in libraries.
- Provide documentation on how to use this infrastructure and how to roll out your own.
- Come up with a naming scheme to allow avoiding clashes between similar libraries. In principle a user or macro package should be able to generate libraries independently of others and use these in a regular TeX setup.

- Make a couple of working examples of libraries, available for all major platforms.

The project will be initially hosted at <https://github.com/luigiScarso/swiglib>.

PAWEŁ ŁUPKOWSKI and MARIUSZ URBAŃSKI, Preparing for scientific conference with LATEX. A short practical how-to.; pp. 37–44

(Published in this issue of *TUGboat*.)

PAWEŁ ŁUPKOWSKI, How to teach LATEX? Cognitive science curriculum case study; pp. 44–47

In this paper I will present my experience teaching LATEX in an introductory information technology course. I will present my syllabus for the course and the idea of embedding LATEX skills into other subjects in the cognitive science curriculum. I will also compare the results of the LATEX course evaluation for its four editions (2010, 2011, 2012 and 2013).

BARTOSZ MARCINIĄK, Converting LATEX source files into XML format with XQuery; pp. 48–50

The XQuery language, designed for querying XML documents, in practice allows performing any transformation of XML document sets. It also allows performing transformations of data in formats which can be formulated as XML, e.g., CSV files.

A source LATEX file is also such a format. Its structure and instructions should be describable with a set of XML tags. In this presentation an attempt to use XQuery for a transformation of a LATEX document into an XML document will be presented.

PRZEMYSŁAW SCHERWENTKE, Translating into the Pokémon language; pp. 51–53

According to one of its definitions, the Pokémon language is a dialect of the Polish language used by the majority of teenage girls or simply Pokémons. According to Nonsensopedia (the Polish encyclopedia of humor, http://nonsensopedia.wikia.com/wiki/Pokemoniaste_pismo): “That language is often regarded as a script of the feeble minded and cretins but in reality comprises an ingenious cipher which in itself is not difficult to crack but a normal person encountering such a cipher loses interest in trying to solve the riddle of the Pokémon language ... We will present a (TeX, of course) tool which allows ordinary people to translate normal language utterings into the Pokémon language.

DAVID KASTRUP, Extension language integration of LuaTeX and LilyPond; pp. 54–58

LuaTeX uses Lua as its extension language while the music typesetter LilyPond employs the Scheme dialect Guile for that purpose.

It is interesting to see how those extension languages are integrated into the “core” language and

what interfaces are used for passing information back and forth between user, principal language, and extension languages and to what degree the languages interact to form a coherent experience or one more modelled along the line of “I’d rather like to discuss this with your brain surgeon”.

KEES VAN DER LAAN, Spirals in PostScript;
pp. 59–66

Programming curves specified in polar coordinates work elegantly in PostScript, because of PS’s facility for rotation of user space. This is shown for the cardioid, limaçon, lemniscate, Archimedes, and growth spirals. The Gyre logo is analysed and imitated in PostScript. Printing of text along spiral-like belts on a sphere in the projection plane is done, yielding poor man’s typesetting on a sphere in projection, for want of better.

KEES VAN DER LAAN, Head and tail in summation: Catching up with numerical math and Mathematica; pp. 67–74

Direct summation of slowly convergent series is not efficient. Splitting up the sum into head and infinite tail and applying Euler summation to the tail yields an efficient technique. Applying Boole’s transformation to the tail may, for slowly convergent alternating series, drastically improve accuracy. Series for $\zeta(x)$, $\eta(x)$, $\lambda(x)$, and $\beta(x)$ have been included. This note reflects my understanding of chapter 1 of Nico Temme’s book *Special Functions* (Wiley, 1996), and is aimed at those who want to refresh their summation-of-series knowledge and skills and add Mathematica to their toolbox. Mathematica has changed calculus education. Notebook publishing is an extra to (L)TeX publishing, for the moment.

JEAN-MICHEL HUFFLEN, MIBIBTeX in 2013: The point; pp. 75–78

Our MIBIBTeX program—aiming to be a better BIBTeX—is now able to build bibliographies for documents in LATEX. It can also take into account some particular features for the bibliographies processed by the biblatex package and the bib module of ConTeXt. We review the present abilities of this program before an important change in its implementation language (now Scheme). We will also explain why this change could enlarge MIBIBTeX’s features.

JEAN-MICHEL HUFFLEN, Why typesetting music is so difficult; pp. 79–84

Some software allows users to specify musical pieces, possibly using several staves for different voices and instruments. These pieces can be typeset in order to get a result suitable for musicians. For simple cases, this result is very nice. However,

typesetting a musical score is not comparable to typesetting a text. Also, some meta-information is needed in order to typeset correctly some pieces, especially in baroque music. We propose an exploration of these difficulties. Attending this talk only requires basic knowledge about music and scores.

JEAN-MICHEL HUFFLEN, XML today: Success or failure?; pp. 85–94

XML came out in the late 1990s as a possible standard for information interchange between diverse programs. What is its point in 2013? Has XML completely eclipsed its predecessor, SGML? What is XML’s place within the world of the Web and within programming activity?

JEAN-MICHEL HUFFLEN, BachoTeX song;
pp. 95–98

A printed score of this renowned tune.

PAWEŁ JACKOWSKI, TeX beauties and oddities;
pp. 99–105

Collected TeX pearls, with contributions from Bogusław Jackowski, Gunter Essers, Jerzy Ludwiczowski, Piotr Strzelczyk, and Marcin Borkowski.

Abstracts; pp. 106–110

The remaining abstracts had no corresponding papers submitted.

KAVEH BAZARGAN, How TeX helps deliver XML-first production to journal publishers

Almost all publishers of academic journals require a granular XML of each article, along with PDF, HTML, and other formats. The XML is the archive of the article, so it is imperative that the content of all formats match precisely. I will show how TeX can be used both to generate XML from the author manuscript, and to convert the XML into PDF and other deliverables.

PIOTR BOLEK, The traditional vs. the future book

What do the tablet revolution and the growing popularity of e-readers mean for the books, publishing industry and readers? A digital book, audiobook, multibook or just an application? A multimedia publication—only a toy, a manual or something more?

Examples of publications, ideas and new opportunities. The book of future and its production—technologies, standards, formats and tools.

Digital distribution vs. intellectual property and access to digital content. Models of distribution and access to content. Contemporary and informal distribution channels.

KATARZYNA BURAKOWSKA, Communication outside of words. Meta-communication.

Research showed that words of a message account only for 7% of the perceived value of a spoken message, while the tone of voice accounts for 38% and facial expression for up to 55%. Thus the non-verbal elements of verbal communication are very important for its efficiency. There is a lot of research regarding importance of meta messages in verbal communication. This knowledge is being used in creative ways by theatre artists. I haven't come across research evaluating meta-messages in written communication, though graphics artists use them very often in their works. I would like to share my thoughts regarding that issue. I'm not a typographer, but do work in visual arts, so my presentation will be, as usual, a little bit off the main conference subject.

WILLI EGGER, Workshop: Bookbinding

Based on last year's success there will be a workshop in bookbinding. This time we are going to build a sturdy *shoe-box* type of box. While waiting for the glued box to dry we will make other types of boxes, small and suitable for gift-packaging.

HANS HAGEN, Those typographic things \TeX ies are proud of ... do they really make sense?

Why do we use \TeX ? Is it because we have no other choice? Is it because we like to program? Do we go for the looks? It is no problem to locate users who, no matter how they started, praise the virtues of this typesetting system. Isn't it one of the reasons why we meet at Bacho \TeX ? How valid are these sentiments? Does all this focusing on details makes sense or not? What are those features that we like so much and do they really make that much sense?

HANS HAGEN, Bits and pieces: Con \TeX Xt, MetaPost, Lua and more, part 1

Last year Con \TeX Xt MkIV became a bit more what I had in mind when we moved to Lua \TeX . I will give a quick overview of what has been (re)done, extended, finalized, set in motion and what might happen.

HANS HAGEN, Bits and pieces: Con \TeX Xt, MetaPost, Lua and more, part 2

Because we want our machinery to do more and more, performance becomes an issue, especially in workflows where there is lots of output. I will discuss some aspects of performance as well as some experiments that Luigi Scarso and I did in the process of getting LuaJIT \TeX up and running.

HANS HAGEN, Lua \TeX tutorial

Participants will learn at least: what Lua \TeX is really; for whom Lua \TeX is useful; the prerequi-

sites to using it; what callbacks are; how to use the Lua \TeX reference manual.

Last but not least several examples will be discussed in detail, so that the basic mechanisms can sink in. Handouts will be provided.

HANS HAGEN and FRANS GODDIJN, Books will go ... are you sure?

In the Netherlands (and probably elsewhere too) there is a web shop where you can buy designer knitware produced by grannies. Apart from the social aspects, this new business model might as well translate to producing books.

If you go to a bookshop you will notice that the kids corner still offers lots of books, and surprisingly, many of these are well designed (and definitely better bound than those for grown-ups).

In this session some old folks will discuss the future of books and design with you from this perspective. Please bring with you, your favourite books from childhood (the ones that impressed you) or nice ones that you gave friends and family. Of course kids are invited to join in.

ALEKSANDRA HANKUS, The end of the world will come. Books will go.

We have been debating for many years during our conferences whether a "paper" book will survive. The title of my talk may suggest the debate to be continued. Perhaps... Still, the talk will concern the XIX century in the first place. The time which people sensitive to beauty would like, in my opinion, to move to. I will attempt to show a journal (a weekly) from those years. A weekly issued (typeset) in such a way that it completely took away my willingness to buy printed stuff nowadays.

RYSZARD KUBIAK, A personal view of markup languages

A markup language is a notation in which a textual document can be written down in order to later give it an elegant graphical form by a computer. Many such languages have been and are continually being designed, the language of \TeX being one of them. The creators of markup languages take into account various aspects of human-computer and human-human communication. The talk will be about my personal views and experience of using various languages.

BOGUSŁAW JACKOWSKI, PIOTR STRZELCZYK and PIOTR PIANOWSKI, On the progress of the \TeX Gyre Math project: The TG Bonum Math font

Two fonts—TG Pagella Math and TG Termes Math—have been released so far within the frame of the \TeX Gyre math project. Currently,

the TG Bonum Math font is under preparation; we will present the current state of the work. The remaining font, TG Schola Math, will, hopefully, be (pre)released by the end of 2013.

PAWEŁ ŁUPKOWSKI, A poster: Online L^AT_EX editors—fancy toys or usable tools?

I review several L^AT_EX editors available online. I will pay attention to the range of offered packages and compilation options available. I will also take a closer look at options of integration with other services (like Dropbox) offered by the editors. In addition, mobile solutions will be described.

ARTHUR REUTENAUER, Polyglossia update

Polyglossia was created five years ago as the X_ET_EX-aware replacement of Babel, whose development had come to a halt. Aiming at providing what its predecessor had done two decades ago for then-existing variants of T_EX, it has grown to support over 70 languages. For many languages it relies heavily on `fontspec`, whose extensive font-handling capabilities are essential, and which several years ago was made to support LuaT_EX. However, Polyglossia itself had notably left LuaT_EX aside until now.

Today, work has started on experimental support for LuaT_EX, and work on Babel has also thankfully resumed. I will discuss the relationship between both packages, and plans for Polyglossia's future in the ever-evolving world of T_EX.

ARTHUR REUTENAUER, Behaviour-driven development for T_EX

Behaviour-driven development is a software development process whose central idea is that in order to write any computer program, one should first specify how its different parts should behave, and only then start implementing them. It builds on a slightly older method called test-driven development, whose main tenet is to write tests before the code. Behaviour-driven development thus recommends to not only write tests and specifications beforehand, but also to conduct an analysis of what the different parts should do, and to let that analysis drive the development workflow.

This is evidently a very different approach from the one we usually use for writing packages and macros, but, having used it for a couple of years in the industry, I would like to introduce it to the T_EX community and to explore options and ideas from these areas that would, in my humble opinion, benefit us.

BARBARA WILIŃSKA, Workshop: Painting initials

You'll be able to paint your own initial, little by little, with your teacher's trifling help, but still all by yourself.

Inspirations may be found at <http://bancroft.berkeley.edu/digitalscriptorium/>, <http://www.enluminures.culture.fr/documentation/enlumine/fr/>, or, perhaps, in your family's collections ...

ANDRZEJ TOMASZEWSKI, On readability of script and print

An attempt to define the underlying notions and present the area of interest of the research community. A survey of some research on readability (during the twentieth century), initially on reading hygiene and sight protection, with a later focus on improving perception. The figure of Miles Albert Tinker, the leading researcher on print readability. How those problems were seen in Poland.

ZOFIA WALCZAK, Spring cleaning in the garden—grafting L^AT_EX

Spring in the garden. We are trying to do everything in order to have a good harvest. And how about L^AT_EX? Are we doing enough for the people who don't know L^AT_EX yet? I will ask many questions and give a few answers.

MARCIN WOLIŃSKI and ADAM TWARDOCH, Designing a scientific journal following the example of *Journal of Language Modelling*

Journal of Language Modelling (<http://jlm.ipipan.waw.pl>) is a new, free (no publication fees) open-access journal. All content is available under a Creative Commons licence.

We will talk about the design process of the journal's layout and its implementation as a X_ET_EX document class.

[Received from Jerzy Ludwichowski.]

TUG Business

TUG 2013 election report

Nominations for TUG President and the Board of Directors in 2013 have been received and validated. Because there is a single nomination for the office of President, and because there are not more nominations for the Board of Directors than there are open seats, there will be no requirement for a ballot in this election.

For President, Steve Peter was nominated. As there were no other nominees, he is duly elected and will serve for two years.

For the Board of Directors, the following individuals were nominated:

Kaja Christiansen, Steve Grathwohl, Jim Hefferon, Klaus Höppner, Arthur Reutenauer, David Walden. As there were not more nominations than open positions, all the nominees are duly elected for the usual four-year term. Thanks to all for their willingness to serve.

Terms for both President and members of the Board of Directors will begin with the Annual Meeting. Congratulations to all.

Board member Jonathan Fine has decided to step down at the end of his term. On behalf of the Board, I wish to thank him for his service, and for his continued participation until the Annual Meeting.

Statements for all the candidates, both for President and for the Board, are appended (in alphabetical order). They are also available online at the url below, along with announcements and results of previous elections.

- ◊ Barbara Beeton
for the Elections Committee
<http://tug.org/election>

Kaja Christiansen



Biography:

I live in the city of Aarhus, Denmark and work at the University of Aarhus. My job at the University involves system administration of Unix machines at the University as well as software, including the responsibility for up-to-date TeX & friends suite.

I heard about TeX for the first time in the fall of 1979. In Palo Alto at the time, I wanted to audit courses at Stanford and my top priority was lectures by Prof. Donald Knuth. That, I was told, was not possible as Prof. Knuth was on leave due to work on a text processing project... This project was TeX! Back home, it didn't take long till we had a runnable TeX system in Denmark.

Personal statement:

I have served as a Board member since 1997, as the chair of TUG's Technical Council since 1999, co-sponsored the creation of the TeX Development Fund and served as TUG vice-president from 2003–2011. I share system administrator's responsibilities for the TUG server (which access to the Internet is currently facilitated by my University). In my rôle as a member of the board, my special interests have been projects of immediate value to the TeX community: TeX Live, TUGboat and TUG's web site. During the years 2002–2011 I served as the president of the Danish TeX Users Group (DK-TUG).

Steve Grathwohl



Biography:

I have used TeX since 1986, first as a hobby, then "professionally" after I joined Duke University Press in 1993 on the staff of the *Duke Mathematical Journal*. Eventually I supervised the production of the journal (for both print and online incarnations), and I wrote and maintained the class files for typesetting. Since 2005 I have been responsible for loading content for our 50 journals and monographs onto multiple platforms as well as being TeXnical liaison for Duke to Project Euclid, a hosting service for over 50 independent mathematics journals. My current work involves a significant amount of work with XML content and metadata schemas as well as being the in-house TeX specialist.

Personal statement:

TeX has proved to be an astoundingly robust piece of software, and the continuing development of projects like LATEX3, LuaTeX and XeTeX helps ensure TeX's vitality into the future. I would like to see the TUG board continue to support these and

others (like TeX Gyre and TeXworks) that contribute to a 21st-century TeX.

Jim Hefferon



I have enjoyed working on the Board, trying to promote the interests of TeX and friends. In the future I would like to continue to do so, trying to balance fiscal prudence with taking the opportunities that arise.

Klaus Höppner



Biography:

I got a PhD in Physics in 1997. After several years in the Control Systems group of an accelerator center in Darmstadt, I've been working at an accelerator for cancer therapy in Heidelberg. My first contact to L^AT_EX was in 1991, using it frequently since then.

I was preparing the CTAN snapshot on CD, distributed to the members of many user groups, from 1999 until 2002. I was heavily involved in the organization of several DANTE conferences and EuroTeX 2005. Since 2000, I am a member of the DANTE board, some years acting as president or vice president, now as treasurer.

Personal statement:

In the years since Karl Berry's presidency the cooperation of TUG and European user groups improved a lot. My candidacy is in the hopes of helping to continue this trend. Projects like TeX Live and CTAN owe their success to the work of active volunteers, but also to the support and cooperation of the user groups.

Steve Peter



Biography:

I am a linguist and publisher originally from Illinois, but now living in New Jersey. I first encountered TeX as a technical writer documenting Mathematica. Now I use TeX and friends for a majority of my publishing work, and work with several publishers customizing TeX-based publishing systems. I am especially interested in multilingual typography and finding a sane way to typeset all of those crazy symbolisms linguists create. As if that weren't bad enough, I also design typefaces. (Do I know lucrative markets, or what?)

I got involved in TUG via translations for *TUGboat*, where I also work on the production team. I was on the TUG board of directors for several terms before becoming TUG president in 2011.

Personal statement:

The future of TeX and TUG lies in global communication and cooperation to promote and sustain the amazing typographic quality associated with TeX and friends. Projects such as LuaTeX show that there remains a dynamic and bright future for our preferred typesetting system. I am especially interested in having TUG support projects (technical and artistic) that will serve to bolster TeX and TUG's visibility in the world at large.

Arthur Reutenauer



Biography:

I have been using TeX for the past 15 years, first as a mathematics student then as a language enthusiast. Having been president of GUTenberg, the French-speaking TeX user group, for one term, I have contributed to founding the ConTeXt Group to help with development of that part of TeX community. I am currently the maintainer of Polyglossia.

Personal statement:

Being about as old as TeX, I have come to it in times where it was already mature but, maybe, not always up to date with the most recent developments in computer typesetting. This situation has however

recently improved with the advent of Xe_TE_X and LuaT_E_X amongst others, and I do believe that T_E_X still has a great potential to produce the best typeset documents, in which TUG can play its part.

David Walden



Biography:

I was supposed to be studying math as an undergraduate at San Francisco State College; but, from my junior year I was hacking on the school's IBM 1620 computer. While working as a computer programmer at MIT's Lincoln Laboratory, I did the course work for a master's degree in computer science at MIT. Most of my career was at Bolt Beranek and Newman Inc. (BBN) in Cambridge, Massachusetts, where I was, in turn, a computer programmer, technical manager, and general manager. At BBN, I had the good fortune to be part of BBN's small ARPANET development team. Later I was involved in a variety of high tech professional services and product businesses, working in a variety of roles (technical, operations, business, and customer oriented).

Throughout my business career and now during my so-called retirement years, I have always done considerable writing and editing. This led to my involvement since the late 1990s with T_E_X, becoming a member of TUG, and eventually as a TUG volunteer. I have served as a member of the TUG Board since 2005 and also served in the role of Treasurer (I know bookkeeping from my business career).

I helped create *The PracT_E_X Journal*, doing its initial website development; I founded TUG's Interview Corner; I have helped behind the scenes with the *TUGboat* web site; and I was the "local organizer" person on the program committee for TUG's 2012 annual conference in Boston.

More personally, I use L_AT_E_X and other tools in the "arsenal" of T_E_X and friends all the time, for example:

- to write and publish books (<http://walden-family.com/public/mybooks>), including two books created for TUG in collaboration with Karl Berry
- to write numerous articles, some of which are related to T_E_X (<http://walden-family.com/texland>)

You can learn more about me at <http://www.walden-family.com> and at <http://www.tug.org/interviews/walden.html>.

Personal statement:

I am interested in continuing to serve on the TUG Board:

1. To continue to serve the community that has so generously served me via `comp.text.tex`, CTAN, *TUGboat*, etc.
2. As a way of explicitly contributing to maintaining the viability for years to come of T_E_X and the T_E_X world, entities I believe are "world treasures".

As a TUG Board member, my frame of mind has been to get things done quickly and pragmatically with enough generality so evolution is possible.

Calendar

2013

- Jul 21–25 SIGGRAPH 2013, “Left Brain + Right Brain”, Anaheim, California.
s2013.siggraph.org
- Aug 5–9 Balisage: The Markup Conference, Montréal, Canada. www.balisage.net
- Aug 21–25 TypeCon 2013: “Portl&”, Portland, Oregon. www.typecon.com
- Sep 9 **TUG 2013** preprints deadline.
tug.org/tug2013
- Sep 10–13 ACM Symposium on Document Engineering, Florence, Italy.
www.doceng2013.org
- Sep 15–20 XML Summer School, St Edmund Hall, Oxford University, Oxford, UK.
www.xmlsummerschool.com
- Sep 23–29 7th International ConTeXt Meeting and TeXperience 2013 (CSTUG meeting), Brejlov (Prague), Czech Republic.
meeting.contextgarden.net/2013
- Sep 26–27 The Eleventh International Conference on the Book, Universität Regensburg Universitätsbibliothek, Regensburg, Germany
booksandpublishing.com/the-conference
- Sep 30, Oct 7, 14 Introduction to Metal Typesetting and Letterpress Printing, St Bride Library, London, England. sbf.org.uk/index.php/styles/200-printwrkshop
(Listing includes other workshops)
- Oct 9–13 Association Typographique Internationale (ATypI) annual conference, Theme: “Point Counter Point”, Amsterdam, The Netherlands.
www.atypi.org
- Oct 18–19 American Printing History Association’s 38th annual conference, “Seeing Color / Printing Color”, Grolier Club, New York.
www.printinghistory.org/programs/conference/conference_2013.php

Oct 22

Beatrice Warde Memorial Lecture,
“Frederic Warde: the Gatsby of Type”,
by Simon Loxley, St Bride Library,
London, England. stbride.org/events

TUG 2013 Tokyo, Japan.

- Oct 23–26 The 34th annual meeting of the TeX Users Group.
Presentations covering the TeX world.
tug.org/tug2013

- Nov 1–6 ASIS&T 2012, 75th Annual Meeting, “Beyond the Cloud: Rethinking Information Boundaries”, American Society for Information Science and Technology, Montréal, Canada.
www.asis.org/asist2013/M2013CFP.pdf

- Nov 2 DANTE Herbstagung and 49th meeting, Köln, Germany
www.dante.de/events.html

- Nov 4 *TUGboat 34:3*, submission deadline (proceedings issue)

2014

- Feb 29, Mar 1–2 Typography Day 2014, Symbiosis Institute of Design Pune, India.
www.typoday.in
- Apr DANTE Frühjahrstagung and 50th meeting, Universität Heidelberg, Germany.
www.dante.de/events/DANTE2014.html
- Apr 10–14 TYPO San Francisco, Yerba Buena Center for the Arts, San Francisco, California. typotalks.com/sanfrancisco

Status as of 10 July 2013

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568. e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

A combined calendar for all user groups is online at texcalendar.dante.de.

Other calendars of typographic interest are linked from tug.org/calendar.html.



The 34th Annual Meeting of the T_EX Users Group
October 23–26, 2013

Graduate School of Mathematical Sciences, the University of Tokyo
3-8-1 Komaba, Meguro-ku
Tokyo, Japan

<http://tug.org/tug2013> ■ tug2013@tug.org

Sept. 9, 2013 — preprint submission deadline
Oct. 23–26, 2013 — conference
Nov. 4, 2013 — deadline for final papers

Sponsored by:

Graduate School of Mathematical Sciences, the University of Tokyo
SANBI Printing
T_EX Users Group ■ DANTE e.V.

Introductory

- 111 *Barbara Beeton* / Editorial comments
 - typography and *TUGboat* news
- 115 *Hans Hagen* / Does *TeX* have a future?
 - musings on *TeX*, interactivity, ebooks, page layout, and more
- 111 *Steve Peter* / Ab epistulis
 - conferences; software; reviews
- 120 *TeX Collection editors* / *TeX* Collection 2013 DVD
 - very high-level overview of changes in the 2013 software releases
- 114 *Adeline Wilcox* / Running *TeX* under Windows PowerShell
 - tips on Unix-ish usage with this Windows shell

Intermediate

- 182 *Karl Berry* / The treasure chest
 - new CTAN packages, March–July 2013
- 168 *Charles Bigelow* / Oh, oh, zero!
 - survey of confusable characters (Oo1l, etc.) in historical and modern typography
- 123 *Taco Hoekwater* / MetaPost: PNG output
 - creating bitmap images directly from MetaPost
- 121 *Khaled Hosny* / What is new in *XeTeX* 0.9999?
 - moving to HarfBuzz, Core Text on Macs, and more
- 124 *Dirk Hünniger* / Converting Wikipedia articles to *LATEX*
 - the *wb2pdf* Haskell project for local conversion of wiki to *LATEX*
- 125 *Michael Sharpe* / A survey of text font families
 - comparison of many font families usable with (I)*A**TEX* for general text
- 196 *Mari Voipio* / Entry-level MetaPost 2: Move it
 - shifting, rotating, reflecting, repeating
- 136 *Yue Wang* / Interview with Charles Bigelow
 - in-depth discussion of Lucida and typography

Intermediate Plus

- 181 *Karl Berry* / Production notes
 - an application of FontForge, *ttf2afm*, and *otftotfm*
- 200 *Juernjakob Dugge* / Creating Tufte-style bar charts and scatterplots using PGFPlots
 - including range frames and dot-dash plots in Tufte's style
- 190 *Hans-Georg Eßer* / LiPPGen: A presentation generator for literate-programming-based teaching
 - Python package to create slides from literate programs
- 184 *Paweł Łukowski* and *Mariusz Urbański* / Preparing for scientific conferences with *LATEX*:
 - A short practical how-to
 - methods for creation of a related paper, presentation, and poster
- 132 *Peter Wilson* / Glisterings
 - a font of fleurons; fonts, GNU/Linux, and *XeTeX*; mixing traditional and system fonts

Advanced

- 205 *Bogusław Jackowski* / Typographers, programmers and mathematicians,
 - or the case of an aesthetically pleasing interpolation
 - in-depth mathematical description of Hobby's algorithm for connecting Bézier segments
- 223 *Kevin Donnelly* / Representing linguistic pitch in (*Xe*)*LATEX*
 - typesetting linguistic pitch representations, directly or with *TikZ*, et al.

Contents of other *TeX* journals

- 235 *Die Texnische Komödie* 2/2013; *ArsTeXnica* 15 (2013); *EuroBachotEX* 2013

Reports and notices

- 110 Institutional members
- 112 *Doug Henderson* / In memoriam: Barry Smith (1953–2012)
- 113 *Barbara Beeton* / Hyphenation exception log
 - update for missed and incorrect U.S. English hyphenations
- 228 *Boris Veytsman* / Book review: *Learning LATEX*, David Griffiths and Desmond Higham
- 229 *Boris Veytsman* / Book review: *Zapf Exhibition: The Calligraphy of Hermann & Gudrun Zapf*
 - description of the catalogue of this 2011 exhibition in Japan
- 230 *Will Adams* / Book review: *What Our Lettering Needs*, Rick Cusick
 - The contribution of Hermann Zapf to calligraphy & type design at Hallmark cards
- 231 *Boris Veytsman* / Book review: *LATEX Quick Reference*, Herbert Voß
- 232 *Boris Veytsman* / Book review: *Presentations with LATEX*, Herbert Voß
- 233 *TeX* consulting and production services
- 240 *Barbara Beeton* / TUG 2013 election
- 243 Calendar
- 244 TUG 2013 announcement