TUGboat

Volume 26, Number 2 / 2005
TUG 2005 Conference Proceedings

TEX Users Group

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

Memberships and Subscriptions
2005 dues for individual members are as follows:

- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership 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: `http://www.tug.org`.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $85 a year, including air mail delivery.

Institutional Membership
Institutional Membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group. For further information, contact the TUG office (`office@tug.org`) or see our web site.

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

Delivery services, parcels, visitors
TEX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone
+1 503 223-9994

Fax
+1 503 223-3960

Electronic Mail
(Internet)

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

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

Technical support for TEX users:
`support@tug.org`

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

World Wide Web
`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: November 2005]

TUGBOAT

The Communications of the TEX Users Group

Volume 26, Number 2, 2005
TUG 2005 Conference Proceedings

TeX Users Group

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

Memberships and Subscriptions

2005 dues for individual members are as follows:

- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership 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: `http://www.tug.org`.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $85 a year, including air mail delivery.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group. For further information, contact the TUG office (`office@tug.org`) or see our web site.

Addresses

General correspondence, payments, etc.
TeX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TeX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 503 223-3960

Electronic Mail

(Internet)

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

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

Technical support for TeX users:
`support@tug.org`

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

World Wide Web

`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

Problems not resolved?

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: November 2005]

TUG 2005 Proceedings

Wuhan, China

August 23–25, 2005

TUGboat

COMMUNICATIONS OF THE TEX USERS GROUP

TUGboat Editor Barbara Beeton

Proceedings Editor Karl Berry

Volume 26, Number 2 • 2005

Portland • Oregon • U.S.A.

TUG 2005
International Typesetting Conference

Sponsors

Chinese TEX User Group ▪ TEX Users Group ▪ DANTE e.V.

Acknowledgements

With great appreciation to all the speakers and teachers, and special thanks to:

- Wai Wong, for agreeing to be the keynote speaker.
- Duane Bibby, for the traditionally excellent drawings.

Conference committee

Karl Berry ▪ Chen HaiBin ▪ Gu XiongZhi ▪ Hong Chan ▪ Hong Feng ▪
Hu WeiMing ▪ Robin Laakso ▪ Liu Hao ▪ Peng ZiQiang ▪ Wan Lin ▪
Wu ZhenHua ▪ Xu JiZhe ▪ Yao WeiZhen ▪ Zhang YuanLin

International participants

Le Khanh Au Duong, HM Revenue & Customs
Nelson Beebe, University of Utah
Jin-Hwan Cho, University of Suwon
Steve Grathwohl, Duke University Press & TUG
Guo Liang, Institute of Software, Chinese Academy of Science, Beijing
Eitan Gurari, Ohio State University
Hans Hagen, Pragma ADE & NTG
Hartmut Henkel, von Hoerner & Sulger GmbH
Klaus Höppner, DANTE e.V. & TUG
Maik Kündig, Fallenden, Switzerland
Jonathan Kew, SIL International
Richard Kinch, TrueTEX, Inc.
Robin Laakso, TEX Users Group
Jerzy Ludwichowski, Nicolaus Copernicus University & GUST
Ross Moore, Macquarie University & TUG
Karel Píška, Czech Academy of Sciences
Arthur Reutenauer, France
Chris Rowley, Open University & LATEX3 Project
Volker R.W. Schaa, DANTE e.V.
Karel Skoupý, Zurich, Switzerland
Philip Taylor, University of London & TUG
S. K. Venkatesan, TnQ Books and Journals
Kevin Warnock, Silveroffice, Inc.
Wai Wong, Chinese University of Hong Kong
Candy Yiu, Portland State University
Sam Zoghaib, France

Local participants

Chen Haibin
Hong Feng
Liu Hao
Lu Ming
Meng Yan
Peng ZiQiang
Wang JiaHui
Wu Xuemou
Wu Zhenhua
Yao Weizhen
Zhu Xiang

TUG 2005 — program and information

Monday August 22 (tutorials)	8–9 am	*registration*	
	9 am	Hong Feng	*TeX as a compiler*
	10:30 am	*break*	
	10:45 am	Ross Moore	*LaTeX to HTML conversion*
	12 pm	*lunch*	
	1:30 pm	Chris Rowley	*LaTeX for beginners*
	2:45 pm	Hartmut Henkel	*MetaPost for beginners*
	4 pm	*break*	
	4:15 pm	Hans Hagen	*ConTeXt for beginners*
	5:30–7 pm	*reception*	
Tuesday August 23	9 am	Hong Feng, CTUG	*Welcome*
	9:15 am	Wai Wong, Chinese University of Hong Kong, China	keynote address: *Typesetting Chinese—A personal perspective*
	10:15 am	*break*	
	10:30 am	Jonathan Kew, SIL International	*XeTeX, the multilingual lion: TeX meets Unicode and smart fonts*
	11:15 am	Philip Taylor, University of London	*Typesetting the Byzantine Cappelli*
	11:45 am	Candy Yiu, Portland State University	*Qin's music notation generator*
	12:15 pm	*lunch*	
	1:15 pm	Nelson Beebe, University of Utah	*The design of TeX and METAFONT: A retrospective*
	2:15 pm	Karel Skoupý, ETH Zentrum	*Free-shape text formatting*
	3 pm	*break*	
	3:15 pm	Suki Venkatesan, TnQ Books and Journals	*Moving from bytes to words to semantics*
	4 pm	Chris Rowley, Open University	*Beyond TeX: An introduction to new models for high quality document fomatting*
	4:45 pm	panel: CJKV and TeX	moderator: *Hong Feng; Jin-Hwan Cho, Hans Hagen, Jonathan Kew, Chris Rowley, Wai Wong*
Wednesday August 24	9 am	Hong Feng	*Wavelet transformations and Chinese font design*
	9:45 am	Karel Píška, Czech Academy of Sciences	*Converting METAFONT sources to outline fonts using MetaPost*
	10:30 am	*break*	
	10:45 am	Jin-Hwan Cho, University of Suwon	*Practical use of special commands in DVIPDFMx*
	11:30 am	Eitan Gurari, Ohio State University	*Spatial math exercises and worksheets*
	12:15 pm	*lunch*	
	1:15 pm	Klaus Höppner, DANTE e.V.	*Strategies for including graphics in LaTeX documents*
	2 pm	Philip Taylor	*Grid typesetting in LaTeX*
	2:45 pm	*break*	
	3 pm	Chris Rowley	*LaTeX maintenance and development*
	3:45 pm	Ross Moore, Macquarie University	`PlanetMath.org` *and the Free Encyclopaedia of Mathematics*
	4:30 pm	Jerzy Ludwichowski, Nicolaus Copernicus University	*World wide TeX user groups review*
	5 pm	q & a	
	5:30 pm	*TUG annual meeting*	
Thursday August 25	9 am	Steve Grathwohl, Duke Univ. Press	*On ConTeXt*
	9:45 am	Hans Hagen, Pragma ADE & NTG	*LuaTeX: Howling to the moon*
	10:45 am	*break*	
	11 am	Volker R.W. Schaa, DANTE e.V.	*XML workflows and the EuroTeX 2005 proceedings*
	11:45 am	panel: Digital publishing	moderator: *Hong Feng; Nelson Beebe, Steve Grathwohl, Ross Moore, Volker R.W. Schaa, Philip Taylor*

Highlights from TUG 2005

Robin Laakso
TEX Users Group
office@tug.org
http://tug.org/tug2005

TUG 2005: International Typesetting Conference and the 26th annual meeting of the TEX Users Group, was held in Wuhan, China, from August 23–25, 2005. CTUG (the Chinese TEX User Group), committed to handling the conference affairs. Congratulations to Hong Feng, chairman of CTUG, and all of his local help, including Chen HaiBin, Hong Chan, Gu XiongZhi, Hu WeiMing, Liu Hao, Peng ZiQiang, Wan Lin, Wu ZhenHua, Xu JiZhe, Yao WeiZhen, and Zhang YuanLin, for making the conference a unique experience and resounding success.

TEX users traveled from near and far to attend this year's conference and tutorials. China, Australia, South Korea, Europe, India and North America were all represented. Hong and/or Chen HaiBin, Wu ZhenHua, Peng ZiQiang met conference attendees at the airport outside of Wuhan arranging bus and taxi transportation to our final destination, the East Lake Hotel in Wuchang, Wuhan. We were invited to enjoy a welcome dinner near the hotel upon our arrival — a restaurant setting that would become familiar over the next several days as the conference room was just down the hall, and where breakfast, lunch and dinner were served, included in the registration.

A variety of brief tutorials was offered on Monday, August 22, the day before the conference opened. The instructors were a top notch fit with regard to their respective topics. The line-up included: Hong Feng, *TEX as a Compiler*; Ross Moore, *LATEX to HTML Conversion*; Hartmut Henkel, *MetaPost for Beginners*; Hans Hagen, *ConTEXt for Beginners*, and Chris Rowley, who ended up choosing to forgo his *LATEX for Beginners* course as we were running behind schedule (and because there were scant few beginners in the room).

Wai Wong, with the Chinese University of Hong Kong, delivered a charming and informative keynote address entitled "Typesetting Chinese: A personal perspective". Wai's presentation was followed by about 20 talks over the three days, ranging from a look back at the design of TEX and METAFONT by Nelson Beebe, to a look ahead at improving pdfTEX via the LuaTEX team as presented by Hans Hagen. A wide range of topics was covered in-between, such as accessing fonts in the operating system, among other features of XƎTEX, presented by its author Jonathan Kew, and using TEX in conjunction with other tools such as Perl and Excel, to sort, analyze, and finally typeset the Byzantine *Cappelli*, presented by Phil Taylor. Each and every one of the presentations was well attended and enthusiastically received.

Wednesday afternoon, Jerzy Ludwichowski of GUST and the Nicolaus Copernicus University in Poland summarized a bit about TEX user group activities worldwide. He passed along some ideas generated at BachoTEX such as producing a font CD, perhaps including Chinese fonts, and setting up a Wiki about the history of TEX. Jerzy's talk preceded the TUG annual meeting where, after the group was updated about TUG business, upcoming conferences, and a proposed TEX Hall of Fame, discussion focused on ways to increase both interest in TEX and membership in TUG. Suggestions ranged from advice about how to advertise TEX and TUG nationally and internationally using a PR firm on the web, to introducing a simple LATEX system to grade schoolers.

The conference officially concluded Thursday with a round-table panel. In the afternoon, many attendees boarded a bus to tour a nearby archeological museum and one of Chairman Mao's former residences. Late afternoon that same day about half the group journeyed overnight by bus from Wuhan to the town of Wudang. After a rest and a traditional Chinese breakfast, the group boarded the bus again, arriving at Wudang Mountain, the birthplace of Taoism, about an hour later. Onward and upward, though this time boarding gondolas two by two, we continued our long vertical climb via pathways and stairs that eventually led to Taoist temples, nunneries, temples on cliffs, bridges, pavilions and ancestral temples. Needless to say, the views were spectacular from most locations on Wudang Mountain. And even though the journey from Wuhan to Wudang was a long one (thus some people departed Wudang Friday afternoon by train), the visit to this magnificent and special place was appreciated by all.

Some comments from attendees:

—For the most part, I think it went pretty well. It would have been helpful in planning travel to know in advance what the Wudang Mountain expedition would involve; at the time I was booking my flights, there was no hint that this was more than a one-day excursion. (But despite the difficulties, I very much enjoyed going.)

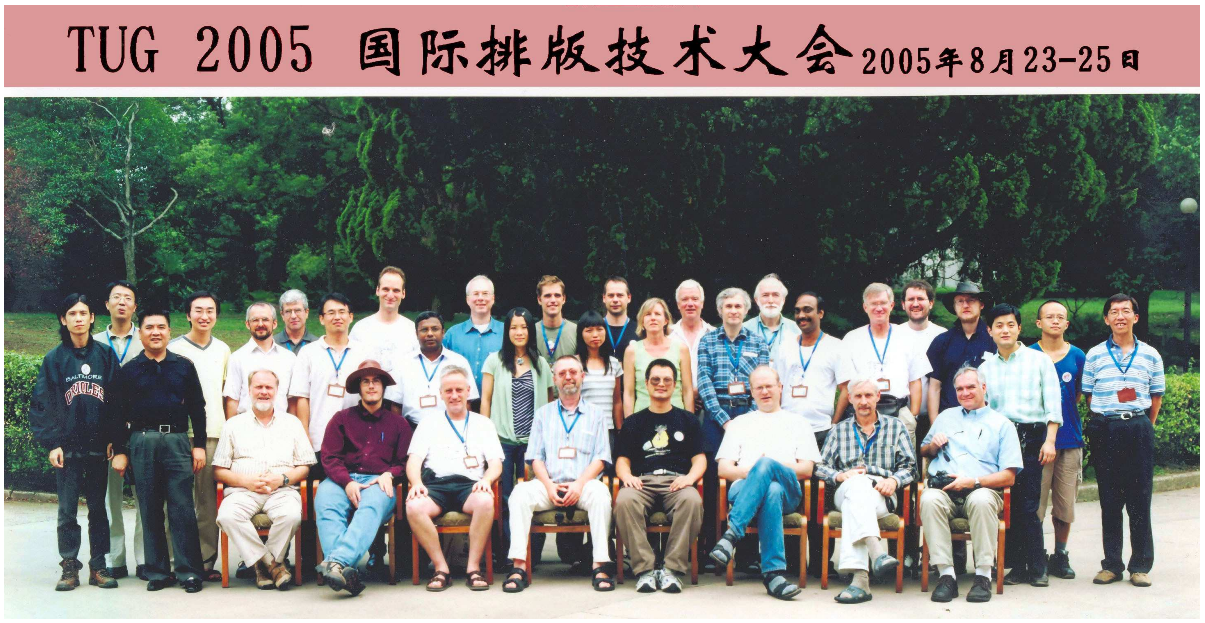
—I think the local Wuhan tours were outstanding, and I loved them.

—I really enjoyed this conference. The bus ride to Wudang added to the richness of the trip, and I'm happy to have had the experience. We saw a side of China we wouldn't have seen otherwise.

—Once again, thank you very much for inviting me to TUG 2005. I did enjoy the talks and the discussion during the conference very much. I learnt a lot through them.

—I think the informal discussions we had during the panels were especially valuable. Some very good ideas were exposed there, and would be less likely to come up in more formal settings such as regular lectures, were we often lack time to answer questions comprehensively. Thus, I believe it would be quite profitable to everyone to have more of these. However, I must admit I don't know how these scale up when there are more participants; we really weren't that many, probably because some people couldn't make it to China.

—Hope to see you next year!



Sitting in chairs left to right: Chris Rowley, Sam Zoghaib, Ross Moore, Volker Schaa, Hong Feng, Hans Hagen, Jerzy Ludwichowski, Richard Kinch

Back, left to right: Yao Weizhen, Guo Liang, anonymous photographer, Liu Hao, Jonathan Kew, Eitan Gurari, Jin-Hwan Cho, Kevin Warnock, Joseph Rajendra, Hartmut Henkel, Candy Yiu, Maik Kündig, Le Khanh Au Duong, Karel Skoupý, Robin Laakso, Phil Taylor, Karel Píška, Nelson Beebe, S.K. Venkatesan, Steve Grathwohl, Klaus Höppner, Arthur Reutenauer, Lu Ming, Wu Zhenhua, Wai Wong.

Thanks to Hong Feng, William Adams, and Mare Joy Smith for work on this photograph for publication.

Special thanks to the photographers: Hans Hagen, Hartmut Henkel, Jerzy Ludwichowski, and Volker R.W. Schaa.

Under the welcome banner at Eastlake Hotel: Hartmut Henkel, Jerzy Ludwichowski, Hans Hagen.

Panel on CJKV and TEX: Hong Feng, Hans Hagen, Jin-Hwan Cho, Wai Wong.

Robin Laakso, Steve Grathwohl (at table); Candy Yiu, Nelson Beebe, Chris Rowley, Klaus Höppner (behind).

TEX is everywhere.

Jerzy making creative use of chopsticks at Hong's birthday evening, with Karel Píška observing.

Panel on digital publishing: Hong Feng, Nelson Beebe.

Typesetting Chinese: A personal perspective

Wai Wong
The Chinese University of Hong Kong
School of Continuing Studies
Shatin, Hong Kong, China
waiwongww@gmail.com

Introduction

First of all, I would like to thank TUG for inviting me to speak at the TUG 2005 conference. This is the first time ever that a TUG conference has been held in China — where typesetting was first invented. As a Chinese citizen, I welcome all of you who traveled such a long way to participate in the conference and to visit my country.

My talk will begin with a very brief survey of the development of printing in China since the invention of movable types in the 11th century. A proper account of this development can easily fill several volumes; therefore, given our limited time, I will show only a few interesting examples, which are selected quite arbitrarily. Hopefully, they will still provide us some insights into certain conventions in typesetting Chinese books that still influence current practice.

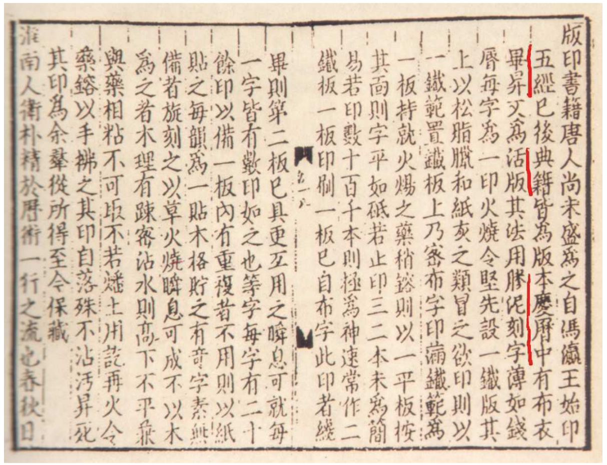
Then, I will compare briefly the page layout and available fonts between European typesetting and Chinese typesetting in the 20th century.

Lastly, from a personal perspective, I will talk about the challenge that high-quality professional typesetting systems such as TEX and its friends face when typesetting Chinese.

A brief history of movable type printing in China

The invention of movable types in China was in the period of 1041 to 1048 by a common man named *Bi Sheng*. He himself did not leave any documented evidence of his invention. The earliest record of movable type was written by *Shen Kuo* in his book *Dream Pool Jotting*. Figure 1 shows the page that describes how to make clay movable types.

Since then, a variety of different materials have been used to make movable types, including copper, tin, lead, wood, clay and porcelain. After *Bi Sheng*'s work, the next most significant development of movable types was by *Wang Zhen*. He surveyed the various techniques of making movable types using clay and tin types in his book *A Treatise of Agriculture*. In the same book, he described a new technique for



版印書籍唐人尚未盛爲之自馮瀛王始印
五經已後典籍皆爲版本慶曆中有布衣
畢昇又爲活版其法用膠泥刻字薄如錢
脣每字爲一印火燒令堅先設一鐵版其
上以松脂臘和紙灰之類冒之欲印則以
一鐵範置鐵板上乃密布字印滿鐵範爲
一板持就火煬之藥稍鎔則以一平板按
其面則字平如砥若止印三二本未爲簡
易若印數十百千本則極爲神速常作二
鐵板一板印刷一板已自布字此印者纔
畢則第二板已具更互用之瞬息可就每
一字皆有數印如之也等字每字有二十
餘印以備一板內有重複者不用則以紙
貼之每韻爲一貼木格貯之有奇字素無
備者旋刻之以草火燒瞬息可成不以木
爲之者木理有疏密沾水則高下不平兼
與藥相粘不可取不若燔土用訖再火令
藥鎔以手拂之其印自落殊不沾汚昇死
其印爲余羣從所得至今保藏
淮南人衛朴精於曆術一行之流也春秋日

Figure 1: A page of *Dream Pool Jotting* by Shen Kuo, the earliest record of making movable types.

typesetting Chinese with a detailed drawing. This employed a revolving table for arranging the types by a rhyming scheme, which was believed to be invented by the book's author. A model of this revolving table is shown in Figure 2.

While movable type printing was used and developed continually throughout the centuries since its invention, it has never dominated the printing industry in China, at least not until the beginning of the twentieth century. The technique of wood block printing has been perhaps more popular than movable type printing throughout history. This is because:

- The intrinsic complexity of the Chinese language. There are thousands to tens of thousands of different characters.
- The difficulty and expense of producing such a large number of types.
- The relatively low demand for books, making movable type printing less advantageous than wood block printing.

Table 1 shows a timeline summarizing the historical development of printing in China.

Table 1: Historical Development of Chinese printing: a brief timeline

1600–1066 BC		writing brush and ink
403–221 BC		earliest existing brush
206BC–220 AD		ink made from pine soot
105 AD	Cai Lun	improved papermaking technique
636 AD		woodblock printing
1041–1048	Bi Sheng	invented movable-type printing with clay type (recorded by Shen Kuo)
1297	Wang Zhen	improved movable-type printing with wooden and metal (tin alloy) type
1455	Gutenberg	metal movable-type printing
1660s	Mueller	wooden Chinese type (3824 pieces)
1812–1822	Morrison	tin alloy type, 200,000 pieces of 20,000 different characters; printed a six-volume dictionary in Macao
1859–1861	Cole	a complete set of types in seven different sizes, Shanghai
1915		old-style type commissioned by the Commercial Press (two sizes)
1927–1934		Fong Song style type made

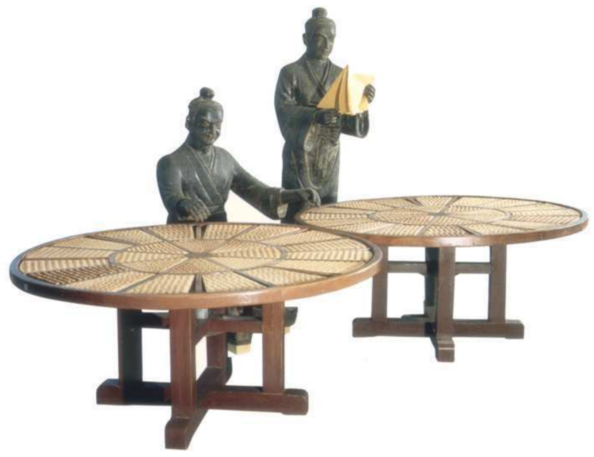


Figure 2: A model of Wang Zhen's revolving table for typesetting

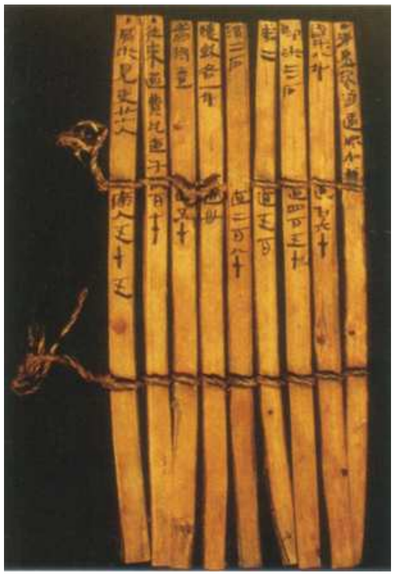


Figure 3: A rolled-up strip of writing on bamboo

Bookbinding

The oldest books in Chinese were written on strips made of wood or bamboo. They were then tied and rolled up using strings, as illustrated in Figure 3. Because of these long thin strips, when paper was invented, followed by the subsequent invention of printing, the Chinese still kept their writing direction vertical.

Bookbinding developed from scroll rollers, to pleated leaf binding, to wrapped-back binding, to thread binding. The vertical direction was preserved through a thousand years of history. Only in the mid-twentieth century did the horizontal direction begin to dominate in most of China. Even today, most books of literature published in Taiwan and Hong Kong are in the vertical direction.

Influence from the west

Starting in the 19th century, trading and exchange in many other fields between China and the west increased dramatically. Probably the most important influence on modern printing in China came from a Christian missionary named Robert Morrison. He arrived in Macao in 1807. Later he set up a printing press and published a 6-volume dictionary of the Chinese language in 1815. For this project, he made 200,000 pieces of type of 20,000 different characters.

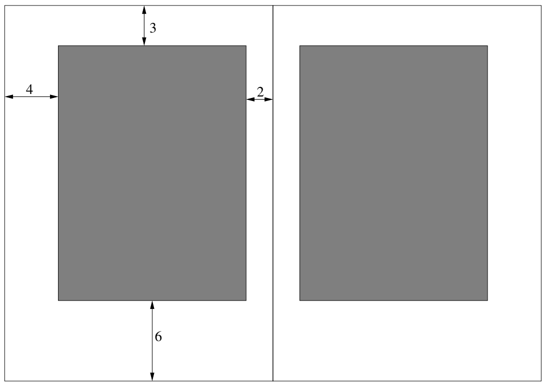
Unfortunately, these type pieces were destroyed in 1856 in a riot.

The first half of the 19th century saw many westerners coming to China. Another influential person in Chinese printing history from that time is an American named Cole, who initially set up a printing press in Macao. The press was moved to Ningpo and later to Shanghai in 1860. He set up a type foundry in Ningpo, and produced type pieces in seven different sizes.

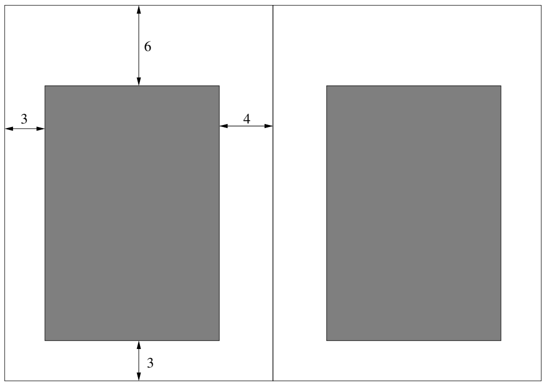
Page layout

It is interesting to compare the typical page layout of western and Chinese books. Although there are numerous variations, traditional European book designers lay out a typical page with the top margin smaller than the bottom margin and the inner margin smaller than the outer margin, as shown in Figure 4a.

In contrast, Chinese tradition puts the emphasis on the opposite side. The top margin is larger than the bottom margin and the inner margin is larger than the outer margin, as shown in Figure 4b. This ratio has been developed from far back in history, when text was printed vertically on one side only, and the pages were folded back and bound using stitches.



a. Traditional European books



b. Traditional Chinese books

Figure 4: Comparison of page layout

Measurement of types

Throughout their long history of printing with movable type, the Chinese did not develop a common system for type measurement. Each type maker made type pieces in their own sizes. It was not until the mid-nineteenth century, when modern movable types and printing presses were introduced from the west, that a common system of type sizes was developed. This system initially had 7 different sizes, numbered sequentially.

Table 2 lists the type size numbers and their equivalent point size. This numbering system was widely adopted and used until the very recent times when digital typesetting replaced traditional metal types.

Table 2: Numbering system for Chinese type sizes

#	#1	#2	#3	#4	#5	#6	#7
pt	28	21	16	14	10.5	8	5.25

Another widely used system of type measurement is the K system. It was introduced together with phototypesetters from Japan. 1K is equal to 0.25 mm, and commonly used sizes are between 8K and 40K. Since the widespread adoption of digital typesetting, the point system has become dominant.

Available fonts

Creating fonts is a complicated and expensive undertaking. Song and Kai are the names of the two main families, with their roots in wood block printing. They were the only font styles for a very long time in the history of movable type printing. Other font families have been developed only recently: Fong Song style was created in 1916. With the introduction of phototypesetters from Japan, available Chinese font families started to increase. However, there are still many fewer Chinese fonts than western fonts.

It is not uncommon to commission typefaces for a book in western languages. In contrast, it is extremely rare for a Chinese printing press or publisher to commission a font for a book. Although with the advance of digital typesetting more fonts are now available, many new fonts lack many characters, often even common characters, let alone the complete repertoire of CJK characters in Unicode.

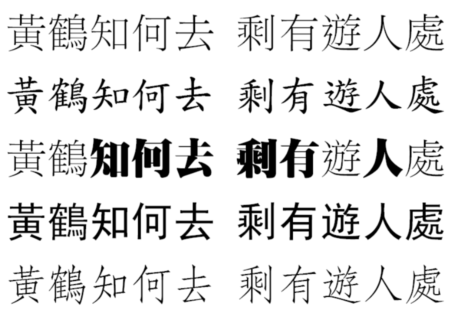


Figure 5: Some common font styles: Song, Kai, Old Song, Hei and Fong Song (from top to bottom)

For example, the Old Song style shown in the middle row of Figure 5 lacks four of the ten sample characters. They therefore have to be substituted by other styles, which is hardly acceptable in professional publishing.

Opportunities and challenges

The current situation in Chinese typesetting is that the era of hot metal has passed away. Digital technology is used but it is dominated by a very small number of imported commercial software applications. Home-grown software has quite a small portion of the market. The demand for high quality typesetting software is high.

On the other hand, open source and free software like TEX and its friends is available, but localization effort is required. As has been described above, localization does not only mean being able to handle Chinese characters. It is necessary to cater to the cultural differences as well. What Chinese users need is integrated solutions. TEX and its friends provides a flexible foundation. Good solutions can be built on top of this foundation. The challenge is to bridge the gap between these existing technologies and the specific requirements of Chinese typesetting.

XƎTEX, the Multilingual Lion: TEX meets Unicode and smart font technologies

Jonathan Kew
SIL International
Horsleys Green
High Wycombe HP14 3XL
England
jonathan_kew@sil.org

Abstract

Professor Donald Knuth's TEX is a typesetting system with a wide user community, and a range of supporting packages and enhancements available for many types of publishing work. However, it dates back to the 1980s and is tightly wedded to 8-bit character data and custom-encoded fonts, making it difficult to configure TEX for many complex-script languages.

This paper will introduce XƎTEX, a system that extends TEX with direct support for modern OpenType and AAT (Apple Advanced Typography) fonts and the Unicode character set. This makes it possible to typeset almost any script and language with the same power and flexibility as TEX has traditionally offered in the 8-bit, simple-script world of European languages. XƎTEX (currently available on Mac OS X, but possibly on other platforms in the future) integrates the TEX formatting engine with technologies from both the host operating system (Apple Type Services, CoreGraphics, QuickTime) and auxiliary libraries (ICU, TECkit), to provide a simple yet powerful system for multilingual and multiscript typesetting.

The most significant extensions XƎTEX provides are its native support for the Unicode character set, replacing the myriad of 8-bit encodings traditionally used in TEX with a single standard for both input text encoding and font access; and an extended `\font` command that provides direct access by name to all the fonts installed in the user's computer. It also provides a mechanism to access many of the advanced layout features of modern fonts.

Additional features that will also be discussed include built-in support for a wide variety of graphic file formats, and an extended line-breaking mechanism that supports Asian languages such as Chinese or Thai that are written without word spaces.

Finally, we look briefly at some user-contributed packages that help integrate the features of XƎTEX with the established LATEX system. Will Robertson's `fontspec.sty` provides a simple, consistent user interface in LATEX for loading both AAT and OpenType fonts, and accessing virtually all of the advanced features these fonts offer; Ross Moore's `xunicode.sty` is a package that allows legacy LATEX documents to be typeset using native Mac OS X fonts without converting the input text entirely to Unicode, by supporting traditional TEX input conventions for accents and other "special" (non-ASCII) characters.

— — ∗ — —

Editor's note: This article is typeset in Adobe Garamond, with Andale Mono for the code examples, and processed on the author's Mac OS X machine with XƎTEX, as Unicode support was needed in several places.

What is XƎTEX?

XƎTEX[1] is an extension of the TEX processor, designed to integrate TEX's "typesetting language" and document formatting capabilities with the Unicode/ISO 10646 universal character encoding for all the world's scripts, and with the font technologies available on today's computer systems, including fonts that support complex non-Latin writing systems.

XƎTEX is in fact based on ε-TEX, and therefore includes a number of well-established extensions to TEX. These include additional registers (`\count`, `\dimen`, `\box`, etc.) beyond the 256 of each that TEX provides; various new conditional commands, tracing features, etc.; and of particular significance for multilingual work, the TEX–XƎT extension for bidirectional layout.

The TEX extensions inherited from ε-TEX are not discussed further here, as they are already described in the ε-TEX documentation[2], except to note that for right-to-left scripts in XƎTEX, it is necessary to set `\TeXXeTstate=1` and make proper use of the direction-changing commands `\beginR`, `\endR`, etc. Without these, there will still be some right-to-left behavior due to the inherent directionality defined by the Unicode standard for characters belonging to Hebrew, Arabic and similar scripts, but overall layout will not be correct.

XƎTEX was created in order to typeset materials — literacy and educational books, translated Scriptures, linguistic studies, dictionaries, etc. — in a wide range of languages and scripts, including lesser-known ones that are not adequately supported in most existing products. It inherits ideas, and even some code, from an earlier system called TEXGX that integrated TEX with the QuickDraw GX graphics system on older Macintosh operating systems.

[1] The name XƎTEX was inspired by the idea of a Mac OS X extension (hence the 'X' prefix) to ε-TEX; and as one of its intended uses is for bidirectional scripts such as Hebrew and Arabic, the name was designed to be reversible. The second letter should ideally be U+018E LATIN CAPITAL LETTER REVERSED E, but as few current fonts support this character, it is normal to use a reflected 'E' glyph. The name is pronounced as if it were written *zee-TEX*.

[2] E.g., *The ε-TEX Short Reference Manual*, `http://www.staff.uni-mainz.de/knappen/etex_ref.html`.

```
\font\Hoefler = "Hoefler Text" at 10pt
\Hoefler This is the Hoefler Text font.
```

This is the Hoefler Text font.

```
\font\VerdanaItal = "Verdana Italic" at 9pt
\VerdanaItal And this is Verdana Italic.
```

And this is Verdana Italic.

Figure 1: Accessing fonts installed natively on the host platform

Host platform fonts

For many users, one of the most significant features of XeTeX is that it makes use of the fonts installed in the operating system, just like mainstream GUI word processing and page layout programs. On Mac OS X, fonts in a number of major formats — in particular, TrueType (`.ttf`) fonts, and both TrueType- and CFF-flavored[3] OpenType (`.otf`) fonts, as well as legacy Macintosh resource file formats — can be installed in any of several **Library/Fonts** folders (system-wide, or per-user), and users expect these fonts to be available in all applications.

With a traditional TeX system, this is not the case. Because of its portable, platform-independent heritage, TeX knows nothing about the fonts installed in a particular operating system, or even about today's major font formats; it relies instead on `.tfm` files (an alien concept to the typical modern font user) to provide the metrics information needed for typesetting, and on output drivers that locate and use the actual font files containing glyph images. All these are specifically installed for TeX and associated tools, quite separately from font installation for the operating system or other applications. Many users find this a challenge, and do not feel confident to use fonts other than those provided with their TeX distribution. So there is a perception that TeX supports a very limited range of fonts. XeTeX aims to change this.

Font access in XeTeX Within a XeTeX document, it is trivial for users to typeset using whatever fonts they have on their computer system. If a Mac OS X user buys or downloads a `.ttf` or `.otf` font and installs it in the standard way with FontBook or by placing the file in **~/Library/Fonts**, the font can be used by just specifying it by name with a `\font` command, as in figure 1. No conversions, no auxiliary files, no TeX-specific installation or configuration; just tell XeTeX to use the font, and there it is. (Note that figure 1, like most examples in this paper, uses simple "plain TeX"-level commands; in the context of packages such as LaTeX or ConTeXt there would be higher-level commands designed to interact properly with the overall package.)

[3] CFF: Compact Font Format, the table type that holds PostScript glyph data in an OpenType font container.

When XeTeX is using "native" fonts from the operating system, it handles text in a slightly different way than standard TeX. Rather than treating each character individually, looking up its metrics (in a `.tfm` file), it collects "runs" (typically, but not always, complete words) and passes them to the font rendering subsystem as complete chunks of text. This is necessary in order to allow the font to implement features such as ligatures, cursive connections, contextual character substitutions or reordering, etc., which may be defined in AAT or OpenType fonts (see below). Such features may represent optional typographic refinements in Latin-based scripts, but in many Asian scripts they are essential for correct rendering.

Output device support Selecting fonts by name within the source document, and having the typesetting process find and use them when building paragraphs, is only half the story. Drivers that render TeX output onto a particular device also need to locate fonts — and in the traditional TeX world, the two stages rely on separate files, with typesetting requiring only `.tfm` files, and output requiring "real" fonts of some kind, e.g., `.pk` or `.pfb` files.

The current implementation of XeTeX creates output in an "extended DVI" format (`.xdv`), and this is then converted to PDF by a second process, `xdv2pdf`.[4] To generate PDF, `xdv2pdf` relies on the user's installed fonts in exactly the same way as the typesetting process. There is no separation between fonts as used during typesetting and those used for output.

Because the output format is effectively PDF (as the `.xdv` $\longrightarrow$ `.pdf` conversion is automatically executed), XeTeX output can then be viewed or printed on any system or device where PDF is supported, using standard viewers and printer drivers.

Support for legacy TeX fonts In addition to using fonts installed natively in the operating system, XeTeX continues to support the use of existing fonts in the `texmf` tree, using `.tfm` files (for metrics) and `.pfb` fonts (Type 1 outlines, for rendering). When using such `.tfm`-based fonts, the results should be identical to those produced by standard TeX.

Note that the current `xdv2pdf` driver supports such legacy fonts only in `.pfb` format; there is no support, in particular, for `.pk` or other METAFONT-derived bitmap formats. There is also no `.vf` support at present.

The use of `.tfm`-based fonts is important partly for compatibility with existing documents that use these fonts, where a user might wish to take advantage of some XeTeX features without changing the overall look of the document. Perhaps more important, `.tfm`-based fonts are *required* for math mode, as TeX's math formatting makes use of detailed

[4] The default behavior is for the `xetex` process to automatically pipe its `.xdv` output through `xdv2pdf`, so that the default output format appears to be PDF.

metric information that comes from the METAFONT fonts and cannot readily be generated for the system's native fonts. This means that math typesetting continues to work unchanged in XƎTEX; however, it also means that *for math*, the range of fonts available remains very limited.

Unicode support

TEX was originally designed for English typesetting, with characters needed for other (primarily European) languages supported via the `\accent` command and additional characters (such as ß and æ) provided in the Computer Modern fonts and accessed via control sequences, to escape the limitations of the ASCII character set. Many other languages and scripts have also been handled, using a variety of techniques including custom codepages and fonts, macros and "active" characters, and even preprocessors that implement specific complex scripts such as Devanagari.

The variety of TEX programming tricks available, together with the use of non-standard input and font encodings and similar techniques, have allowed many scripts to be typeset; however, they have also meant that the input text used for typesetting is often encoded in a non-standard way, unique to the particular TEX package in use, making for problems of data interchange with other systems. And the use of preprocessors and/or TEX macros to implement script behavior can easily conflict with other levels of macro programming (document markup and formatting control), making for complex and fragile systems.

The Unicode standard offers the possibility of a much simpler, cleaner multilingual system. In Unicode, every character needed for any script has (in principle) its own code, so there is no longer any need for multiple codepages, where the meaning of a particular character code depends on the input encoding or font in use. Nor is there any need for escape sequences or preprocessors to access characters that cannot be entered directly in the input; text in any language can be represented as simple character data. So XƎTEX aims to extend TEX such that the standard character encoding used throughout the typesetting process, from text input to accessing glyphs in fonts, is Unicode.

Character codes The first step towards Unicode support in TEX is to expand the character set beyond the original 256-character limit. At the lowest level, this means changing internal data structures throughout, wherever characters were stored as 8-bit values. As Unicode scalar values may be up to U+10FFFF, an obvious modification would be to make "characters" 32 bits wide, and treat Unicode characters as the basic units of text.

However, in XƎTEX a pragmatic decision was made to work internally with UTF-16 as the encoding form, making "characters" in the engine 16 bits wide, and handling supplementary-plane characters using UTF-16 surrogate pairs. This choice was made for a number of reasons:

- The operating-system APIs that XƎTEX uses in working with Unicode text require UTF-16, so working with this encoding form avoids the need for conversion.
- A number of internal arrays in TEX are indexed by character codes. Enlarging these from 256 elements each to 65,536 elements seems reasonable; enlarging them to a million-plus elements each would dramatically increase the memory footprint of the system. To avoid this, a sparse array implementation might be used, but this would be significantly more complex to develop and test, and might well have a negative impact on typesetting performance.
- It seems unlikely, in any case, that there will be much need to customize these properties (see next section) for characters beyond Plane 0.

In view of these factors, XƎTEX works with UTF-16 code units. Unicode characters beyond U+FFFF can still be included in documents, however, and will render correctly (given appropriate fonts) as the UTF-16 surrogate pairs will be passed to the font system.

Another possible route would have been to use UTF-8 as the internal encoding form, retaining the existing 8-bit code units used in TEX as characters. However, this would have made it impossible (without major revisions) to provide properties such as character category (letter, other printing character, escape, grouping delimiter, comment character, etc.), case mappings, and so on to any characters beyond the basic ASCII set; and it would also require conversion when Unicode text is to be passed to system APIs. Overall, therefore, UTF-16 was felt to be the most practical choice, and the appropriate TEX data structures were systematically widened.

Extended TEX code tables Along with widening character codes from 8 to 16 bits, the TEX code tables that provide per-character properties were enlarged to cover the range 0...65,535. This means that XƎTEX has `\catcode`, `\sfcode`, `\mathcode`, `\delcode`, `\lccode` and `\uccode` values for each of the characters in Unicode's Basic Multilingual Plane. The default format files provided with XƎTEX initialize the `\lccode` and `\uccode` arrays based on case mapping properties from the Unicode Character Database, so that the `\uppercase` and `\lowercase` primitives will behave as expected. Figure 2 shows how these extended code tables might be used.

Because these arrays are indexed by the individual code units of the UTF-16 data used in XƎTEX, it is not possible to set these properties for characters beyond Plane 0. However, as these are mainly either CJK ideographs or characters of relatively obscure archaic scripts, it seems unlikely that there will be much need to change their `\catcode` values or apply case-changing commands.[5]

[5] Full use of math characters from Plane 1 is a separate issue, as math mode requires additional font and character properties.

```
\lowercase{DŽIN}
```

džin

```
\uppercase{Esi eyama klɔ míaƒe nuvɔwo ɖa vɔ la}
```

ESI EYAMA KLƆ MÍAƑE NUVƆWO ƉA VƆ LA

```
\catcode`王=\active \def王{...}
```

% defines an individual Chinese character as a macro

Figure 2: Per-character code tables in XeTeX support Unicode

Input encodings While XeTeX is designed to work with Unicode throughout the typesetting process, users may well wish to typeset text that is in a different encoding. By default, XeTeX interprets input text as being UTF-8, converting multi-byte sequences to Unicode character codes appropriately, unless inspection of the file suggests that the text is UTF-16 (identified by a Byte Order Mark code, or by null high-order bytes in the initial 16-bit code units). Either way, the input is assumed to be valid Unicode.

Existing TeX documents that use purely 7-bit ASCII are of course also valid Unicode (UTF-8); but documents in 8-bit encodings such as Windows Latin or Cyrillic codepages, legacy Mac OS character sets, or East Asian double-byte encodings cannot be interpreted this way. They will typically contain byte sequences that are not legal in UTF-8; but even if the bytes are not ill-formed when read as UTF-8, they will not result in the intended characters.

To address this problem, XeTeX provides two commands that allow the input to be converted from a different encoding into Unicode:

`\XeTeXinputencoding "`***codepage-name***`"`
: changes the codepage for the current input file, beginning with the next line of text

`\XeTeXdefaultencoding "`***codepage-name***`"`
: sets the initial codepage for subsequently-opened input files (does not affect files already open for reading)

These commands allow input text in a non-Unicode encoding to be converted (using the converters from the ICU library[6]) into Unicode as it is read. Thus, text in Latin-1 or Big5 or Shift-JIS or many other encodings can be typeset directly using Unicode-compliant fonts.

Note that output text, whether in the transcript file or files written using `\openout` and `\write`, will always be UTF-8 Unicode, regardless of the codepage or encoding form of the input text.

Hyphenation support Along with other character-code-oriented parts of TeX, the hyphenation tables in XeTeX have been extended to support 16-bit Unicode characters. This means that it is possible to write hyphenation patterns that use any (Plane 0) Unicode letters, including non-Latin scripts as well as extended Latin (accented characters, etc.) Figure 3 shows a fragment from a Sanskrit hyphenation file created by a XeTeX user. With the traditional TeX approach to such scripts, using complex macros and preprocessing, it would be much more difficult to support hyphenation patterns.

```
\patterns{
% break before or after any full vowel
1अ1
1आ1
1इ1
1ई1
1उ1
1ऊ1
1ऋ1
1ॠ1
% ...etc...
% break after vowel matra, but never before
2ा1
2ि1
2ी1
2ु1
2ू1
2ृ1
2ॄ1
2ॢ1
2ॣ1
% ...etc...
}
```

Figure 3: Hyphenation patterns using Devanagari letters

The implementation of native Unicode font support in XeTeX, treating each word as a "black box" measured as a unit by the font subsystem, made it easy to form paragraphs of such "boxes" without extensive changes to the overall algorithms. However, TeX's automatic hyphenation mechanism, which comes into effect if it is unable to find satisfactory line-break positions for a paragraph on the initial attempt, applies to lists of character nodes representing runs of text within a paragraph to be broken into lines. But when using Unicode fonts in XeTeX, the line-break process sees "word nodes" as indivisible, rigid chunks.

Explicit discretionary hyphens may of course be included in TeX input, and these continue to work in XeTeX, as they become discretionary break nodes in the list of items making up the paragraph. The word fragments on either side, then, would become separate nodes in the list, and a line-break can occur at the discretionary node between them.

[6] `http://www.ibm.com/software/globalization/icu/`

In order to provide automatic hyphenation support, however, it was necessary to extend the hyphenation routine so as to be able to extract the text from a word node, use TEX's pattern-based algorithm (and exception list) to find possible hyphenation positions within the word, and then replace the original word node with a sequence of nodes representing the (possibly) hyphenated fragments, with discretionary nodes in between.

One more refinement proved necessary here: once the line-breaks have been chosen, and the lines of text are being "packaged" for final justification to the desired width, any unused hyphenation points are removed and the adjacent word (fragment) nodes re-merged. This is required in order to allow rendering behavior such as character reordering and ligatures, implemented at the smart-font level, to occur across unused hyphenation points. With an early release of XƎTEX, a user reported that OpenType ligatures in certain words such as *different* would intermittently fail (appearing as *different*, without the *ff* ligature). This was occurring when automatic hyphenation came into effect and a discretionary break was inserted, breaking the word node into sub-words that were being rendered separately.

Typographic features

Beyond simply allowing the use of any font on the user's system, XƎTEX also provides access to various advanced typographic features of AAT and OpenType fonts, so that users can take advantage of the full richness of these fonts.

AAT font features AAT (Apple Advanced Typography) is the native Mac OS X technology for advanced fonts that provide typographic layout information (besides simple glyph metrics). An AAT font may contain tables that define layout features such as ligatures, alternate glyph forms, swashes, etc. These features may be specified by the font as being enabled by default, in which case XƎTEX will automatically use them; or they may be optional features that are only used when explicitly turned on.

The font designer provides names, stored in the font itself, for any features that are intended to be controlled by the user. While there is a registry of known features, designers are free to implement and name new behaviors in their fonts, so the possible set of features and settings is open-ended.

The extended `\font` command in XƎTEX allows AAT font feature settings to be specified as a list of *feature* = *setting* pairs appended to the name of the font. Feature settings that are enabled by default can also be turned off, by prefixing the setting name with '`!`'. Figure 4 illustrates a few optional features available in the Apple Chancery font.

Vertical text with AAT fonts An additional attribute that can be specified for AAT fonts in XƎTEX is `vertical`. This causes the text rendering system to use vertical text-layout

```
\font\x="Apple Chancery" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

The quick brown fox jumps over the lazy dog.

```
\font\x="Apple Chancery:
  Design Complexity=Simple Design Level;
  Letter Case=Small Caps" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

```
\font\x="Apple Chancery:
  Design Complexity=Flourishes Set A" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

The quick brown fox jumps over the lazy dog.

Figure 4: Selecting optional AAT font features

techniques, although it does not in itself re-orient the overall layout. Typically, glyphs will be rotated 90° counterclockwise, like this, and laid out according to their vertical rather than horizontal metrics.

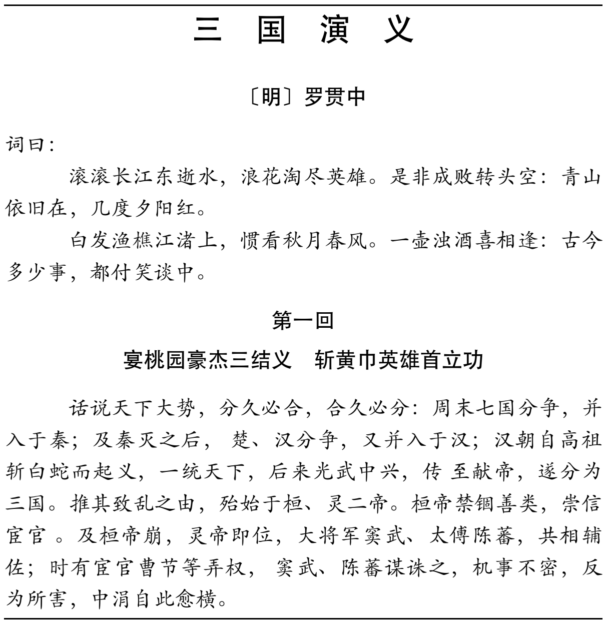
If this capability is combined with macros that rotate the text block as a whole, which is readily achieved through graphic transformations in the output driver (see figure 5), it becomes possible to typeset languages such as Chinese using a traditional vertical layout. Figures 6 and 7 show the same text formatted in horizontal and vertical styles. Note how certain glyphs such as the brackets do not undergo the same rotation as the rest of the text; the `vertical` attribute automatically gives the correct behavior here.

OpenType: optional features Like AAT fonts, OpenType fonts may also include layout features that can be enabled or disabled to affect the rendering of the text. Unlike AAT, there are no feature names provided in the font, only four-character "tags" (which are generally somewhat mnemonic). The expectation in OpenType is that all features will be officially registered with Microsoft and Adobe, and applications can then provide whatever user interface and names are needed for the features they choose to support.

XƎTEX takes a low-level approach, allowing feature tags to be used directly in the `\font` command in a similar way to AAT names; individual features can be turned on or off for any given font definition, using + or - with the four-character tag. Therefore, any features defined in the font can be used, even if not defined in the OpenType feature registry. (Macro packages could be used to provide more meaningful names; for example, the *fontspec* package for LATEX provides a unified interface for many registered features across both AAT and OpenType fonts.) Figure 8 shows a few examples of the use of OpenType feature tags to select alternate renderings of a font.

```
\newif\ifVertical \Verticaltrue % \Verticalfalse gives horizontal layout
\vsize=7in \hsize=4.5in \def\Vert{} % set up page size
\ifVertical % set parameters for vertical layout
  \hsize=7in \vsize=4.5in \def\Vert{:vertical} % attribute used in font defs
  % macro to rotate a box of Chinese text set with the "vertical" font attribute
  \def\ChineseBox#1{\setbox0=\vbox{\boxmaxdepth=0pt #1}\dimen0=\wd0 \dimen2=\ht0
    \vbox to \dimen0{\hbox to \dimen2{\hfil\special{x:gsave}\special{x:rotate -90}\rlap
        {\vbox to 0pt{\box0\vss}}\special{x:grestore}}\vss}}
  \def\ChineseOutput{\shipout \vbox{\ChineseBox{\makeheadline \pagebody \makefootline }}
      \advancepageno \ifnum \outputpenalty >-20000 \else \dosupereject \fi}
  \output={\ChineseOutput} \fi
\font\body="STKaiti\Vert" at 12pt \body
\font\bold="STHeiti\Vert" at 12pt  \font\title="STHeiti\Vert" at 18pt
\centerline{\title 三 国 演 义}
\bigskip \centerline{\bold〔明〕罗贯中}
% ...etc...
```

Figure 5: Using an AAT font attribute and graphic transformations to implement vertical typesetting


三 国 演 义

〔明〕罗贯中

词曰：

滚滚长江东逝水，浪花淘尽英雄。是非成败转头空：青山依旧在，几度夕阳红。

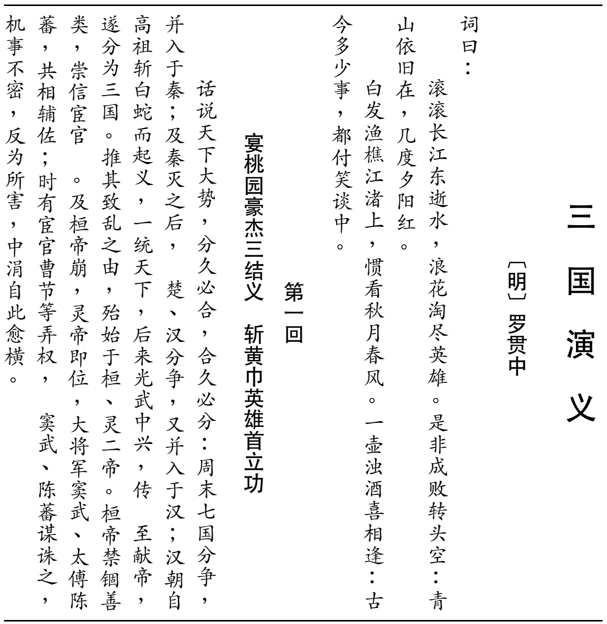
白发渔樵江渚上，惯看秋月春风。一壶浊酒喜相逢：古今多少事，都付笑谈中。

第一回

宴桃园豪杰三结义　斩黄巾英雄首立功

话说天下大势，分久必合，合久必分：周末七国分争，并入于秦；及秦灭之后， 楚、汉分争，又并入于汉；汉朝自高祖斩白蛇而起义，一统天下，后来光武中兴，传 至献帝，遂分为三国。推其致乱之由，殆始于桓、灵二帝。桓帝禁锢善类，崇信宦官 。及桓帝崩，灵帝即位，大将军窦武、太傅陈蕃，共相辅佐；时有宦官曹节等弄权， 窦武、陈蕃谋诛之，机事不密，反为所害，中涓自此愈横。

Figure 6: Chinese text in horizontal format


三 国 演 义

〔明〕罗贯中

词曰：

滚滚长江东逝水，浪花淘尽英雄。是非成败转头空：青山依旧在，几度夕阳红。

白发渔樵江渚上，惯看秋月春风。一壶浊酒喜相逢：古今多少事，都付笑谈中。

第一回

宴桃园豪杰三结义　斩黄巾英雄首立功

话说天下大势，分久必合，合久必分：周末七国分争，并入于秦；及秦灭之后， 楚、汉分争，又并入于汉；汉朝自高祖斩白蛇而起义，一统天下，后来光武中兴，传 至献帝，遂分为三国。推其致乱之由，殆始于桓、灵二帝。桓帝禁锢善类，崇信宦官 。及桓帝崩，灵帝即位，大将军窦武、太傅陈蕃，共相辅佐；时有宦官曹节等弄权， 窦武、陈蕃谋诛之，机事不密，反为所害，中涓自此愈横。

Figure 7: Chinese text in vertical format

OpenType: optical sizing Some OpenType font families include multiple faces designed for use at different sizes; for example, the Adobe Brioso Pro family includes Caption, Text, Subhead, Display, and Poster faces, each optimized for a different range of point sizes. If the full collection of fonts has been installed, XƎTEX will use the OpenType "size" feature to automatically select the appropriate face for the point size used, as shown in figure 9. Generally, this automatic behavior is helpful; however, it can be overridden if necessary by using a `/S=`*optical-size* modifier on the font name. Figure 10 shows several different optical sizes of Brioso used at the same physical size, making the design difference between the faces more apparent to the eye.

OpenType: script and language In addition to optional typographic features, OpenType fonts may include layout features that are necessary for the correct rendering of complex writing systems such as Arabic or Indic scripts. To apply these features, it is necessary to have a "shaping engine" that applies the appropriate feature tags to individual characters of the text. There are specific rules for each supported script, and complex scripts will only render properly

```
\font\x="Brioso Pro" \x Hello World! 0123456789
```
Hello World! 0123456789
```
\font\x="Brioso Pro:+smcp" \x Hello World! 0...
```
HELLO WORLD! 0123456789
```
\font\x="Brioso Pro:+sups" \x Hello World! 0...
```
H^{ello} W$^{orld!}$ ^{0123456789}
```
\font\x="Brioso Pro Italic:+onum" \x Hello W...
```
Hello World! 0123456789
```
\font\x="Brioso Pro Italic:+swsh,+zero" \x H...
```
Hello World! Ø123456789

Figure 8: Selecting OpenType feature tags

```
\font\x="Brioso Pro" at 7pt \x Hello World!
```
Hello World! (Brioso Pro Caption)
```
\font\x="Brioso Pro" at 10pt \x Hello World!
```
Hello World! (Brioso Pro Text)
```
\font\x="Brioso Pro" at 16pt \x Hello World!
```
Hello World! (Brioso Pro Subhead)

Figure 9: Automatic optical sizing

```
\font\x="Brioso Pro/S=7" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Caption)
```
\font\x="Brioso Pro/S=10" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Text)
```
\font\x="Brioso Pro/S=16" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Subhead)

Figure 10: Overriding normal optical sizing

```
\font\x="Code2000" \x العربي हिन्दी
```
ي ب ر ع ل ا हिन्दी *default (Latin) features only; incorrect rendering of both scripts*
```
\font\x="Code2000:script=arab" \x العربي
```
العربي *correct Arabic script rendering*
```
\font\x="Code2000:script=deva" \x हिन्दी
```
हिन्दी *correct Devanagari script rendering*

Figure 11: Specifying OpenType shaping engines using the `script=...` feature

```
\font\Brioso="Brioso Pro"
\Brioso …gelen firmaları…tarafından…
```
…gelen firmaları…tarafından…
```
\font\BriosoTrk="Brioso Pro:language=TRK"
\BriosoTrk …gelen firmaları…tarafından…
```
…gelen firmaları…tarafından…
Turkish requires the i/ı distinction maintained

```
\font\D="Doulos SIL/ICU"
\D cung cấp một con số duy nhất cho mỗi ký tự
```
cung cấp một con số duy nhất cho mỗi ký tự
```
\font\V="Doulos SIL/ICU:language=VIT"
\V cung cấp một con số duy nhất cho mỗi ký tự
```
cung cấp một con số duy nhất cho mỗi ký tự
Vietnamese uses different diacritic positioning

Figure 12: Using alternate language systems in OpenType fonts to achieve correct rendering

if the correct engine is specified in the `\font` command, as illustrated in figure 11. (Note that this is different from the situation with AAT fonts, where complex rendering behavior is programmed entirely in the font tables, and no script-specific engine is needed.)

OpenType fonts may also support multiple "language systems" to handle differences in the appropriate rendering for different languages. For example, many serifed Latin fonts include an *fi* ligature, and this will normally be enabled by default. However, Turkish makes a distinction between *i* and *ı* (dotless *i*). Using an *fi* ligature typically causes this distinction to be lost, and therefore this ligature must be disabled in the Turkish language system. Another example of language-specific behavior occurs in Vietnamese, where the positioning of multiple diacritics on a base character differs from the default vertically-stacked behavior that would be expected elsewhere. When loading an OpenType font in XƎTEX, the desired language tag can be included in the `\font` command to control the behavior, as shown in figure 12.

Font mappings In addition to the font-specific AAT and OpenType features that can be included in a `\font` command, XƎTEX has a general-purpose mechanism known as "font mappings" that can be applied to any native font.

To understand the purpose of font mappings, consider TEX input conventions such as `---`, which normally generates an em-dash, or ``` `` ```, which generates an opening double quote. These conventions are not built into TEX, nor are they generally implemented in TEX macros (like most other "extended" characters); rather, they are implemented as ligatures in the Computer Modern fonts, and similar ligature rules have been created in most other fonts configured for use with TEX.

However, these ligatures, unlike standard typographic ligatures such as *fi*, are not generally known or used

```
LHSName "TeX-text"
RHSName "UNICODE"

pass(Unicode)
U+002D U+002D <> U+2013 ; -- -> en dash
U+002D U+002D U+002D <> U+2014 ; --- -> em dash

U+0027 <> U+2019 ; ' -> right single quote
U+0027 U+0027 <> U+201D ; '' -> right dbl quote
U+0022  > U+201D ; " -> right double quote

U+0060 <> U+2018 ; ` -> left single quote
U+0060 U+0060 <> U+201C ; `` -> left dbl quote

U+0021 U+0060 <> U+00A1 ; !` -> inv. exclam
U+003F U+0060 <> U+00BF ; ?` -> inv. question
```

Figure 13: The `tex-text` font mapping

outside the TEX world. They were designed as a convenient workaround for limitations of the character set that could be entered on typical keyboards. But we cannot expect general-purpose fonts from outside the TEX world to implement these ligatures. Therefore, if a XƎTEX user working with a standard Unicode font enters ``` ``Help---I'm stuck!'' ```, the result is likely to be something like ``*Help---I'm stuck!*'', which is not what was intended.

One solution is to convert the input text to directly use the desired Unicode characters for quote marks, dashes, etc., but this may not be convenient where there are large amounts of pre-existing text. Even for new text, experienced TEX typists may be more comfortable continuing to use these conventions rather than learning new key sequences, or document portability between XƎTEX and standard TEX may require that they be used.

XƎTEX's font mappings can solve this issue. A font mapping is a transformation, expressed as mapping rules that convert Unicode characters or sequences from an "input" form (that found in the document text) to an "output" (the character or characters to be rendered from the font). Such mappings are written in the TECkit mapping language.[7] A `\font` command may include a `mapping=`*filename* qualifier, and XƎTEX will then apply the given mapping as part of the text rendering process when using that font. An example `tex-text` mapping is included with XƎTEX to implement the common ligatures found in Computer Modern fonts; figure 13 shows the TECkit source of this mapping file. If this mapping is loaded along with a standard Unicode font, then the TEX-style input text ``` ``Help---I'm stuck!'' ``` will render as expected: *"Help—I'm stuck!"*.

[7] `http://scripts.sil.org/teckit/`

```
\def\SampleText{Unicode - это уникальный
 код для любого символа, независимо от платформы,
 независимо от программы, независимо от языка.}
\font\gen="Gentium"
\gen\SampleText\par
```

Unicode - это уникальный код для любого символа, независимо от платформы, независимо от программы, независимо от языка.

```
\font\gentrans="Gentium:mapping=cyr-lat-iso9"
\gentrans\SampleText\par
```

Unicode - èto unikal'nyj kod dlâ lûbogo simvola, nezavisimo ot platformy, nezavisimo ot programmy, nezavisimo ot âzyka.

Figure 14: Using a font mapping to render the same text in its native script and transliterated

While font mappings were originally implemented to provide compatibility with TEX typing conventions, they can be used in other ways, too; figure 14 shows an example where the same input text is printed both in its original form and in Latin transliteration, using a Cyrillic/Latin transliteration mapping associated with the font.

Asian-language linebreaking

A number of east and south-east Asian languages, such as Chinese, Japanese, Thai, and others, are normally written without word spaces. The only spaces in the text may be between phrases or sentences, or even entire paragraphs may be lacking any space characters. Hyphenation is also not used in many of these languages. This presents a problem for line-breaking, as TEX normally expects to find interword glue where line-breaks can be attempted.

Line-break positions To support typesetting text in such languages, XƎTEX includes a feature known as "locale-based line-breaking", based on the Unicode line-break algorithm implemented in the ICU library. The command `\XeTeXlinebreaklocale="`*locale-code*`"`, where the *locale-code* is a standard locale (language/region) code, tells XƎTEX to look for possible line-break positions according to the rules of the given locale; the paragraph can then be broken at these places despite the lack of spaces or hyphenation rules.

Justification In addition to the problem of finding legitimate line-break positions, the lack of inter-word glue also makes it difficult for TEX to justify the lines. One option, of course, is ragged-right typesetting, and this may be the appropriate solution if a rigid character grid (as sometimes seen in Chinese, for example) is to be maintained. However, another option is to set the parameter `\XeTeXlinebreakskip` to a slightly stretchable glue value.

```
\def\thaitext{%
โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข.
คอมพิวเตอร์จัดเก็บตัวอักษรและอักขระอื่นๆ
โดยการกำหนดหมายเลขให้สำหรับแต่ละตัว.
ก่อนหน้าที่ Unicode จะถูกสร้างขึ้น, ได้มีระบบ encoding
อยู่หลายร้อยระบบสำหรับการกำหนดหมายเลขเหล่านี้.}
\font\thai="Thonburi" at 10pt
\thai \thaitext
```

โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข. คอมพิวเตอร์จัดเก็บตัวอักษรและอักขระอื่นๆ
โดยการกำหนดหมายเลขให้สำหรับแต่ละตัว. ก่อนหน้าที่ Unicode จะถูกสร้างขึ้น, ได้มีระบบ encoding อยู่หลายร้อยระบบสำหรับการกำหนดหมายเลขเหล่านี้.

Thai text with spaces only between phrases

```
\XeTeXlinebreaklocale "th"
\XeTeXlinebreakskip=0pt plus 1pt
\thai \thaitext
```

โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข. คอมพิวเตอร์จัด เก็บ ตัว อักษรและอักขระ อื่นๆ โดย การ กำหนด หมาย เลข ให้ สำหรับ แต่ ละ ตัว. ก่อนหน้าที่ Unicode จะ ถูก สร้าง ขึ้น, ได้ มี ระบบ encoding อยู่ หลาย ร้อย ระบบ สำหรับ การกำหนดหมายเลขเหล่านี้.

Using locale-based line-breaking to improve results

Figure 15: Line-breaking and justification without word spaces

XƎTEX will then insert this glue at each *potential* break position found by the line-break algorithm, which makes the overall text slightly stretchable and allows fully justified setting. Figure 15 illustrates the use of the Asian line-breaking parameters.

There is also a parameter `\XeTeXlinebreakpenalty` that can be set to control the desirability of inter-character breaks found by the Unicode algorithm, as compared to normal breaks at other penalties and glue.

Built-in graphics support

TEX traditionally knows nothing about graphics, leaving this to output drivers and merely passing along information from `\special` commands. It is left to macro packages and users to determine the amount of space that an included image occupies; the `\special` that causes the image to be included by the driver does not itself take any space during the typesetting process.

XƎTEX provides an alternative approach, adding primitive commands that actually load graphic files during typesetting. The advantage of this is that the typesetting process can know the size of the image; typically, it is loaded into an `\hbox`, and macros can then examine the `\wd` and `\ht` of that box to make decisions about layout, or to re-load the image with different scaling, etc.

```
\centerline{%
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100}\quad
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100 xscaled 2000}\quad
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100 rotated 90}}
```

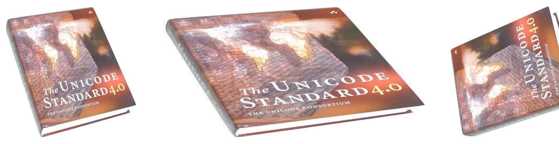


Figure 16: Including graphics in a XƎTEX document

QuickTime-based graphics The XƎTEX primitive command `\XeTeXpicfile "`*filename*`"` locates and includes the named graphic file, which may be in any format recognized by the QuickTime library on Mac OS X. This includes common image formats such as `.jpg`, `.bmp`, `.tiff`, `.png`, and many others. A number of keywords such as `width`, `height`, `scaled`, and `rotated` may be used after the filename to transform the image. Figure 16 shows some simple examples of image inclusion.

PDF documents One of the formats supported by the `\XeTeXpicfile` command is `.pdf`; however, if a PDF graphic is included in this way, it will be rendered as a raster image at relatively low resolution. It is better to use an alternative command, `\XeTeXpdffile`, which includes the specified PDF in its native form, complete with vector graphics, embedded fonts, etc. `\XeTeXpdffile` also supports an additional keyword `page` to select the required page from a multi-page PDF document.

Note that there is a `xetex.def` driver available for the standard LATEX `graphics.sty` and `graphicx.sty` packages; this driver will automatically use the XƎTEX primitives to implement the higher-level `\includegraphics` command, and will choose the proper XƎTEX function depending on the type of graphic file.

LATEX packages

Many users like to combine the Unicode and font support of the XƎTEX engine with the document markup and formatting features of LATEX. In most cases, this works well; the exceptions typically involve LATEX packages dealing with input and font encodings (which are generally superfluous in a Unicode-based process) or packages that depend on the features of a particular output driver (such as drawing packages that rely on `dvips` or `dvipdfm` `\specials`, or on pdfTEX extensions). In some cases, such packages may need to be adapted to work with the `xdv2pdf` driver; in others, the output driver features needed may not currently be available.

```
\usepackage{fontspec} % load fontspec.sty
\setmonofont[Scale=0.8]{Andale Mono WT J}
  % use scaled Andale Mono for \tt
\defaultfontfeatures{Mapping=tex-text}
  % load the tex-text font mapping by default
\setromanfont{Adobe Garamond Pro}
  % use Garamond Pro as \rm, etc
```

Figure 17: Use of `fontspec.sty`, from the preamble of this document

In addition to the `xetex.def` driver files for the standard LaTeX graphics and color packages, allowing these to be used with the XeTeX engine, two important packages written specifically for XeTeX deserve mention.

fontspec The `fontspec.sty` package, written by Will Robertson, provides a high-level interface to native Unicode fonts in XeTeX, integrating them with the LaTeX font selection mechanism, and supporting a wide range of features in both AAT and OpenType fonts. Extensive documentation is available with the package; figure 17 shows a simple excerpt from the preamble of this document. These few lines are sufficient to set up all the typefaces needed for this document, except those used within figures to illustrate specific points. Note that there are no auxiliary `.tfm`, `.fd`, `.sty`, or other TeX-specific files associated with the fonts used here; they are simply installed in the **~/Library/Fonts** folder in the standard Mac OS X manner.

xunicode To improve support for standard LaTeX documents when using Unicode fonts, Ross Moore has provided `xunicode.sty`. This package reimplements many of the control sequences used in LaTeX for accents, symbols, and other "special" characters, mapping them to the correct Unicode codepoints instead of to their locations in traditional TeX fonts. This allows documents that use these symbols via their LaTeX names to run unchanged under XeTeX, with the correct Unicode characters being rendered in the output.

XeTeX and ConTeXt

While LaTeX is probably the macro package most commonly used with XeTeX, it is also possible to use ConTeXt. My understanding is that the standard ConTeXt distribution includes an option to use the XeTeX engine in place of the default pdfTeX. A brief example of how XeTeX font support can be used in ConTeXt is shown in figure 18. There is further information on the ConTeXt Wiki site,[8] from which this example was copied.

```
\definedfont["Hoefler Text:mapping=tex-text;
  Style Options=Engraved Text;
  Letter Case=All Capitals" at 24pt]
Big Title
```

BIG TITLE

Figure 18: Loading a native Unicode font in ConTeXt

Future directions

In conclusion, a few comments on the possible future of XeTeX. The system has been publicly available for about 18 months as of the time of writing, and has been used for a wide range of document types and languages. While it remains a "work in progress", it appears to work reliably for most users, within the limitations of its design.

Besides on-going bug fixes and minor features, there are several major enhancements that could be undertaken to further improve XeTeX:

- Enhanced PDF back-end, via one of several approaches:
  - leverage improved PDF support in Mac OS X 10.4
  - new `xdv2pdf` driver based on `dvipdfmx`
  - integration with pdfTeX output routine
- True Unicode math support:
  - requires extensions to `\mathchar` etc., and underlying structures
  - also requires extended (at least 16-bit) font metric format
  - may be possible to make use of code from Omega/Aleph
- XeTeX for non-Mac OS X platforms:
  - should include full integration with TeX Live sources

Assistance towards implementing any and all of these ideas, or others, is most welcome! The XeTeX source code is currently available in a Subversion repository at `svn://scripts.sil.org/xetex/TRUNK`; this URL may change at some point, but the XeTeX web pages at `http://scripts.sil.org/xetex` should always indicate where to look.

[8] See `http://wiki.contextgarden.net/XeTeX` and `http://wiki.contextgarden.net/Fonts_in_XeTeX`.

Hóng-Zì: A Chinese METAFONT

Javier Rodríguez Laguna
SISSA, Trieste (Italy)
jrlaguna@sissa.it
http://www.sissa.it/~jrlaguna

Abstract

Hóng-Zì (红字) is a new Chinese METAFONT, still at a very early stage of development, freely available at http://hongzi.sourceforge.net under the GPL. The structure is split into four levels of abstraction: strokes, radicals, distributions and full characters, such that the highest level description of the characters becomes truly simple.

1 Introduction

Chinese characters (*hanzi*) and their close relatives[1] appear to be specially suitable for treatment under METAFONT [KNU 86]. Their shapes, notwithstanding their complexity, are made up of a finite (though large) number of parts. But these parts are not just pasted inside a box, like letters in a word. The parts change their shape depending on the size and position where they are going to be inserted. METAFONT is an approach to font design which is especially suited for these shape modifications.

Nonetheless, the TEX community still lacks a full fledged Chinese METAFONT. Of course, it is easy to include CJKV text in a TEX document using packages which employ other types of fonts, but then we lack the tunability and high quality to which we are accustomed.

The first development of a Chinese METAFONT was performed by Gu Guoan and John Hobby in the METAFONT-79 dialect [GH 84], which is not compatible with the current version. In the November 1982 issue of Scientific American, Douglas Hofstadter described his own project, in collaboration with David Leake, called *Hàn-Zì* [HOF 82], [HOF 86]. Unfortunately, it was abandoned shortly after.

In 1993, Martin Dürst provided an interesting approach for a Chinese METAFONT [DÜR 93] which, as far as the present author knows, did not have a follow-up. The recent proposal of Candy L.K. Yiu and Wai Wong [YW 03] may be more fruitful. In it, the authors describe a language for the description of *hanzi*, known as *HanGlyph*, and a METAPOST program called CCSS (Chinese Character Synthesis System) which renders the characters. Although the approach introduced in this paper was developed independently, it shares many features with the ideas of Yiu and Wong.

Hóng-Zì (红字) was born as an attempt to overcome this gap in the TEX world. It is supported by SourceForge and its first release was published in May 2003 at http://hongzi.sourceforge.net. It is organized into four levels of abstraction, so that reading (and writing) the highest level files is easy even for a METAFONT-newbie. This way, non-experts should be able to help in its development.

The current version, described in this paper, is Hóng-Zì 0.5. Although this distribution contains only 126 characters, it has the potential to build many more. The characters are composed from 69 *radicals*, which are drawn using 32 different types of strokes. The basic mechanism is probably fixed now, although some details may change in subsequent versions. The main tasks are now (a) the development of more radicals and more characters and (b) the design of a nice TEX interface.

Section 2, which is the core of this paper, is a description of the abstraction levels of Hóng-Zì. This paper ends with a discussion of the open problems and future work.

2 Abstraction levels of Hóng-Zì

Hanzi are made up of parts which, going beyond the strict meaning of the term, we shall name *radicals*. (Strictly speaking, radicals are the 214 *hanzi* parts used by Chinese dictionaries for indexing.) Radicals are distributed in certain ways inside the character, always fitting a square box. (To learn more about the structure of *hanzi*, I recommend the books

[1] Chinese *hanzi* (both simplified and traditional), Japanese *kanji*, Korean *hanja* (and even *hangul*) and Vietnamese *chữ hán* and *chữ nôm*. Writing systems from these countries are usually termed CJKV text. See [LUN 98] for details.

[HEI 01] and [FAZ 86], and the program *Hanzi Master* (`hanzim`) [ROB 02].)

In summary, *characters* consist of *radicals*, *distributed* in certain ways. Radicals, in turn, consist of *strokes*. The distribution of Hóng-Zì contains (at this moment) four files, one for each abstraction level. Let us describe each one separately.

(a) Strokes. This is the lowest level, the one which only uses METAFONT primitives. There are 32 stroke functions defined in the current version. Each one requires either two, three or four parameters. Some examples: `point(cx,cy)` only needs two parameters, which correspond to the coordinates of the peak of the point. The stroke `hook_v(cx,cy,l)` (which draws a vertical hooked bar) requires three: the coordinate of the starting point and the length. And `down_right(cx,cy,lx,ly)` (which draws a stylized stroke going down and right) requires four: the coordinates of the starting point, and the width and height of a box which contains it.

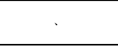
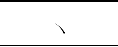
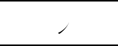
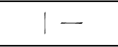
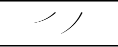
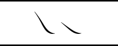
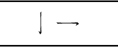
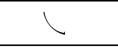
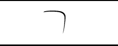
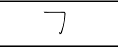
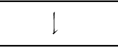
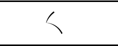
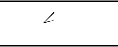
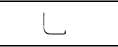
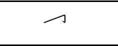
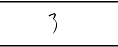
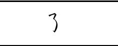
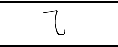
Prototype	Shape
`point(cx,cy);`	
`point_sized(cx,cy,lx,ly);`	
`point_ne_sized(cx,cy,lx,ly);`	
`bar_v(cx,cy,l);`	
`down_left(cx,cy,lx,ly);`	
`down_right(cx, cy, lx, ly);`	
`hook_v(cx, cy, l);`	
`hook_ob(cx, cy, lx, ly);`	
`corner_tr(cx, cy, lx, ly);`	
`corner_hook(cx, cy, lx, ly);`	
`vert_raise(cx, cy, l);`	
`nu_stroke(cx, cy, lx, ly);`	
`angle(cx, cy, lx, ly);`	
`ell(cx, cy, lx, ly);`	
`ye_stroke(cx, cy, lx, ly);`	
`three(cx, cy, lx, ly);`	
`three_hook(cx, cy, lx, ly);`	
`spoon_stroke(cx, cy, lx, ly);`	

Table 1. Some of the stroke functions defined in Hóng-Zì. Strokes with more than one shape are polymorphic.

Some strokes are *polymorphic*; for instance, the shape of a `down_right(0f,10f,8f,2f)` is not just a rescaling of a `down_right(0f,10f,8f,8f)`. The most important parameter which decides the shape is (usually) the aspect ratio, $A :=$ height/width.

Table 1 shows some of the stroke functions defined so far, along with some of the possible output. The complete list is in the file `strokes.mf` in the current distribution.

(b) Radicals. The term *radical* is extended in this work to cover any combination of strokes we regard as a unit inside a *hanzi*. Each radical function takes four parameters: coordinates of the upper-left corner, width and height. Radicals are given "meaningful" names in English, such as `child` (子) or `eye` (目). It is not always easy to find an appropriate name for a given radical.

Many radicals are also polymorphic. For example, let us consider the radical `water`. if the aspect ratio is $A > 2$, its appearance is: 氵. But if $A \leq 2$, it looks like the full character: 水. Table 2 shows some of the radicals defined in Hóng-Zì, as described in file `radicals.mf`.

Radical name	Rendered form
`water`	水 氵
`roof`	宀
`child`	子
`woman`	女
`tongue`	舌
`omen`	氺
`person`	人 亻
`heart`	心 忄 ⺗
`sun`	日
`grass`	艹
`moon`	月
`fire`	火 灬
`ghost`	鬼
`way`	辶
`knife`	刀 刂
`rice`	米

Table 2. A few of Hóng-Zì's radicals. Polymorphic radicals show some of their possible renditions.

(c) Distributions. A number of binary operators have been defined which are useful for distributing radicals on the character. E.g.: the `H(·,·)` operator composes a single radical out of two, distributing them horizontally. Each radical get 50% of the width. It is similar to the TeX expression `\hbox{X Y}`, but taking into account the fact that radicals have no natural height or width.

If some space must be left between the radicals, we may use `H_(·,·)`, which leaves 10% of blank space. The `Hl` series, which splits horizontally and gives more space to the left part) is depicted in full. We have three equivalent series: `Hr` (horizontal splitting, more space to the right part), `Vu` (vertical splitting, more space to the upper part) and `Vd` (vertical splitting, more space to the lower part).

There are a few other distribution operators which do not fall into any of the above four series: the L-box `L(X,Y)` operator, used in 道 *dāo* (road); the inner-box `I(X,Y)` operator, used in 国 *guó* (country); and the `Is(X,Y)` operator, which leaves a smaller inner box, as in 问 *wèn* (question).

Table 3 shows some of the distribution operators created so far. They are defined in the source file `distributions.mf`.

Of course, these operators may be composed, thus providing a way to describe complex characters, as we shall discuss in the following section.

(d) Full characters. All the lower layers are designed to make the highest level description of characters easy. As an example, the Hóng-Zì code for 明 *míng* (luminosity), as expressed in our file `hongzi.mf`, is just

```
H_rr(sun)(moon)
```

This line is to be read as follows: "Make up a character whose left part is the radical for sun and whose right part is the radical for moon. Give much more space to the right part and leave some space between them". In the real code, this line is enclosed by a `zi...iz` pair.

A more complex character is 凉 *liáng* (cold),

```
Hrr(ice)(Vdd(above)(Vd(box)(small)))
```

METAFONT reads this complex line *downwards*, like this: it first prepares a big box and splits it horizontally (giving more space to the right part, `Hrr`). It fills the left part with the `ice` radical. The right part, on the other hand, is processed further. First, it is split vertically (giving much more space to the lower part, `Vdd`). In the upper *half*, it renders the `above` radical. The lower part is again split (giving slightly more space to the lower part, `Vd`), and there the `box` and `small` radicals are rendered.

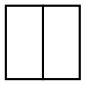
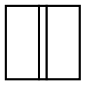
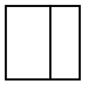
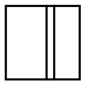
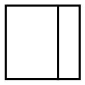
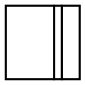
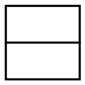
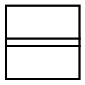
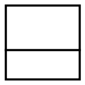
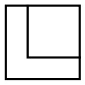
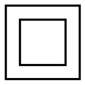
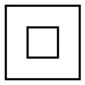
Distribution operator	Effect
`H(X,Y)`	
`H_(X,Y)`	
`Hl(X,Y)`	
`H_l(X,Y)`	
`Hll(X,Y)`	
`H_ll(X,Y)`	
`V(X,Y)`	
`V_(X,Y)`	
`Vu(X,Y)`	
`L(X,Y)`	
`I(X,Y)`	
`Is(X,Y)`	

Table 3. Some of Hóng-Zì's distribution operators.

Table 4 shows the code for some full characters, ordered according to their complexity.

3 Open problems and future work

The basic structure of the project seems to be reasonably fixed, although some issues of design must still faced. For example, regarding the distribution operators, perhaps we should replace the `H`, `Hl`, `Hll` series with a single operator containing an extra parameter. Also, the character descriptions should make up a sort of dictionary, containing the shape

Hanzi	Pinyin	English
女	*nǚ*	woman
`woman`		
水	*shǔi*	water
`water`		
好	*hǎo*	good
`H(woman)(child)`		
安	*ān*	peace
`Vdd(roof)(woman)`		
迷	*mí*	lost
`L(way)(rice)`		
谜	*mí*	riddle
`Hrr(word)(L(way)(rice))`		
孬	*nāo*	not good
`Vd(no)(H(woman)(child))`		
花	*huā*	flower
`Vdd(grass)(H(person)(spoon))`		
影	*yǐng*	shadow
`H_ll(Vdd(sun)(Vdd(above)(Vd(box)(small))))(beard)`		

Table 4. Character descriptions as they appear in the `hongzi.mf` file. They become quite easy at the highest level. Even complex characters such as 影 are explained in a single line.

description, pinyin, English translation and some compounds.

But the main problems of Hóng-Zì at this stage are that we must have (a) many more radicals and characters; (b) calligraphic improvements and different sets of strokes; and (c) a nice TEX interface. Therefore, we need and welcome new volunteers for collaboration! Deep knowledge of METAFONT is not needed as much as deep knowledge of the Chinese language, as can be seen from the high-level structure. And of course, extensions to other scripts in the CJKV family would be very nice.

The project web page, `http://hongzi.sf.net`, aspires to contain a full-fledged introduction both to the Chinese language for non-native speakers, and to CJKV typing.

Suggestions, comments, criticisms and, most of all, offers to help, are most welcome.

Acknowledgements

The author acknowledges Douglas R. Hofstadter and the organizers of TUG'05 for their encouragement and very useful comments. This project is supported by SourceForge (`http://sourceforge.net`).

Bibliography

[DÜR 93] Martin J. Dürst, *Coordinate-independent font description using Kanji as an example*, Electronic Publishing **6**(3), 133–143 (1993).

[FAZ 86] Edoardo Fazzioli, *Caratteri cinesi*, Mondadori (1986).

[GH 84] John D. Hobby and Gu Guoan, *A Chinese Meta-Font*, *TUGboat* **5**(2), 119–136 (1984). `http://cm.bell-labs.com/who/hobby/pubs.html`

[HEI 01] James W. Heisig, *Remembering the Kanji*, Tokyo: Japan Publishing Co. (2001).

[HOF 82] Douglas R. Hofstadter, *Changes in default words and images, engendered by rising consciousness*, Sci. Am., Nov. 1982.

[HOF 86] Douglas R. Hofstadter, *Metafont, metamathematics and metaphysics*, in *Metamagical themas* (chapter 13), Penguin Books (1986).

[KNU 86] Donald E. Knuth, *The METAFONTbook*, Addison-Wesley (1986).

[LUN 98] Ken Lunde, CJKV *information processing*, O'Reilly (1998).

[ROB 02] Adrian Robert, *Hanzi Master, a Chinese character learning-aid program* (2002). `http://zakros.ucsd.edu/~arobert/hanzim.html`

[YW 03] Candy L.K. Yiu and Wai Wong, *Chinese character synthesis using METAPOST*, *TUGboat* **24**(1), 85–93 (2003).

Qin notation generator

Candy L. K. Yiu, Jim Binkley
Department of Computer Science
Portland State University
Portland, OR USA
[candy,jrb]@cs.pdx.edu

Abstract

A Chinese character is a two-dimensional typological representation of strokes and radicals. This is unlike English letters, where there are only 26 letters which combine in one dimension. As there are thousands of Chinese characters, any Chinese character font requires more space (typically two bytes) for its computer encoding. Additionally, it takes more time for the creation of a complete font set.

Another problem is web communication. If a character is not available from the client's font set, there is no clean way to put the information in the web document. This is what originally motivated us to work on dynamic character generation.

The Qin is a Chinese musical instrument with a music specification language that serves as a good example of Chinese character stroke and radical combinations. The notation which represents the music is made with a set of characters and radicals. People who show Qin notation on the web have to scan an image and put it on the web. This project attempts to use Qin notation generation to demonstrate the possibility of Chinese character rendering.

This project is divided into two parts: the first is the Qin notation description language, and the second is the web notation generator. The description language will be based on Hanglyph, which is a syntax able to describe the hierarchical structure of Chinese characters. The Qin description language will use the basic characters and components. The rendering will construct the notation using MetaPost, based on Hanglyph. A web generator will output the pages with the Qin notation to allow communication over the internet.

1 Introduction to the Guqin

The guqin [5] (old zither) has a long history in China. It is mentioned in the Book of Odes (Shi Jing) and Confucius is said to have played it, thus we know it existed long before 200 B.C.E. Some have said that the shape of the instrument has not changed since the late Han dynasty, which is roughly two thousand years ago, between 200–500 C.E. Although many of the current pieces played on the Qin are from the Ming and Qing dynasties and may be "only" 200 to 500 years old, some pieces exist that may have started musically in B.C.E. times; for instance, the famous piece "Flowing Water" (Liu Shui). This piece is said to have been created by the legendary Qin player Bo Ya in the Spring and Autumn period. Flowing Water was included on the Voyager satellite launched in 1977, as played by the famous 20th century Qin master Guan Ping-hu. Thus the piece has been performed by Qin players for three millennia.

The Qin belonged to the old scholar class who ruled China until the fall of the Qing dynasty in 1911. The scholar class was said to practice "qin qi shu hua", "the four arts of the gentleman":

qin, the art of playing the qin
qi, the art of playing go
shu, the art of calligraphy
hua, the art of painting

They celebrated the Qin in poetry and in painting. Emperors played it, poets such as Li Bo mentioned it in poetry, and painters would often include it in a painting. The scholar class invested the Qin with an ideology that could manifest itself as a subtle form of reflective meditation through playing or view the Qin as a physical object of connoisseurship. Much of the music for the Qin is said to be thematic in the sense that it is related to nature

This work was supported as part of the Google 2005 Summer of Code, `http://code.google.com/summerofcode.html`.

(and Taoism), including pieces like Gao Shan (High Mountains), Liu Shui (Flowing Water), and Ping Sha Lo Yan (Geese at the Seashore). The Qin certainly has been influenced by Taoism, however other songs include thematic material taken from history, philosophy, and even occasionally romance.

As the Qin belonged to the so-called "writing class", scholars and musicians collected Qin songs, which they published in books called "qin-pu" or Qin handbooks. Hundreds of qin-pu exist with many songs (not all of which are currently played). Although there are a few individual songs on paper that predate the early Ming dynasty, in general most traditional books of songs date from around 1425 to the time of the Qing dynasty's collapse.

2 Qin tablature and the typesetting problem

The nature of traditional written Qin music is a gesture-oriented tablature. Complex symbols describe the motions of the left and right hands. Scholars believe that at some point between the Tang and Song dynasties a set of simplified Chinese characters was created as a shorthand form. These symbols took an old longhand verbose set of instructions for the left hand and right hand written in classical Chinese and combined them into a more terse form. Thus a composite Qin symbol was created that looks like an ordinary Chinese character, but is actually more variable internally than an ordinary Chinese character.

In a sense, this composite symbol is more like a sentence's worth of instructions in classical Chinese. It tells the player how to make a note with a combined left and right hand gesture. The basic components were taken from existing Chinese characters and could be combined and recombined to create simple and complex gestures that might, for example, say (see figure 1):

- using the left-hand thumb, at string 3, at stop position 10, play the string with the right-hand middle finger while pulling the string towards the player.
- follow this by sliding the left hand up to position 9 and do nothing with the right hand.
- now with the left hand not pressing any strings, with the right hand index finger play string 6 out from the player.

The "sentences" above are expressed with Qin symbols in a very concise form, requiring only a few symbols, as we can see in the figure.

The problem for typesetting is that any given symbol has a high degree of internal variability. The

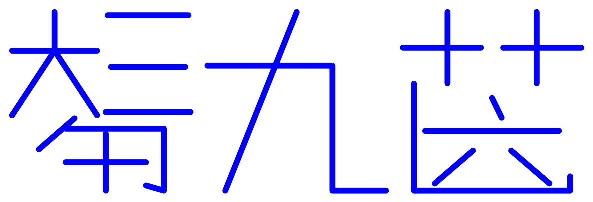


Figure 1: Example qin notation

Qin has seven strings and thirteen "stops" or left-hand position markers. For example, even though the first instruction above is relatively simple, we could change the string (1 to 7) and the left-hand stop position (1 to 13). This leads to 100 or so different symbols. We could also change the right-hand playing technique. One recent qin-pu from the 19th century [6] has about 50 separate sub-symbols for right-hand techniques and about 80 for left-hand techniques. Each stop position itself may be subdivided into 100 different parts. Thus a purportedly simple combination, without fancy glissando techniques, can easily lead to thousands of different combinations. This may be good for calligraphic artistry, but it is bad for font designers.

In short, although the traditional Qin tablature is made up out of component Chinese characters, the combined results may take many forms — and of course no existing Chinese font set would have them or be able to deal with the resulting complexity. Even today it is normal in the printing of modern Qin music for the transcriber to use traditional ink-based calligraphy to write out the Qin music and then somehow photographically insert the images onto the printed page.

From the point of view of font creation, there are many difficulties. For one, a composer could invent a new symbol. Furthermore existing handbooks in some cases document the individual character components, but the handbooks do not always agree on the atomic class of symbols! In addition, older handbooks from the Ming dynasty may use some symbols that have more or less dropped out of use in later handbooks. So although the Qin tablature system is rich in terms of tradition and semantic expression, it is fair to say it is not easily amenable to machine-based typography.

3 Structure of Qin notation

Qin notation is similar to Chinese characters. The notation can be decomposed into a number of parts called *components*. Each component consists of a number of strokes. Qin notation has a fixed number of components. Because of the fixed set of components, we can decompose Qin symbols in three ways: top-bottom, left-right, inside-out.

Figure 2 shows example Qin notation symbols which have been decomposed into all three components. In the figure, the four areas named *A, B, C* and *D* represent four components. *A* and *B* is a left-right relation which forms one element *AB*. *C* and *D* is a inside-out relation which forms another element *CD*. Now we can see that elements *AB* and *CD* form a top-bottom relation which gives you the complete symbol structure.

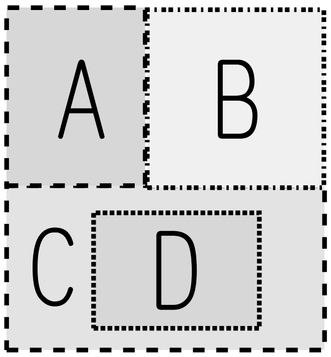


Figure 2: Example of Qin notation structure

Using the three combination operations above, we can build a hierarchical model of Qin notation based on the structure of the notation. Figure 3 shows an example which contains all 3 combinations. Figure 3 shows a Qin notation example which used the same combination structure in 2, and Figure 4 shows the hierarchical tree model after decomposing the Qin notation.

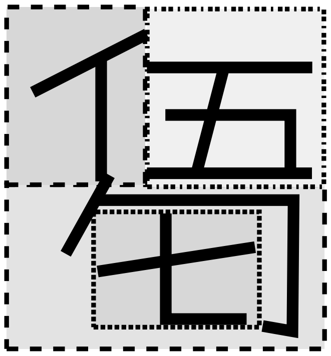


Figure 3: Qin notation decomposed elements

Thus we can use this model to build and combine single or composed elements into a new notation. We abstractly define the three combinations of the elements as follows:

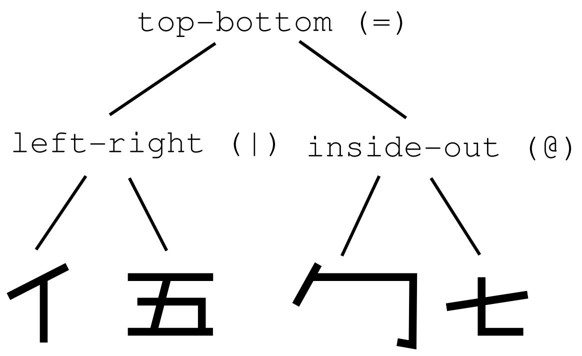


Figure 4: The hierarchical Qin notation tree

1. top-bottom: combine any two elements with one on the top of another.
2. left-right: combine any two elements with one on the left and another on the right.
3. enclosed: put one element inside another element, which has defined the enclosed area. This combination is only used when the outer element has an enclosed area.

As a result, the Qin notation generator structure is represented by a tree structure. Each internal node in the tree is any one of the above combinations (operators). The leaves of the tree are the basic pre-defined elements.

4 A Qin notation description language

In 2003, in papers entitled "Chinese Character Synthesis System using METAPOST" [3] and "Typesetting Rare Chinese Characters in LaTeX" [4], we defined a Chinese description language called Hanglyph (Chinese Description Language). This language can be used to give any Chinese character a typological representation. In Hanglyph, there are five operators and the system is based on strokes as a basic unit. In the full Hanglyph, there are also optional relations which can be used to specify more detail in the relation between the operands such as width, height, and alignment.

Our present Qin notation description language uses Hanglyph to describe the Qin notation. Qin notation, complex as it is, uses only a small subset of the more than 60,000 Chinese characters known, so it becomes a somewhat simpler problem. Therefore we only choose three operators without using any optional relations for the description of the Qin notation. However, to achieve this reduction, we require more work and information from the element library and a smarter generator engine. In the next section we will discuss the approach needed to achieve this goal.

Here is a formal definition of the Qin notation description language:

```
      ⟨Qin Notation⟩ ::= ⟨expr⟩+
              ⟨expr⟩ ::= ⟨Qin_char⟩ ⟨Qin_char⟩
                           ⟨parallel_operator⟩
                         | ⟨en_Qin_char⟩ ⟨Qin_char⟩
                           ⟨enclosed_operator⟩
 ⟨parallel_operator⟩ ::= ⟨top_bottom_operator⟩
                         | ⟨left_right_operator⟩
⟨top_bottom_operator⟩ ::= =
⟨left_right_operator⟩ ::= |
  ⟨enclose_operator⟩ ::= @
          ⟨Qin_char⟩ ::= ⟨en_Qin_char⟩
                         | ⟨basic_Qin_char⟩
       ⟨en_Qin_char⟩ ::= predefined element
                           allowing enclosed
    ⟨basic_Qin_char⟩ ::= predefined basic element
```

5 Basic components

Traditionally there are two kinds of basic components which are taught for Qin notation in handbooks. These components are combined to make composite symbols and can be said to belong to two sets: one for the left hand, and one for the right hand. Left hand and right hand sets of symbol elements are thus predefined. Many of the elements are Chinese character components or simplified versions of them.

Our basic elements come from a late Qing dynasty Qin handbook [6]. We use a pinyin derived variable name for the elements so that they can be encoded with our basic operators. There are 87 left hand components and 47 right hand components. Figure 5 shows some examples of basic components.

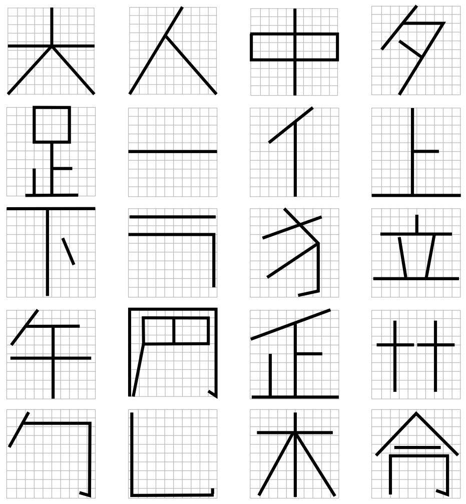


Figure 5: Examples of basic components

6 Notation generator

The notation generator generates the Qin notation in a geometric format such that we can convert the result into an image or font. It takes the Hanglyph input and generates the graphical representation as output. Figure 6 shows the structure of the Qin notation generator.

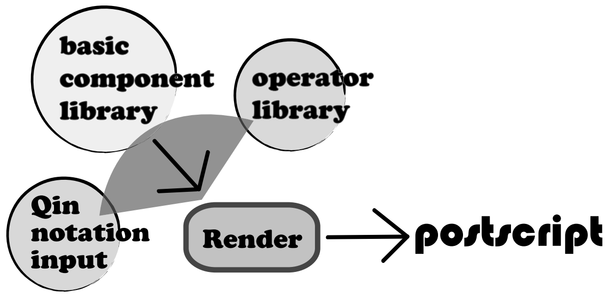


Figure 6: Qin notation generator architecture

The generator requires the basic components library and operation library, which are both written in MetaPost. We thus can divide the process into three stages:

1. MetaPost translator: Translates the Hanglyph input of the Qin notation into MetaPost output.
2. Qin notation in PostScript format: The generator uses the basic component library and performs desired operations using the operation libray to generate the Qin notation output in PostScript format.
3. Converter: The last step is to convert the PostScript format into whatever other format may be desired. In this project, we use a third-party tool, namely ImageMagick, to convert to png format in order to allow web display.

6.1 Basic component library

The basic component library is written as MetaPost macros, named for each component pinyin symbol. Figure 5 shows examples of basic components. For each component, we have control points of the path representing the component.

We also have a path creation macro to allow scaling the control points. This means when one scales the control point, one does not scale the path at the same time. Thus we achieve separation of control points and paths. This is very important because otherwise different and undesirable results may occur.

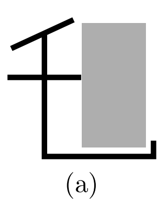
6.2 Operation library

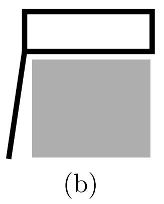
As mentioned before, we use only three operations: top-bottom, left-right and enclosed. The operation must determine the ratio of the two components and related spacing occupied by each component. In addition, the space between the two components also has to be determined, in order to check whether two components are touching each other.

Thus, the component library contains two properties, *enclosed area* and *surrounding*, to allow the operation to obtain the needed information for each component.

6.2.1 Enclosed area property

The enclosed area specifies the possible area for an element which can contain another element. The operation uses this information to scale the inner element as needed, combining both elements to generate output. Figure 7 shows three examples of possible enclosed components. The gray area represents the location of the inner element.





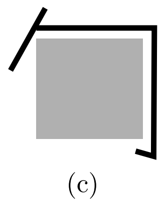


Figure 7: Examples of areas available for enclosed components

6.2.2 Surrounding property

The surrounding properties are defined in terms of four directions:

1. North (N): top side of the element.
2. East (E): left side of the element.
3. South (S): bottom side of the element.
4. West (W): right side of the element.

Furthermore, each direction can be flat or non-flat. Figure 8(a) shows the definition of each face direction. Zero represents flat and one represents non-flat. Below we show the equation to determine if spacing between two elements is needed.

$$space = 1, \text{ if } s_1 = s_2 \tag{1}$$

$$space = 0, \text{ if } s_1 \neq s_2 \tag{2}$$

For the left-right operator shown in figure 8(b), we consider the east face (right side) of the first operand and west face (left side) of the second operand. If the top-bottom operator is used as in figure 8(c), we consider the south face (bottom side) of the first operand and north face (top side) of the second operand.

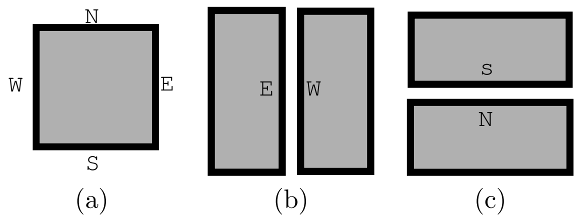


Figure 8: The direction of each component has a defined face: (a) face definitions; (b) left-right operations on faces; (c) top-bottom operations on faces

The Qin notation generator can use these properties and rules to determine the needed spacing between two operands. The equation above states that if there are two faces which have the same properties, either flat or non-flat, we insert space. Otherwise, two elements touch each other by default.

6.2.3 Area ratio

Our next concern is to determine the ratio of the visual area of the two operands. We use the elements below to consider how to estimate the ratio:

1. The number of strokes in each component.
2. The total length of the strokes in each component.
3. The width of each component.

The first row of figure 9 (1a,1b,1c) shows different possible area ratios occupied by each operand. The Qin notation generator has to calculate and determine the ratio of each operand in order to give a good visual output for the notation. The second

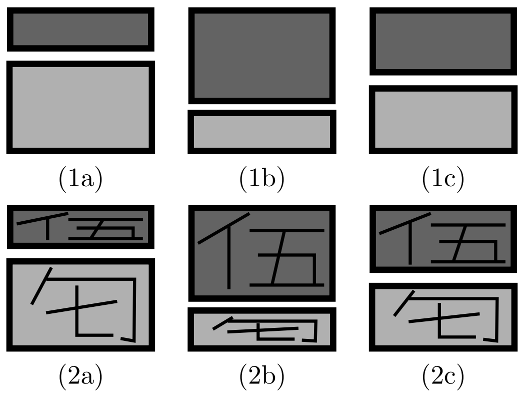


Figure 9: Spacing operands: first row shows generic ratios; second row shows possible characters; with (2c) being the best result.

row of figure 9 (2a,2b,2c) shows example character for different ratios. In this example, (2c) is the best output. In our experiment, the above rules to calculate the notation work well.

7 Future work

We can further fine-tune the Qin notation generator by studying previously printed qin-pu and the symbols used in them to do a better job estimating a more aesthetic ratio between our operands. After we generate the notation, the layout of the music piece is another interesting topic as well. To typeset an entire piece of music, we need a language for describing the layout of the piece. This could, for example, combine Western staff notation with the traditional Qin notation in order to provide necessary information about the tempo of a piece. We hope to continue this project in the future [1].

References

[1] Candy L.K. Yiu, Portland State University, Qin notation generator. `http://web.pdx.edu/~candy/qin`.

[2] Candy L.K. Yiu, Wai Wong. *Chinese Character Synthesis*. ACM PG Conference, Hong Kong, China. January 25, 2003.

[3] Candy L.K. Yiu, Wai Wong. *Chinese character synthesis using METAPOST*. In proceedings of TUG 2003, *TUGboat* **24**(1), 85–93 (2003). `http://tug.org/TUGboat/Articles/tb24-1/yiu.pdf`.

[4] Wai Wong, Candy L.K. Yiu, and Kelvin C.F. Ng. *Typesetting Rare Chinese Characters in LaTeX*. In proceedings of the 14th European TeX Conference (EuroTeX 2003), *TUGboat* **24**(3), 582–587 (2003). `http://tug.org/TUGboat/Articles/tb24-3/wong.pdf`.

[5] `http://en.wikipedia.org/wiki/Guqin`. September 2005.

[6] `http://www.cs.pdx.edu/~jrb/chin`. Yuguzhai Qinpu, 1855, Fujian, China. September 2005.

[7] John Hobby and Gu Guoan. *A Chinese metafont*. *TUGboat* **5**(2), pp. 119–136, 1984.

[8] John D. Hobby. *A user's manual for MetaPost*. AT&T Bell Laboratories Computer Science Technical Report, No. 162, 1992. `http://cm.bell-labs.com/who/hobby/pubs.html`.

[9] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986.

SwiExr: Spatial math exercises and worksheets, in Braille and print

Nandan Bagchee and Eitan M. Gurari
Ohio State University
gurari@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~gurari

Abstract

LaTeX is a highly expressive authoring language considered to be the lingua franca of the mathematics community. Yet, until recently, except for a few contributions concerning long division, it offered very little support for expressing planar layouts of problems that employ elementary mathematic operations.

We present a highly configurable tool [2] (written in Java) for producing spatial representations of elementary math exercises and worksheets. The available configurations produce verbatim and tabular forms of exercises and worksheets in print and Nemeth Braille formats for inclusion in LaTeX, MathML, HTML, and text files. In addition, they ensure Braille output entirely equivalent to material prepared in print formats. Our current attention is devoted to the addition, subtraction, multiplication, division, and root operations.

We are interested in identifying potential users from the LaTeX community with the objective of developing widely acceptable LaTeX interfaces for requesting math exercises and worksheets.

1 General background

The arithmetic tasks of addition, subtraction, multiplication, division, and square root assume a central place in elementary math education. Spatial arrangements of problems with such operations offer the foundation for applying computational procedures to instances which involve operands with large values. Students are expected to acquire fluency with the algorithms involved in dealing with the representations and understanding of the underlying ideas. To reach this end, they typically practice the approaches to the point of automaticity. The teachers are required to supply variants of the problems for the students to drill with.

Printed and written material, in original and xeroxed forms, as well as exercise program generators, are readily available to sighted students and their teachers. When it comes to math in Braille, outstanding detailed guidelines are available for creating worksheets of exercises [7, 8] but very few actual resources for blind students are provided. The guidelines are legal requirements for Braille to be produced according to the Braille Authority of North America standards [4].

`math worksheets`	2,110,000
`"math worksheets"`	276,000
`math worksheets Braille`	17,600
`"math worksheets" Braille`	223

Table 1: Number of entries listed for different searches with Google.

Figure 1 exhibits a sample of spatial layouts of problems provided in the guidelines. On the other hand, Table 1 shows some data obtained in searching the web for math worksheets. In the case of Braille, only one of the entries turned out to be relevant: it offered worksheets of arithmetic operations on numbers of single digits, and got listed in 1999 with a cost quote of $110 per 650 worksheets.

The problem of a lack of mathematical and scientific content in Braille, and the high cost of producing such material, are well known. For instance, the American Printing House for the Blind estimated that, due to a severe shortage of transcribers, only 78 out of the 3000 general textbooks published in 1999 were available in Braille in January 2000 [11]. Transcribing a single textbook can take more than six months [11], and may cost up to $9,500 [6].

A vast amount of scientific content is available

Editor's note: This material is based upon work supported by the National Science Foundation under Award No. IIS-0312487. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

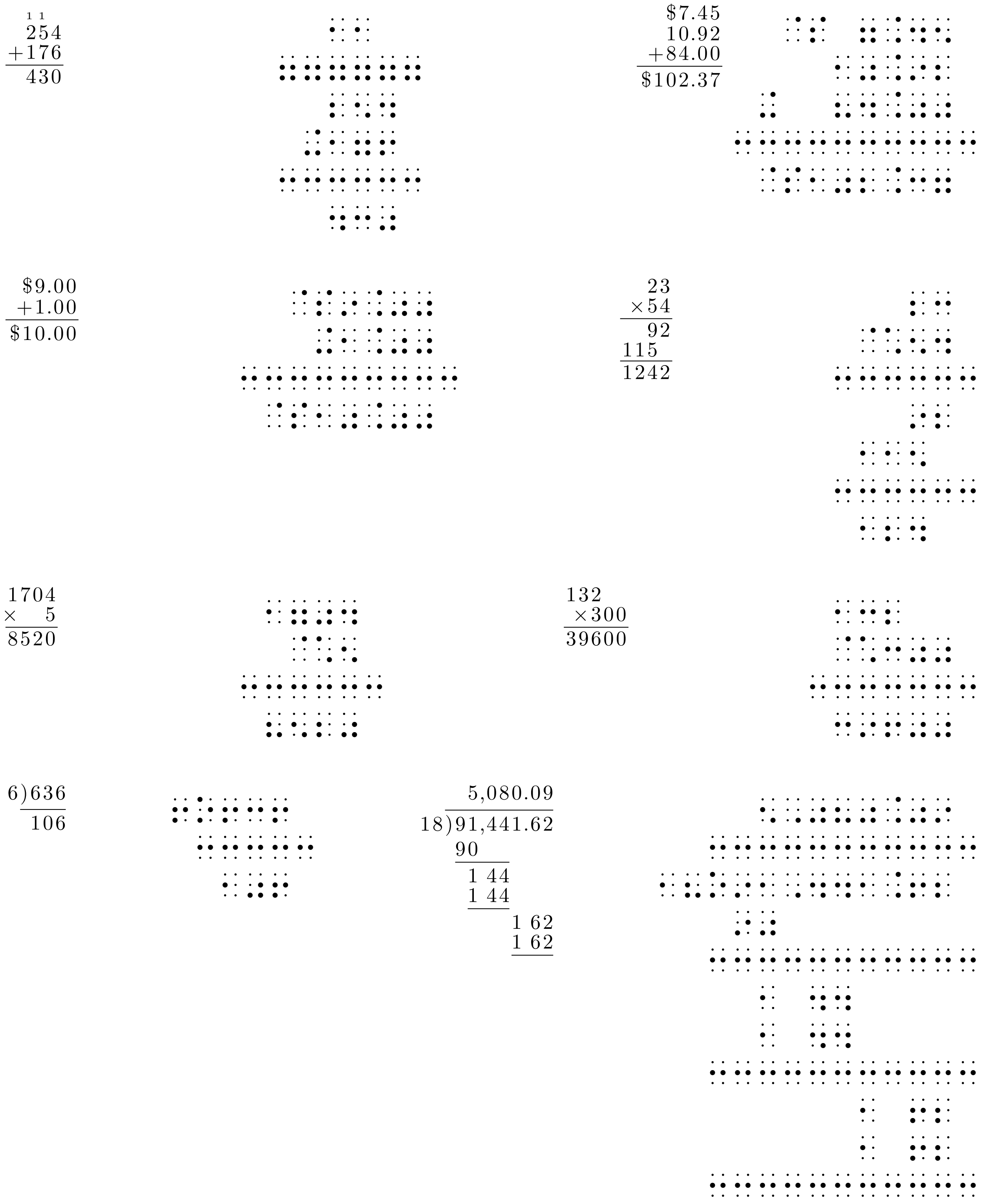
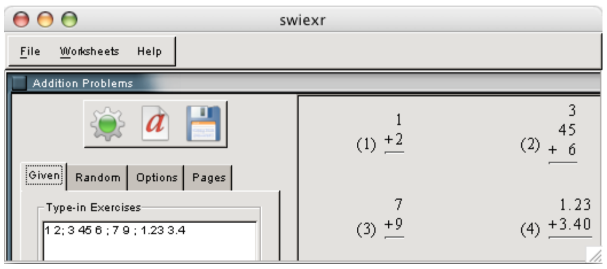


Figure 1: Spatial layout of math exercises in Braille.

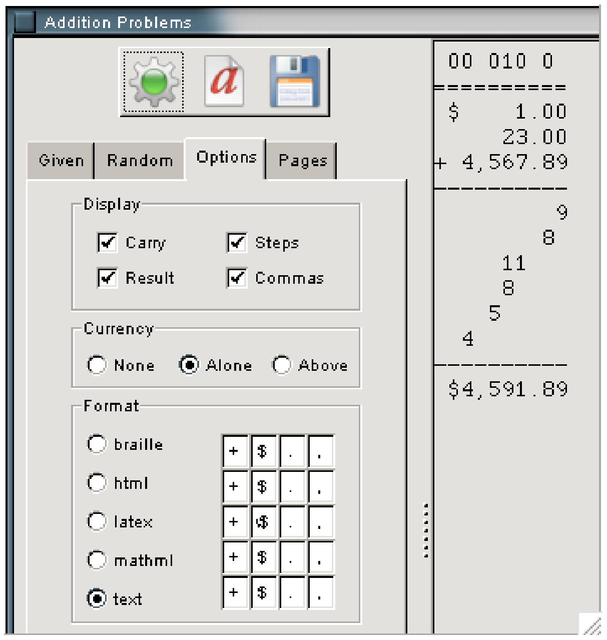


Type-in Exercises: 1 2; 3 45 6 ; 7 9 ; 1.23 3.4

```
     1                3
(1) +2               45
    --          (2) + 6
                    ---

     7             1.23
(3) +9        (4) +3.40
    --            -----
```

(a)



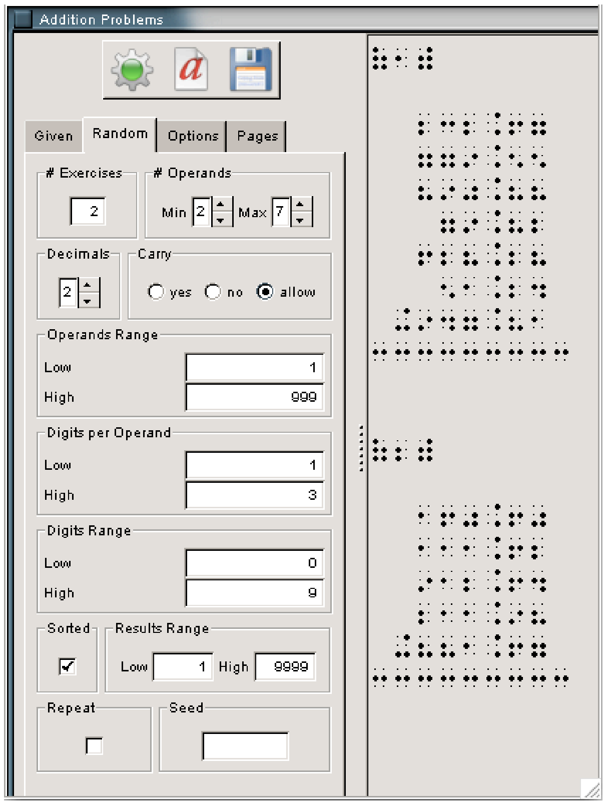
Display: [✓ Carry] [✓ Steps] [✓ Result] [✓ Commas]

Currency: None [✓ Alone] Above

Format: braille html latex mathml [✓ text]

```
00 010 0
==========
$     1.00
     23.00
+ 4,567.89
----------
         9
        8
      11
      8
     5
    4
----------
$4,591.89
```

(b)



Exercises: 2

Operands: Min 2 Max 7

Decimals: 2

Carry: yes no [✓ allow]

Operands Range: Low 1, High 999

Digits per Operand: Low 1, High 3

Digits Range: Low 0, High 9

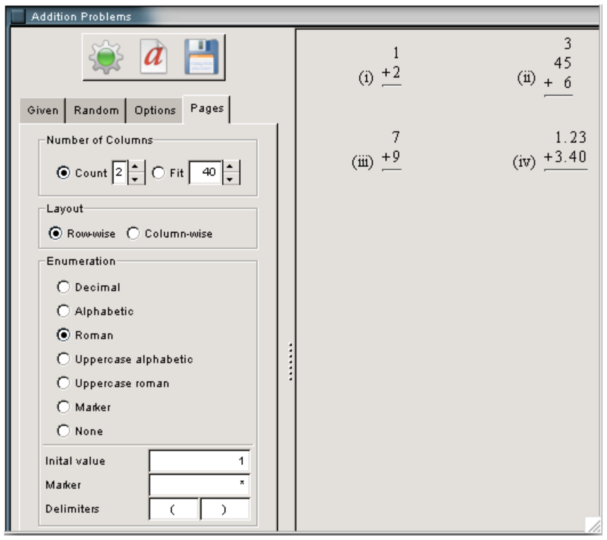
Sorted: ✓

Results Range: Low 1, High 9999

Repeat: ☐

Seed:

(c)



Number of Columns: [✓ Count 2] Fit 40

Layout: [✓ Row-wise] Column-wise

Enumeration: Decimal Alphabetic [✓ Roman] Uppercase alphabetic Uppercase roman Marker None

Inital value: 1

Marker: *

Delimiters: ( )

```
      1                 3
(i)  +2                45
     --          (ii) + 6
                      ---

      7              1.23
(iii) +9        (iv) +3.40
      --             -----
```

(d)

Figure 2: Graphical user interfaces for addition.

in LaTeX, authored either directly with text editors or indirectly through word processors that export LaTeX. LaTeX also seems to be the authoring language preferred by visually impaired scientists.

The objective of our work in general is to automate the production of technical and scientific material in Braille, with an emphasis on translating content available in LaTeX. Our current attention is devoted to the area of producing spatial arrangements for arithmetic exercises.

We introduce a highly configurable utility dedicated to this task, capable of producing worksheets in different forms for general users, while ensuring an option of Braille output for all supported features. We present a graphical user interface (GUI) of the tool, and exhibit a corresponding XML representation for the data to be fed into the tool. Then we consider LaTeX in the role of a user interface for the tool.

2 Graphical user interfaces

In order to serve a large base of users with different needs and preferences, our tool was designed in a modular manner which allows easy connection to independent standalone user interfaces. Figure 2 displays GUIs currently provided for the addition operation. GUIs of a similar nature are also offered for the other operations.

The GUI of Figure 2(a) lets the user explicitly introduce the problems to be processed. The GUI of Figure 2(b) provides the means to request the desirable content and representation for the problems. In particular, there are options to present the problems with their results, with intermediate steps, with the carries produced during the computations, and a currency sign. The addition sign may be placed in the same columns as the currency sign, or leftward. The output may be exported in LaTeX, MathML, text, or Braille format.

A user may request randomly generated problems satisfying desirable constraints via the GUI shown in Figure 2(c). The data to be included refers to the number of problems to be generated, the desirable number of decimal digits, lower and upper limits on the number of desirable operands, the total number of digits per operand, and the magnitudes of the digits. Figure 2(d) shows a GUI for determining the characteristics involved in typesetting the problems within the worksheets.

3 System architecture

Two aspects make the system highly configurable: a modular architecture which gives independent attention to the different functions of the system,

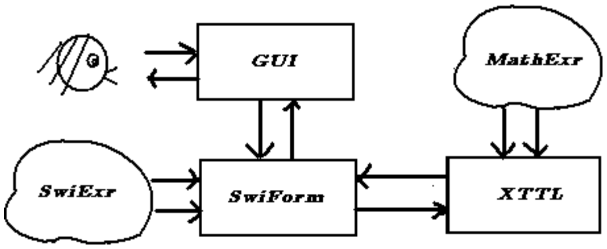


Figure 3: The underlying structure of the system.

and a script-based approach which provides for the functions to be described within external configuration files instead of being hardwired into the code. The system consists of four major components (Figure 3).

The back end of the system employs a script-based driver, called XTTL, for performing XML transformations. The driver is offered a library, called MathExr, of scripts for typesetting exercises and worksheets. In addition, the library includes a utility able to compute data for the exercises. The scripts for handling the different operations are independent of one another, and they can be supplemented and augmented by additional scripts to achieve alternative outcomes. The examples of Figure 1 were obtained with those scripts.

The front end of the system uses a script-based driver, called SwiForm, for managing the GUIs. The driver is built as an extension to the SwiXml system [10] and is offered a library, called SwiExr, of scripts that specify the desirable features for the GUIs. In addition, the library provides scripts and a utility for binding the front end to the back end, and for filtering the data communicated between these ends.

The MathExr scripts are currently tailored to receive the requests in XML format, and to deliver the exercises in Braille, HTML, LaTeX, MathML, and plain text formats. The type of the information being transmitted is fully dependent on MathExr, and that makes it possible to substitute a given graphical front end for another. A LaTeX front end is of interest for us here.

4 An XML view

The MathExr component takes as input a high level description of the desirable outcome and processes it into a detailed account of how the digits, symbols, and rulers of the exercises are to be placed on a grid. The input of MathExr in its XML forms is to a large degree a mirror of the visual format available from the GUIs.

```
<addition xttl="add-tex.xttl"
          carry="yes"
          steps="yes"
          result="yes"
          currency="alone">
  <operands>
    <operand>1</operand>
    <operand>23</operand>
  </operands>
</addition>
```

(a)

```
<multiplication steps="yes"
                result="yes"
                currency="alone"
                xttl="mult-txt.xttl">
  <operands>
    <operand>1234</operand>
    <operand>5</operand>
  </operands>
</multiplication>
```

(b)

```
<division steps="no"
          result="yes"
          remainder="yes"
          currency="yes"
          rulers="full"
          xttl="div-brl.xttl" >
  <operands>
    <operand>1234</operand>
    <operand>56</operand>
  </operands>
</division>
```

(c)

```
<generate xttl="worksheet-generate-tex"
          cols="3"
          enumerate="decimal"
          enumerateBefore="("
          enumerateAfter=")"
          enumerateStart="1">
  <exr type="addition"
       howmany="2"
       xttl="add-tex"
       carry="yes"
       steps="yes"
       result="yes"
       currency="alone">
    <minEntries>2</minEntries>
    <maxEntries>7</maxEntries>
    <low>100</low>
    <high>999</high>
    <minDigits>1</minDigits>
    <maxDigits>3</maxDigits>
    <decimals>3</decimals>
  </exr>
</generate>
```

(d)

Figure 4: XML requests for typesetting three types of problems and a worksheet.

Figure 4 contains examples of requests in XML for single exercises and for a worksheet of exercises. Each of the examples states the data required for deriving the outcome and the name of the script to be applied on the data. For instance, the first example asks for the script stored in a file named `add-tex.xttl`. That script typesets the addition problem onto a grid and exports the result in a tabular format acceptable to LaTeX.

5 A LaTeX perspective

The motivation to develop our tool was rooted in the desire to provide the means to prepare spatial arrangements of math exercises in Braille. The Braille exercises are to serve mainly blind students attending mainstream classes populated mostly by sighted children. To properly serve the blind students, it is crucial for the Braille exercises to accurately represent the problems given to the sighted students. Consequently, we programmed our tool to support standard print formats as well as Braille, with the hope of making it a useful tool for preparing worksheets in any format. In this way, we can ensure that the Braille material is entirely equivalent to the material in print formats.

LaTeX offers to our tool an obvious method of producing high quality printouts, through style files that are easy to tailor and modify. In addition, LaTeX also provides our tool with an option for a natural text-based user interface to request the exercises. A front end of this kind obviously enables authors to seamlessly incorporate exercises and worksheets into documents, and introduces a friendly authoring environment for those who shy away from WYSIWYG platforms. In particular, a non-WYSIWYG front end for accessing the tool is obviously crucial to blind users.

A LaTeX style file to support the input environment can consist of macros that imitate the high level structures captured by the XML code presented to MathExr. For instance, the following LaTeX-oriented command expresses the same request as the XML code of Figure 4(a).

```
\mathexr[op="addition"
         xttl="add"
         carry="yes"
         steps="yes"
         result="yes"
         currency="alone"]{1,23}
```

The implementation of the macros can be quite straightforward, relying on a process similar to that for bibliographies and indexes in LaTeX. Specifically, the arguments given to the macros may be written into a file in XML format, with the expectation that the file will be processed by MathExr. The exercises produced by MathExr can then be imported by the macros into the LaTeX source in consecutive compilations.

6 Connecting the dots

We introduced our tool to the public domain at the TUG 2005 International Typesetting Conference with the hope of promoting the tool with support from the LaTeX community. In particular, we looked for potential users and developers, interested in LaTeX interfaces and typesetting criteria for elementary math exercises. In addition, we wished to connect with potential developers for Braille and related proofreading fonts.

Despite the central role LaTeX plays in documenting mathematical and scientific content, until recently the only public support available for elementary math exercises was a single macro by Barbara Beeton and Donald Arseneau [3]. This macro uses input interfaces such as the macro invocation `\longdiv{12345}{13}` and produces output like that shown in Figure 5.

```
        949
   13 ) 12345
        11700
        -----
          645
          520
          ---
          125
          117
          ---
            8
```

Figure 5: A display by `\longdiv`.

The situation has changed recently, as a new LaTeX package for typesetting spatial math exercises, named Xlop, was submitted to CTAN [5]. In spirit, the new package answers our wish for a LaTeX counterpart to our tool. We hope future work on Xlop and our tool will enrich both utilities with additional features, and will bridge the differences between the two utilities to allow to support each other. In particular, we would like to see Xlop providing a LaTeX extension to our system, and have our utility offer alternative output formats to Xlop in general and to include Braille in particular.

We are not aware of any LaTeX fonts for Braille. A substitute through LaTeX pictures is available via a style file [9]. The issue of backward translating math exercises in Braille to LaTeX got some attention in an experimental system called INSIGHT [1].

7 Acknowledgment

We are very grateful to Susan Jolly for introducing us to Braille and for her generous guidance and help. Thanks also to Barbara Beeton and Karl Berry for editing the manuscript.

References

[1] N. Annamalai, D. Gopal, G. Gupta, A. Karshmer, and H. Guo, *INSIGHT: A comprehensive system for translating Braille-based mathematical documents to LaTeX*, in Proceedings of the International Conference on Human Computer Interaction, pp. 1245–1249. 2003. (Demos available at `http://www.logicalsoft.net/frame/demo.htm`.)

[2] N. Bagchee and E. Gurari, *SwiExr*, `http://www.cse.ohio-state.edu/~gurari/mathexr/`.

[3] B. Beeton, D. Arseneau, *Long division*, *TUGboat* 18(2), June 1997, p. 75–76, `http://tug.org/TUGboat/Articles/tb18-2/tb55works.pdf`.

[4] Braille Authority of North America (BANA), `http://www.brailleauthority.org/`.

[5] J. Charpentier, *Xlop*, 28 April 2005, `http://www.ctan.org/tex-archive/macros/generic/xlop`.

[6] Computers To Help People, Inc. (CHPI), Sponsoring Technical Reference Books and Manuals, `http://www.chpi.org/refspons.htm`.

[7] R. Craig, *Learning the Nemeth Braille Code: A Manual for Teachers and Students*, American Printing House for the Blind, 1987.

[8] A. Nemeth, *The Nemeth Code of Braille Mathematics and Science Notation.* The Braille Authority of North America (BANA), American Printing house for the

Blind, 1972 revision. (French version: `http://www.meq.gouv.qc.ca/ens-sup/ens-coll/Braille/Code_Braille_mathematique.pdf`)

[9] W. Park, *LaTeX 2ε package for typesetting Braille*, April 1999, `http://www.ctan.org/tex-archive/macros/latex/contrib/braille/braille.html`.

[10] W. Paulus, *SwiXml*, `http://www.swixml.org/`.

[11] *Texas Partnership for Increasing Braille Production*, Report of Braille Production Specialist Focus Group Meeting, January 2000, `http://www.tsbvi.edu/textbooks/afb/texas-transcriber.htm`.

Typesetting the Byzantine *Cappelli*

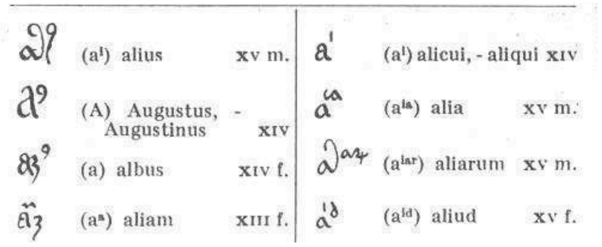
Philip Taylor
The Computer Centre, Royal Holloway,
University of London, TW20 0EX,
United Kingdom
`mailto:P.Taylor@Rhul.Ac.Uk`

Abstract

An overview of the author's rôle in the preparation of the forthcoming *Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands*, with particular emphasis on two challenges: sorting TEX markup for polytonic Greek using multiple concurrent keys, and deriving statistical data which could be used to provide input to the book design.

Introduction

One of the greatest pleasures that I get from my position as Webmaster at Royal Holloway, University of London, is the only-too-rare opportunity to work with truly gifted and dedicated scholars. For the last few years, I have been truly privileged to be able to work with Miss Julian Chrysostomides, Director of our Hellenic Institute, and with Dr Charalambos Dendrinos, a Research Fellow within the same Institute. These two extraordinary scholars have both devoted a considerable portion of their lives to the collection, collation and preparation of material for a *Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands* which is intended to do for Byzantine scholarship what Adriano Cappelli's *Dizionario di Abbreviature latine ed italiane* has been doing for Latin scholarship for the past 100 years.



	(a^{l}) alius	xv m.		(a^{l}) alicui, - aliqui	xiv
	(A) Augustus, - Augustinus	xiv		(a^{la}) alia	xv m.
	(a) albus	xiv f.		(a^{lar}) aliarum	xv m.
	(a^{a}) aliam	xiii f.		(a^{ld}) aliud	xv f.

Figure 1: A fragment from Cappelli's *Dizionario*

For both Latin and Byzantine scholars, the task of deciphering manuscripts which may be more than a thousand years old is not simply one of reading a long-dead scribe's handwriting: far more difficult is the task of identifying and correctly interpreting the various abbreviations, ligatures and other scribal shorthand notations that he or she may have used. Even a skilled palæographer may have difficulty in deciphering these, although for Latin scholars Cappelli's *Dizionario* provides an invaluable tool.

Only too aware of the difficulties that their students were experiencing in attempting to decipher Byzantine manuscripts, Julian & Charalambos decided to compile a Byzantine dictionary that would provide their students, and future scholars, with a key to those scribal notations which were most likely to cause problems in interpretation. For over five years, these two scholars have been painstakingly researching and deciphering hundreds if not thousands of individual manuscripts and recording the results of their work, initially using fairly primitive technology such as Windows 3.1's *Cardfile* and Eberhard Mattes' *emTEX* but more recently using spreadsheet technology (Microsoft's *Excel*) and the TEX Live Windows implementation of Hàn Thế Thành's *Pdf(LA)TEX* by Fabrice Popineau.

The Work of the Scholars

Although locating and obtaining copies of the manuscripts requires a not-insignificant amount of time, I will concentrate here on the tasks which the scholars undertake once the copies have been received. Each scribal notation that is potentially of interest is identified and scanned, and any artifacts that might serve to confuse are eliminated using a light pen and suitable software (JASC's *Paintshop PRO*). The resulting "clean" image is then stored as a PDF file using a fixed naming convention, and a corresponding entry made in an *Excel* spreadsheet: this entry contains the filename, the transcription, an explanation (if the notation is an abbreviation or similar) and the provenance (typically the date, but occasionally a more detailed provenance where this is felt to be important). Lest this create the impression that the rôle of the scholars is trivial, let me emphasise that the task of deciphering and interpreting the scribal notations is one requiring much skill and many many years of experience!

Scanning and transcription take place in batches, following which proofing takes place. During the proofing stage, the size and relative position of each scanned image is adjusted to ensure that all scanned images reproduce at approximately the same height and with the same vertical offset from the notional baseline. As will be seen later, ensuring that all images reproduce at the same height has only a limited effect on their *widths*: as a result, a statistical analysis of the widths of the scanned images will later be needed to allow an accurate assessment of the proportion of images that would fit without problem were a particular book design to be adopted. The scaling and offset parameters are stored within the *Excel* spreadsheet.

As far as is possible, syntax errors are identified and corrected at the proofing stage, although some may sneak through and require further intervention at the galley or page-proof stages. Because of the complexity of the markup used to represent transcriptions and explanations, such errors can easily creep in — these frequently involve braces, either as mismatched pairs or through the accidental use of non-brace symbols such as parentheses or brackets.

Markup and Syntax

Each entry in the spreadsheet consists of a number of fields, most of which are destined to become incorporated in a TEX document. Each record in the resulting TEX file conforms to the following pattern:

```
\Byzantine
    <scale>
        <y-offset>
            <filename>
                <transcription>
                    <explanation>
                        <provenance>
```

of which an example might read

```
\Byzantine
    -0.2
        -0.5
            abr-qwrafion2
                {{q\raise {f}}}
                    {{qwr'afion}}
                            {{1430}}
```

The `\Byzantine` command is used to introduce each record, and its meaning is redefined in various programs used to process the data. The first two parameters represent the scaling to be used (a negative value implies shrinkage rather than magnification) and the offset from the horizontal axis (a negative value implies lowering rather than raising). The third parameter is the filename, the portion preceding the hyphen indicating into which of about ten general classifications the record should be subsumed. The fourth parameter is the transcription, marked up according to Silvio Levy's encoding scheme for polytonic Greek with additional commands required to indicate scribal ornamentations such as `\raise {}`, `\overbar {}`, etc. The fifth parameter is the explanation, again marked up using Levy's scheme (this time with no extensions); and the sixth and final field is the provenance.

Sorting the Data

Although the data are coarsely pre-sorted by virtue of the prefix element of the filename, the actual lexicographic sorting needed before the data can be incorporated in the final lexicon is *considerably* more complex. Not only is it necessary to sort — by Greek collating rules — the latin transliteration of the Greek characters originally used, it is also necessary to ensure that the sorting takes into account all of the additional orthographic devices which may occur: breathings, accents, iota subscripts, ornaments (raised [groups of] letters, overbars), diareses and of course case-differences themselves. Furthermore, it is necessary to sort initially by transcription, but if — for a given record — the transliteration is absent, or identical to another transcription, then sorting must instead be by explanation, and if two or more records are found *still* to be identical after all of these criteria have been considered, then the date (provenance) and finally the original order in the spreadsheet must be taken into account (this last fallback key ensures that no manual sorting will ever be required once the data have been correctly entered in the spreadsheet).

Needless to say, sorting of this complexity is a task for which TEX is rather less than ideally suited. Even though earlier workers (e.g., Kees VAN DER LAAN, 1993; Bernd RAICHLE, 1994) have shewn that TEX is perfectly *capable* of performing sorting, the magnitude of the data (some 4000 records) and the complexity of the sorting required suggest that a more appropriate tool be used. The problem, however, is that the TEX markup used is fairly complex, and in order to parse it effectively, TEX itself is really required. Thus we appeared to be on the horns of a dilemma: on the one hand, TEX was felt to be unsuitable for the task of sorting, yet on the other TEX was considered to be essential if the markup were to be correctly parsed and interpreted. This *impasse* was finally resolved during discussions at a EuroTeX conference with Professor Klaus Lagally, who had experience of similar problems when trying to sort Arabic text in TeX: his solution, which proved to be absolutely ideal, was that we should treat the task as two separate problems — (1) parsing the TeX markup, and (2) sorting the data. The key to the solution lay in his suggestion that, during the sorting phase, TeX be asked to output far simpler keys (e.g., purely numeric) which could then be easily interpreted by any conventional sorting routine.

Parsing in TeX

With Klaus's suggestions firmly in mind, work started on writing the TeX parser. Although it would have been possible to write all keys to a single file, it was decided to associate each key with a unique file:

```
\immediate \openout  \TRAccents
    = Transcription.accents
    . . .
\immediate \openout  \TROrnaments
    = Explanation.ornaments

\immediate \openout  \EXAccents
    = Transcription.accents
    . . .
\immediate \openout  \EXOrnaments
    = Explanation.ornaments

\immediate \openout  \Dates
    = Byz-data.dates
\immediate \openout  \Sequence
    = Byz-data.sequence
```

Note that accents, breathings, cases, diareses, iotas, letters and ornaments are replicated for transcription and explanation but that dates and sequence numbers require only a single instance of each.

The main loop of the program iterates over its input file:

```
\loop
    \read \source to \buffer
    \ifeof \source
        \repeatfalse
    \else
        \expandafter
                \parse \buffer \endparse
        \advance \entry by 1
        \repeattrue
    \fi
\ifrepeat
\repeat
```

Before parsing the transcription and the explanation, we re-define the output files as being `\TR...` or `\EX...` as appropriate:

```
\def \Usetranscription
    {%
     \let \Accents = \TRAccents
      . . .
     \let \Ornaments = \TROrnaments
    }

\def \Useexplanation
    {%
     \let \Accents = \EXAccents
      . . .
     \let \Ornaments = \EXOrnaments
    }
```

Remembering that each input record consists of the control word `\Byzantine` followed by six parameters, we define `\parse` to call the analysis routine twice, passing first the transcription and then the explanation:

```
\def \parse \Byzantine
            #1 #2 #3-#4 #5#6#7\endparse
    {%
     \reset
     \transcription = {#5}
     \Usetranscription
     \analyse #5\endparse \endanalyse
     \print

     \reset
     \Useexplanation
     \explanation = {#6}
     \analyse #6\endparse \endanalyse
     \print
    }
```

The apparent difference between the earlier statement that the transcription forms the fourth parameter (and the explanation the fifth parameter) and the code above, which appears to refer to them as the fifth and sixth parameters respectively, is explained by the fact that during parsing we treat the filename as two separate parameters separated by a hyphen: this allows the prefix (representing the general category into which the entry fits) to be extracted and used to qualify the date, since the lexicon is macro-ordered by general category, and micro-ordered by the sorting criteria currently being described.

The analysis code itself is fairly straightforward:

```
\def \analyse #1#2\endanalyse
    {\ifx #1\endparse
     \else
```

```
        \def \flag {#2}
        \advance \index by 1
        \process {#1}
        \analyse #2\endanalyse
     \fi
    }
```

all of the complexity being delegated to the `\process {}` routine :

```
\def \process #1%
    {
        \csname +\string #1\endcsname
    }
```

Before `\process {}` can be understood, it is first necessary to explain how the parser identifies into which category each token fits. The program starts by listing the various categories :

```
\newentity {accent}
\newentity {breathing}
\newentity {diaresis}
\newentity {grouping}
\newentity {iota}
\newentity {letter}
\newentity {ornamentation}
```

Then, for each category, the tokens which compose that category are enumerated. Here, for example, the possible accents are enumerated :

```
\newaccent ‘
\newaccent ’
\newaccent ˜
```

To avoid the risk of human error, we interrogate an internal counter to find how many elements there are in each category :

```
\numberof \accents = \valueof ˜
\advance \accents by 1 %%% null accent
```

Now take a deep breath, because we need to discuss how `\newentity {}` is defined :

```
 1 \def \newentity #1
 2   {\expandafter \NewCount
 3                     \csname #1\endcsname
 4    \expandafter \def \csname
 5                     new#1\endcsname ##1%
 6     {\advance \csname #1\endcsname by 1
 7      \expandafter \edef \csname
 8                    +\string ##1\endcsname
 9        {\expandafter \noexpand
10           \csname #1token\endcsname
11            {\string ##1}
12            {\the \csname #1\endcsname}%
14        }
15     }
16   }
```

As this code is somewhat opaque, let’s make life simpler by considering what happens when `\newentity {}` is called with a parameter, as in `\newentity {accent}`. Lines 2 to 3 expand to yield `\NewCount \accent`. `\NewCount` can be used in macro expansions but is otherwise identical to Plain’s `\newcount`. Lines 4 onwards expand to yield a definition for the single-parameter macro `\newaccent {}`; the definition is equivalent to the following pseudo-TEX code :

```
\def \newaccent #1
    {\advance \accent by 1
     \edef \+#1%
        {\accenttoken {#1}{\the \accent}
    }
```

Again it will be simpler to understand through the medium of an example, so we will consider what happens when `\newaccent {}` is called with parameter `˜`, as in `\newaccent ˜`. The counter `\accent` is incremented by one (it starts life at zero), and the control sequence `\+˜` is defined to expand to `\accenttoken {˜}{<current value of \accent>}`. The sole function of the `+` prefix used in constructing the name of the control sequence is to reduce the risk of a namespace clash.

We will next need to look at `\accenttoken {}`, which as we see below is just one of a family of identically treated control sequences :

```
\def \accenttoken #1#2%
                {\do {Accent}{#1}{#2}}
\def \breathingtoken #1#2%
                {\do {Breathing}{#1}{#2}}
\def \diaresistoken #1#2%
                {\do {Diaresis}{#1}{#2}}
\def \groupingtoken #1#2%
                {\do {Grouping}{#1}{#2}}
\def \iotatoken #1#2%
                {\do {Iota}{#1}{#2}}
\def \lettertoken #1#2%
                {\do {Letter}{#1}{#2}}
\def \ornamentationtoken #1#2%
                {\do {Ornament}{#1}{#2}}
```

after which we need to examine `\do {}` :

```
\def \do #1#2#3%
    {%
     \csname #1\endcsname {#2} {#3}
    }
```

Since parameters 1 & 2 of `\accenttoken {}` become parameters 2 & 3 of `\do {}`, we can see that when `\do {}` is launched from `\accenttoken {}` the expansion is :

```
\Accent {<accent-token>}{<numeric-value>}
```

Remembering that TeX is case-sensitive, it should be clear that `\accent` and `\Accent {}` are totally different entities — the former is an integer register (declared with `\NewCount {}`), whilst the latter is explained below:

```
\def \Accent #1#2%
   {
   \lastaccent = #2
   }
```

Thus the sole effect of `\Accent {}` is to store the numeric value associated with the accent (its ordinal) in `\lastaccent`; most of the entities analogous to `\Accent {}` behave in a similar way, with the key exception of `\Letter {}` at which we must next look:

```
 1 \def \Letter #1#2%
 2     {%
 3      \lettervalue = #2
 4      \ifodd \lettervalue
 5         \edef \Caseskey {\Caseskey 1}
 6         \advance \lettervalue by 1
 7      \else
 8         \edef \Caseskey {\Caseskey 0}
 9      \fi
10      \edef \Accentskey
11         {\Accentskey \the \lastaccent}
12      \edef \Breathingskey
13        {\Breathingskey
14                 \the \lastbreathing}
15      \edef \Diareseskey
16        {\Diareseskey
17                 \the \lastdiaresis}
18      \edef \Iotaskey
19         {\Iotaskey 0}
20      \edef \Letterskey
21         {\Letterskey \expandafter
22             \expandafter \expandafter
23                 \twodigits \expandafter
24                     0\the \lettervalue
25                         \sentinel
26         }
27      \edef \Ornamentskey
28         {\Ornamentskey
29                  \the \lastornament}
30      \lastaccent = 0
31      \lastbreathing = 0
32      \lastdiaresis = 0
33     }
```

It is, in fact `\Letter {}` (which is triggered by the parser detecting a letter, as opposed to any diacritic or other orthographic mark) that is at the heart of the TeX parser under discussion. Remember that `\Letter {}` will be called with the actual letter as parameter 1 and the ordinal of that letter as parameter 2. At line 3, the ordinal is saved in `\lettervalue`. At lines 4 to 9, a test is made to see whether this is odd or even, a simple test which discriminates between upper- and lower-case letters. If the result is odd (uppercase), `\lettervalue` is rounded upwards to renormalise it as lowercase after noting the fact that it was originally uppercase. Note carefully the `\edef`s at lines 5 & 9, which append a zero or a one to the current value of `\Caseskey`: this same mechanism is used from lines 10 to 29 to append the last (accent, breathing, diaresis, or ornament) ordinal to the corresponding key.

Here at last we begin to see the results of all of our efforts: the various keys are extended (by a fixed amount) each time a letter is encountered in the input record to capture, as a set of sequences of fixed-length integers, the possible features which may be used to differentiate each letter from an otherwise identical letter when sorting finally takes place.

After parsing the transcription, and again after parsing the explanation, we write each of the seven keys to the associated file:

```
\def \print
    {
     \immediate \write \Accents
        {\Accentskey}
         ...
     \immediate \write \Ornaments
        {\Ornamentskey}
    }
```

After the keys for the transcription and explanation have been written to file, the date (with filename prefix prepended) and sequence number are similarly recorded:

```
\immediate \write \Dates {#3-#7}
\immediate \write \Sequence {\the \entry}
```

The Results

The end result of all of this is a series of files, each of which consist of n records, where n is the number of records in the original data set. Each record in each file will be of length $K \times l$, where l is the number of letters in the corresponding input record and K is a constant which varies from file to file (some keys can be represented as a single digit per character, some require two digits per character, and so on). A short fragment of a typical input file, and the corresponding extracts from sample key files, are shewn on the following pages; the samples are intended to illustrate most of the scribal devices used.

byz-data.dat

```
\Byzantine -0.5 0.4 abr-adelfou
    {{>ad\raise {e}}} {{>adelfo~u}}
        {{Thebes}}
\Byzantine -0.4 0.4 abr-adelfous
    {{>ad\raise {o‘us}}} {{>adelfo‘us}}
        {{1374}}
\Byzantine -0.3 0 abr-adelfwn
    {{>adelf}} {{>adelf~wn}}
        {{1374}}
\Byzantine -0.3 -0.2 abr-aer
    {{a\raise {e}r}} {{>’aer}}
        {{15\th c.}}
\Byzantine -0.2 -0.2 abr-aer1
    {{a\raise {e}r}} {{>’aer}}
        {{1492}}
\Byzantine -0.3 -0.5 abr-afierwthrion
    {{>af"I\raise {e}rw\raise {tr}}}
        {{>afierwt’hrion}} {{1420}}
\Byzantine -0.3 -0.2
    abr-afrodith-fwsforos-qalkos {{}}
        {{>Afrod’ith fwsf’oros / qalk‘os}}
            {{16\th c.}}
```

transcription.letters

```
041012
0410324638
0410122448
041236
041236
0448201236544436
<blank>
```

transcription.accents

```
000
00010
00000
000
000
00000000
<blank>
```

transcription.breathings

```
100
10000
10000
000
000
10000000
<blank>
```

transcription.cases

```
000
00000
00000
000
000
00100000
<blank>
```

transcription.diareses

```
000
00000
00000
000
000
00100000
<blank>
```

transcription.ornaments

```
001
00111
00000
010
010
00010011
<blank>
```

explanation.letters

```
04101224483246
0410122448324638
04101224485428
041236
041236
0448201236544441636203228
04483632102044164854384832363238500042422...
```

explanation.accents

```
0000003
00000010
0000030
200
200
000000020000
000002000000200000010
```

explanation.breathings

```
1000000
10000000
1000000
100
100
100000000000
100000000000000000000
```

explanation.cases

```
0000000
00000000
0000000
000
```

```
000
000000000000
1000000000000000000000
```

explanation.diareses

```
0000000
00000000
0000000
000
000
000000000000
0000000000000000000000
```

explanation.ornaments

```
0000000
00000000
0000000
000
000
000000000000
0000000000000000000000
```

Sorting in Perl

After the complexities of the TEX coding required to implement the parser, the matching Perl code will come as something of a relief! We start by opening or creating a few files:

```
$Data = "Byz-Data.dat" ;
open Data or die
    "File $Data cannot be opened :
        $ !\n" ;
@data = <Data> ;

$TRAccents = "Transcription.Accents" ;
open TRAccents or die
    "File $TRAccents cannot be opened :
        $ !\n" ;
@TRaccents = <TRAccents> ;

. . .

$EXOrnaments = "Explanation.Ornaments" ;
open EXOrnaments or die
    "File $EXOrnaments cannot be opened :
        $ !\n" ;
@EXornaments = <EXOrnaments> ;

$Dates = "Byz-Data.dates" ;
open Dates or die
    "File $Dates cannot be opened :
        $ !\n" ;
@dates = <Dates> ;

$Sequence = "Byz-Data.Sequence" ;
open Sequence or die
    "File $Sequence cannot be opened :
        $ !\n" ;
@sequence = <Sequence> ;

$Sink = ">Byz-Data.Srt" ;
open Sink or die
    "File $Sink cannot be created :
        $ !\n" ;
```

We then enumerate the keys that will be used for sorting:

```
@key11 = @TRletters ;
@key12 = @TRbreathings ;
@key13 = @TRaccents ;
@key14 = @TRiotas ;
@key15 = @TRornaments ;
@key16 = @TRdiareses ;
@key17 = @TRcases ;

@key21 = @EXletters ;
@key22 = @EXbreathings ;
@key23 = @EXaccents ;
@key24 = @EXiotas ;
@key25 = @EXornaments ;
@key26 = @EXdiareses ;
@key27 = @EXcases ;

@key31 = @dates ;
```

Then we perform a detached key sort and output the results:

```
@keys = sort polytonically @sequence ;
foreach $key (@keys)
            {print Sink $data [$key]} ;
```

All that remains is to define the comparison algorithm:

```
sub polytonically
    {if     (($key11 [$a]
                cmp $key11 [$b]) != 0)
                    {return $key11 [$a]
                        cmp $key11 [$b]}
     elsif  (($key12 [$a]
                cmp $key12 [$b]) != 0)
                    {return $key12 [$a]
                        cmp $key12 [$b]}
     . . .

     elsif  (($key27 [$a]
                cmp $key27 [$b]) != 0)
                    {return $key27 [$a]
                        cmp $key27 [$b]}
     elsif  (($key31 [$a]
                cmp $key31 [$b]) != 0)
                    {return $key31 [$a]
```

```
                     cmp $key31 [$b]}
  else   {print Errors "Warning :
            duplicate entry : \n",
              $sequence [$a]+1,
                " : ", $data [$a],
                  $sequence [$b]+1,
                     " : ", $data
                         [$b], "\n"}
}
```

Statistical Analysis of Field Widths

As explained above, although the scanned images are normalised for height and position, it is impossible to normalise them for width since some are inherently narrow and others are inherently wide. The transcriptions, explanations and provenances, too, vary widely in length. In order to gain an insight into the best distribution of the available space between the various fields (and, indeed, in order to determine the minimum page size which would accommodate the longest possible entry for one-, two- and three-column designs), it was decided to perform a statistical analysis of the variation in minimum width of each of the fields. For the scanned images, all that was necessary was to record the width of each (after scaling) and to use *Excel* to analyse these (the actual analysis techniques used are discussed below), but for the textual fields there was an additional and very interesting problem : how does one decide what is the minimum width that can be used to typeset a given stretch of text ?

Obtaining the Statistics for Text Fields

One possible approach is to typeset the text in a `\vbox {}` with `\hsize = 0pt`. This will forcibly hyphenate every possible word, but the problem is knowing how to access the results : if one examines the dimensions of the `\vbox {}` after this operation, it will have finite height and depth, but the width will still be `0pt`, and even if one unboxes it and reboxes it, one finds that the internal `\hbox {}`es that TEX constructs whilst paragraph building also have zero width. However, all is not lost : if one uses TEX's box destructor primitive `\lastbox`, one can gain access to the last line of the paragraph ; unboxing and reboxing this line yields an `\hbox {}` with finite width as well as finite height and depth. Applying this technique iteratively allows access to (the widths of) all the lines of the paragraph, and if one then selects the largest of these, one then knows the narrowest measure within which the stretch of text could be typeset.

In reality, of course, this may be far too narrow to be usable, but once one has established a lower bound one is as least on the way to determining the optimal width.

Sample code which can be used to perform this operation is shewn below :

```
\toks 0 = {Research by the School of
           Management is not confined
           to for-profit corporations,
           it is also a leading centre
           for research into public
           sector organisations.  In
           recognition of the rising
           impact of sustainability for
           business and society, the
           School, together with the
           Department of Geography,
           have created an
           inter-disciplinary Centre for
           Research into Sustainability
           (CRIS).
           }

\newif \ifloop
\newdimen \minwidth

\def \findminwidth #1#2%
    {%
     \minwidth = 0 pt
     \setbox 0 =
     \vtop \bgroup
           \hsize = 0 pt
           \hfuzz = \maxdimen
           #1 \noindent #2 \par
           \loop
               \setbox 2 = \lastbox
               \ifvoid 2
                   \loopfalse
               \else
                   \setbox 4 = \hbox
                           {\unhbox 2}
                   \ifdim \wd 4 > \minwidth
                       \global \minwidth
                                    = \wd 4
                   \fi
                   \unskip
                   \unpenalty
                   \looptrue
               \fi
           \ifloop
           \repeat
           \egroup
     \message {min-width : \the \minwidth}
     \global \setbox 0 = \vtop
             \bgroup
             \hsize = \minwidth
             \rightskip = 0 pt
                plus \minwidth
             #1 \noindent #2 \par
             \egroup
    }

\findminwidth {\uchyph = 1}{\the \toks 0}
\setbox 2 = \box 0
\findminwidth {\uchyph = 0}{\the \toks 0}
\setbox 4 = \box 0
\findminwidth {\hyphenchar \font = -1 }
                              {\the \toks 0}
\setbox 6 = \box 0
\leftline
       {\box 2 \quad \box 4 \quad \box 6}
\end
```

The test lines at the end demonstrate that different minimum widths can be achieved depending on whether or not hyphenation is permitted, and if permitted, whether upper-case words may legitimately be hyphenated. For the sample stretch of text used, the three minima were `50.36124pt`, `60.05573pt` and `74.00015pt` for the uc & lc case, the lc-only case, and the no-hyph case respectively. Figure 2 shews the results of typesetting the sample text to these three measures:

Research	Research by	Research by
by the	the School of	the School of
School of	Management	Management
Manage-	is not	is not confined
ment is not	confined to	to for-profit
confined to	for-profit	corporations,
for-profit	corporations,	it is also a
corpora-	it is also	leading centre
tions, it	a leading	for research into
is also a	centre for	public sector
leading	research into	organisations. In
centre for	public sector	recognition of
research	organisa-	the rising impact
into public	tions. In	of sustainability
sector	recognition	for business
organisa-	of the rising	and society,

Figure 2: Typesetting to the narrowest measure: uc & lc hyphenation, lc-only, and no hyphenation

Analysing the Statistics

For many years, I eschewed spreadsheets completely, believing that they were yet another manifestation of "The Emperor's New Clothes" and offering nothing whilst promising everything ... It was only when I started chairing the TUG Bursary Committee that I began to realise that spreadsheets did indeed have something to offer, and so when I began to search for a tool to help with the analysis of field widths for the current project I started by looking at the possibilities of *Excel*.

At first I was defeated by little things: *Excel*, for example, seems unfamiliar with the concept of the point as a unit of measure, so it was unable to deal with TeX's `58.88902pt` notation. This was very easily dealt with once I realised that *Excel*'s `Data/Text to Columns.../Delimited/Other/"p"` would do exactly what I needed — strip off the trailing `pt` leaving only the unitless number in the source column.

The second task was considerably more difficult: given a set of some 4000 real numbers (representing the widths of one of the fields in the Lexicon), I wanted to (a) sort them, and (b) derive statistics which would shew what percentage were less than each unique value. I was convinced that *Excel* could manage this, but none of my *Excel*-literate colleagues (including my wife!) could tell me how to persuade *Excel* to do the necessary.

In the end, a Google search led me to the solution: the necessary statistical tools are *not* installed by default, and it is first necessary to install the appropriate options pack. `Tools/Add-Ins.../Analysis ToolPak` proved to be the required incantation, after which `Tools/Data Analysis.../Histogram/Cumulative Percentage` provided the exact statistics that I needed. Figures 3–4 shew a sample of the output from *Excel* covering the range from 85% to 98,5%. Access to statistics such as these for each of the four fields of the Lexicon will prove invaluable when putting the finishing touches to the book design, since the authors will be able to see for themselves what fraction of the entries would fit without compromise were a particular design to be selected.

Conclusions

TeX is a superb program, capable of producing the finest quality typeset output; however, it is neither the ideal tool for sorting, nor for producing statistical analyses. When used in conjunction with other tools such as Perl (sorting) and *Excel* (statistics), the combined power far exceeds the sum of the parts. Synergies such as these are surely the key to the rôle of TeX in the future: TeX should no longer be perceived as a tool in isolation, but rather as a partner in a whole suite of tools, each perfectly adapted to the task for which it is used.

43.51428	31	83.30%
44.06853	92	85.42%
44.62279	16	85.79%
45.17705	18	86.21%
45.73131	10	86.44%
46.28556	25	87.02%
46.83982	19	87.46%
47.39408	22	87.96%
47.94834	18	88.38%
48.50260	24	88.94%
49.05685	20	89.40%
49.61111	19	89.84%
50.16537	15	90.18%
50.71963	8	90.37%
51.27389	13	90.67%
51.82814	34	91.45%
52.38240	13	91.75%
52.93666	14	92.08%
53.49092	12	92.35%
54.04517	19	92.79%
54.59943	33	93.56%
55.15369	8	93.74%
55.70795	9	93.95%
56.26221	12	94.22%
56.81646	6	94.36%

Figure 3: Width, frequency & cumulative %age

57.37072	8	94.55%
57.92498	15	94.89%
58.47924	7	95.06%
59.03349	8	95.24%
59.58775	1	95.26%
60.14201	7	95.43%
60.69627	34	96.21%
61.25053	4	96.30%
61.80478	3	96.37%
62.35904	4	96.47%
62.91330	6	96.60%
63.46756	7	96.77%
64.02181	12	97.04%
64.57607	2	97.09%
65.13033	4	97.18%
65.68459	4	97.27%
66.23885	21	97.76%
66.79310	4	97.85%
67.34736	3	97.92%
67.90162	7	98.08%
68.45588	4	98.18%
69.01014	8	98.36%
69.56439	3	98.43%
70.11865	3	98.50%
70.67291	1	98.52%

Figure 4: Width, frequency & cum. %age (cont)

Acknowledgements

The author would like to acknowledge the unwavering and unstinting affection, dedication and patience of Miss Julian CHRYSOSTOMIDES and Dr Charalambos DENDRINOS, with whom it is truly a joy to work. He would also like to thank Professor Klaus LAGALLY for his advice on sorting non-Latin scripts, and Claudio BECCARI for his courtesy and patience whilst the author was trying to effect the transition from Silvio LEVY's polytonic Greek fonts to the "CB" series. Finally he would like to thank the TUG Bursary Committee for financial support which enabled him to present this paper at Wu Han (China) during the TUG 2005 conference.

Bibliography

VAN DER LAAN, C.G. ("Kees") 1993: *Sorting in BLUe*; Minutes and Appendices (MAPS) 10, Nederlandstalige TEX Gebruikersgroep (NTG).

RAICHLE, Bernd 1994: *Sorting in TEX's Mouth*; Proceedings of the 1994 EuroTEX Conference.

LuaTeX: Howling to the moon

Hans Hagen
Pragma ADE, The Netherlands
pragma@wxs.nl

Abstract

Occasionally we reach the boundaries of TeX and programming then becomes rather cumbersome. This is partly due to the limitations of the typesetting engine, but more important is that a macro language is not always best suited for the task at hand.

1 Some observations

Occasionally we reach the boundaries of TeX and programming then becomes rather cumbersome. This is partly due to the limitations of the typesetting engine, but more important is that a macro language is not always best suited for the task at hand. One problem is for instance that intermediate operations and final results are intermixed (which has a certain charm as well).

```
\def\RepeatMe#1#2%
  {\bgroup
   \count0=#1%
   \loop
     \ifnum\count0>0
       #2%
       \advance\count0 -1
   \repeat
   \egroup}
```

The next call gives *****:

```
\RepeatMe{5}{*}
```

If you want to use this macro to construct a new one holding 5 stars, you're tempted to do something:

```
\edef\ShowMe{\RepeatMe{3}{*}}
```

This will give an error and the reason for this is that `\RepeatMe` is (in TeX-speak) not fully expandable.

In ε-TeX we can define this macro in a fully expandable way:

```
\def\RepeatMe#1#2%
  {\ifnum#1>0 #2\else
     \expandafter\IgnoreMe
   \fi
   \expandafter\RepeatMe\expandafter
   {\the\numexpr#1-1\relax}{#2}}
\def\IgnoreMe#1#2#3#4#5{}
```

Now we can safely say:

```
\edef\ShowMe{\RepeatMe{3}{*}}
```

Or even:

```
\edef\ShowMe{\RepeatMe{3}{\RepeatMe{3}{*}}}
```

Unfortunately, many users will turn away from writing macros as soon as something like `\expandafter` shows up, so this solution is not of much help for novice users, given that they already have noticed that they need to do this kind of expansion in order not to end up in too deep nesting and in order to finish the condition. Also, we only have to add a simple counter that keeps track of the total number to end up with a non-fully expandable macro again.

```
\newcount\NOfRepeats

\def\RepeatMe#1#2%
  {\ifnum#1>0
     \advance\NOfRepeats 1
     #2%
   \else
     \expandafter\IgnoreMe
   \fi
   \expandafter\RepeatMe\expandafter
   {\the\numexpr#1-1\relax}{#2}}
\def\IgnoreMe#1#2#3#4#5{}
```

If we try:

```
\edef\ShowMe{\RepeatMe{2}{*}}
```

We get a macro `\ShowMe` with the meaning:

```
   \advance \NOfRepeats 1 *\advance \NOfRepeats
1 *
```

But there is a way out of this! See the following variant:

```
\def\RepeatMe#1#2%
  {\lua
     {for i=1,#1 do
        tex.print("#2")
      end}}
```

Isn't this much more convenient? It even looks readable. Before I explain the `\lua` command, I will make a few more observations.

As said, TEX programming can be cumbersome, but this is no reason to throw away the macro language. There is much macro code around and there are many documents that depend on TEX's longevity. So in order to overcome limitations, we need to talk in terms of extensions, not replacements.

Nowadays TEX is used for more than typesetting documents directly from a source. While manipulating the input, some applications demand more advanced programming features than good old TEX can (conveniently) provide. In many cases, it's not so much the whole machinery that we want to replace. We just want to extend or replace some of the subsystems. In order to provide robust solutions we may also want to have access to the internals. We'd like to add extensions and delegate activities to other programs. Of course there may also be personal motives for extending TEX, for instance because every now and then we need a new challenge.

2 Why Lua

I use SCITE as my main editor for documents and programs. This editor is built on top of the SCINTILLA framework. It's a fast, lightweight but powerful program. Being mainly a program editor it lacks some features that you need for text editing and processing, like adjusting text and spell checking. However ... it has a scripting engine that gives access to the user interface and the editing component. This scripting language happens to be LUA.

For a while I was hesitant to use 'yet another programming language', but using the built in LUA machinery made more sense than programming in C++ and trying to keep that up to date with the rest of the program. I was more or less familiar with the SCINTILLA programming interface because I had written the TEX and METAPOST lexers, but in contrast to the lexer code other extensions would involve changes spread more widely over the code base. So, the LUA way we went.

While the PERL, PYTHON, and (my favourite) RUBY languages are heavyweight languages, LUA is a simple but nevertheless powerful language. The first three come with a constantly growing number of libraries and users expect the whole lot to be present, which makes them unsuitable as extension languages: cross platform issues, slow startup, much bigger TEX distributions, etc.

LUA can interface to libraries but its main purpose is to be embedded in an application and present the user with an interface to the parent application. It's made for embedding! It has a small footprint and adds only some 50–100K to the parent binary. One can add additional functionality, for instance support for sockets.

The more I looked into it, the more I liked it, and after positive experiences with integrating METAPOST in ConTEXt, integrating LUA didn't seem that strange to me.

Not being a 'TEX the Program' hacker, I managed to trick Hartmut Henkel (member of the PDFTEX development team) into implementing a `\lua` command. Hartmut and I share a passion for experiments in PDFTEX and he knows how to implement them: I got back a (LINUX) binary within a few days.

After we had this proof of concept, it was time to involve Taco Hoekwater (member of the ConTEXt, METAPOST and other development teams), who knows more about TEX's internals than I ever will. Another few days later we had the first interface to some of TEX's internals.

3 A few examples

I will show a few of the examples that I demonstrated at the TUG 2005 conference in Wuhan, China. Keep in mind that an experimental version was used and that the interfaces may change. For instance, it will be possible to activate several LUA engines (`\newlua`) and all variables, the hash table, internal lists, etc., are on the agenda.

For starters, if we say:

```
\lua{a = 1.5 ; b = 1.8 ; c = a*b}
```

we only set some variables, and get nothing in return. However, the next call puts the value of `c` back into the input stream.

```
\lua{a = 1.5 ; b = 1.8 ; c = a*b ;
     tex.print(c)}
```

The result comes back as a string. This means that we need to process it in the right way in order get something that TeX sees as TeX code. For this reason we define:

```
\def\luatex#1{\scantokens\expandafter
              {\lua{#1}}}
```

The `\scantokens` primitive has some undesired side effects: it cannot properly handle multiple lines (catcode problem) and acts as a pseudo-file, but Taco is working on a variant.

So, in the next example, we get different results:

```
\lua   {tex.print("$\string\\sqrt{2} = "
                  .. math.sqrt(2) .. "$")}
\luatex{tex.print("$\string\\sqrt{2} = "
                  .. math.sqrt(2) .. "$")}
```

The `\lua` call returns a verbatim copy, i.e. a dollar shows up as a dollar, while the `\luatex` call gives a typeset formula, i.e. the dollar sign is seen as the signal to enter math mode.

Next we demonstrate how to use LUA to match strings (using regular expressions):

```
\lua {
  match = {} ;
  match.expression = "" ;
  match.string = "" ;
  match.result = {} ;
  match.start = 0 ;
  match.length = 0 ;
}
\def\matchstring#1#2{\lua {
  match.expression = "#1" ;
  match.string = "#2" ;
  match.result = {} ;
  match.start,match.length,match.result[1],
   match.result[2], match.result[3],
   match.result[4], match.result[5],
   match.result[6], match.result[7],
   match.result[8], match.result[9]
    = string.find(match.string,
                  match.expression);
}}

\def\matchresult#1{\lua {
    tex.print(match.result[#1]) ;
}}
```

When we use the last macro as:

```
\matchstring
  {(\letterpercent d+) (\letterpercent d+)
   (\letterpercent d+)}
  {2005 08 08}
```

the three components are available in:

```
\matchresult{1} \matchresult{2} \matchresult{3}
```

The percent symbol is LUA's escape character, which is why we need `\letterpercent`. The results can be used in a table like the following:

```
\starttabulate[|l|c|r|]
\NC year  \NC \matchresult{1}
          \NC \number \matchresult{1} \NC \NR
\NC month \NC \matchresult{2}
          \NC \number \matchresult{2} \NC \NR
\NC day   \NC \matchresult{3}
          \NC \number \matchresult{3} \NC \NR
\stoptabulate
```

We can rewrite the LUA code in a more efficient way (less code in the main macro):

```
\lua {
  function match.find(expr, str)
   match.expression = expr ;
   match.string = str ;
   match.result = {} ;
   match.start, match.length, match.result[1],
   match.result[2], match.result[3],
   match.result[4], match.result[5],
   match.result[6], match.result[7],
   match.result[8], match.result[9]
     = string.find(match.string,
                   match.expression) ;
 end
}

\def\matchstring#1#2{\lua {
    match.find("#1","#2") ;
} }

\def\matchresult#1{\lua {
    tex.print(match.result[#1]) ;
} }
```

The usage is the same. In the previous example we used `\number` to get rid of leading zeros, but we can also adapt the expression like this:

```
\matchstring
  {0*(\letterpercent d+)
   0*(\letterpercent d+)
   0*(\letterpercent d+)}
  {2005 08 08}
```

The next example takes some close reading, but it demonstrates how to mix TEX and LUA. Let's start with defining some tables, a very LUAish but powerful data structure (the more one reads about LUA, the more one realizes that it is pretty well designed).

```
\lua {
    document = { }
    document.dimens = { }
}
```

Next we store the widths of a bunch of characters in the dimens subtable. (We could have used ε-TEX's char dimension primitives instead.)

```
\dostepwiserecurse{`a}{`z}{1} {
 \setbox\tempbox\hbox{\char\recurselevel}
 \lua {
  document.dimens[\recurselevel]
   = tex.wd[\number\tempbox]
 }
}
```

Now we can calculate the average widths of the characters. Of course this particular case can be done in pure TEX quite conveniently, but one can envision more tricky calculations.

```
\lua {
  local total, n = 0, 0
  for d in pairs(document.dimens) do
    total, n
      = total + document.dimens[d], n + 1
  end
  if n>0 then
    document.mean = total/n
  else
    document.mean = 0
  end
}
```

We leave it to your imagination to envision what the next code will produce (this article is not produced using LUATEX):

```
\mathematics {
 \lua { tex.dimen[0] = document.mean }
 \withoutpt \the\dimen0  =
 \lua { tex.print(document.mean/65536) }
 \approx
 \lua { tex.print(
         math.ceil(document.mean/65536)) }
}
```

Did you notice how we have access to TEX's dimensions? We can read and write them — when a string is assigned, the usual dimensions are interpreted. In a similar fashion we have access to counters.

```
\bgroup
 \count0=10 \count2=30
 \scratchcounter =
   \lua { tex.print((tex.count[0]
                    + tex.count[2])/2) }
 \number\scratchcounter
\egroup
```

Now keep in mind, we're not piping data from TEX into some external program and reading it back in again. Here LUA is an integral part of TEX!

Token registers can be accessed as well:

```
\toks0 = {interesting}
\lua {
  tex.toks[0]
    = string.gsub(tex.toks[0],
                  "(.)", " (\letterpercent1)
")
}
\the\toks0
```

This will return the individual characters of 'interesting' surrounded by parentheses. LUA has sufficient functionality for string manipulations.

Can you envision what this next bit of code does? Again we use a token register but this time we also rescan the result because it contains a control sequence. (The quote is from Hermann Zapf.)

```
\toks0 = {
 Coming back to the use of typefaces in
 electronic publishing: many of the new
 typographers receive their knowledge and
 information about the rules of typography
 from books, from computer magazines or the
 instruction manuals which they get with
 the purchase of a PC or software. There
 is not so much basic instruction, as of
 now, as there was in the old days,
 showing the differences between good and
 bad typographic design. Many people are
 just fascinated by their PC's tricks,
 and think that a widely-praised program,
 called up on the screen, will make
 everything automatic from now on.
}

\lua {
 str = tex.toks[0]
 str = string.gsub(str,"\letterpercent w+",
        function(w)
         if str.len(w) > 4 then
          return "\string\\color[red]{"
                 .. w .. "}"
         else
          return "\string\\color[green]{"
                 .. w .. "}"
         end
        end
       )
 tex.toks[0] = str
}

\scantokens\expandafter{\the\toks0}
```

Words longer than 4 characters will become red, while short words become green. The `..` operator concatenates strings. Of course a solution with more complex input will demand a different approach, and in the end we need the ability to process TEX's internal lists.

We don't yet have access to the `lccode` table but we can already efficiently define vectors. Currently TEX lacks a way to quickly initialize switches, which is a pity because nowadays PDFTEX provides a lot of them.

```
\lua {
  lccodes = { }
}

\dorecurse{255} {
  \lua { lccodes[\recurselevel]
          = \number\lccode\recurselevel ;
       }
}

\luatex {
  for i in pairs(lccodes) do
    tex.print("\string\\lccode" .. i
              .. "=" .. lccodes[i] .. " ")
  end
}
```

In the last example we return to the kind of example that we started with: fully expandable definitions.

```
\lua {
 interface = {}
 interface.noftests = 0
 function interface.oneoftwo(result)
  interface.noftests = interface.noftests+1
  if result then
    tex.print("firstoftwoarguments")
  else
    tex.print("secondoftwoarguments")
   end
 end
}
```

This function returns a given string, depending on the boolean (condition) fed into it. The two possible strings correspond to macro names (ConTEXt has a bunch of those):

```
\long\def\firstoftwoarguments #1#2{#1}
\long\def\secondoftwoarguments#1#2{#2}
```

Now imagine the following definitions:

```
\def\DoIfElse#1#2{%
  \csname\lua{%
      interface.oneoftwo("#1"=="#2")
  }\endcsname
}
```

We feed the string comparison into the function, which returns a string, which in turn is expanded into a control sequence. Without any chance of interference we also keep track of the number of tests. We can ask for this number with:

```
\def\NOfTests
  {\lua{tex.print(interface.noftests)}}
```

The following code shows how to use the test. They all return (typeset) `OK`.

```
\def\xxx{yyy} \def\yyy{yyy}

\DoIfElse{one}{one}{OK}{NO}
\DoIfElse{two}{one}{NO}{OK}
\DoIfElse\xxx \yyy {OK}{NO}
```

What goes in gets expanded, but when it enters LUA it has been turned into a harmless sequence of characters:

```
\DoIfElse
  {this is an \hbox to \hsize{real} mess}
  {and \rm this is even worse}
  {NO}{OK}
```

This test can be used in an `\edef` and fed back into LUA if needed.

```
\edef\zzz{this
          \DoIfElse{a}{b}{or}{and}
          that}

\lua{tex.print("\zzz")}
\lua{tex.print("this
                \DoIfElse{a}{a}{or}{and}
                that")}
```

These examples demonstrate that we can comfortably mix TEX and LUA and use whichever suits the problem best. Also keep in mind that nesting LUA calls is no problem either:

```
\lua{tex.print("this
                \lua{tex.print("---")}
                that")}
```

Of course we need additional trickery. For instance we need a way to escape characters before they enter LUA, or the following will fail:

```
\DoIfElse{one "two" three}
         {one 'two' three}
         {OK}{NO}
```

Instead we need something like this:

```
\DoIfElse
  {\luaesc{one "two" three}}
  {\luaesc{one 'two' three}}
  {OK}{NO}
```

which will comfortably escape the sensitive characters. So, there is some work left to do.

4 The potential

Since we recognized the potential we decided to continue this effort and the lean and mean LUATEX team was there: Hartmut Henkel, Taco Hoekwater & Hans Hagen. We will discuss, meet, develop and finally present a working and stable version, probably around EuroTEX and/or TUG 2006. The binary will start as a fork of PDFTEX, and in the end perhaps replace PDFTEX since its author likes the idea.

In the process we consider removing some of the fuzzy parts of PDFTEX; good candidates for removal are `enctex` and `mltex`, among others, especially if we can preprocess the input stream. We may as well replace/extend some of the subsystems (such as font handling for which we need to go to OPENTYPE anyway) and/or prototype new subsystems. We will provide an interface to plugins; for instance, the paragraph building components that Karel Skoupý is building and has demonstrated at conferences.

In the end we will provide a playground for all kinds of features; for instance, language/script specific font handlers and paragraph builders for languages such as Chinese. Of course my personal intent is to replace (or extend) some macro-defined parts of ConTEXt by alternatives in LUA (think of ConTEXt 4). We can even add functionality that until now was beyond reason to implement. And this is just the start . . .

My thanks to Hartmut Henkel and Taco Hoekwater for their collaboration on this project and this paper, and Karl Berry for editorial help.

Converting METAFONT sources to outline fonts using METAPOST

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague
Czech Republic
piska@fzu.cz
http://www-hep.fzu.cz/~piska/

Abstract

The paper describes a multistep conversion process from METAFONT sources to outline fonts (Adobe Type 1 format). An important step, finding contours, is based on an accurate algorithm fitting the envelope curve of a stroke drawn by a pen along a cubic Bézier curve by the least square method, specially extended (adapted) for a rotated elliptical pen applied, for instance, in the Devanagari font design. After converting the EPS files produced by METAPOST to the corresponding outline representation, the FontForge font editor is used for removing overlap, simplification, autohinting, generating outline fonts, and necessary manual modifications. The conversion results, the faithful Indic Type 1 fonts (significantly more precise and closer to optimal than earlier attempts made by autotracing bitmaps) will be released.

Keywords: font conversion, bitmap fonts, METAFONT, METAPOST, outline fonts, PostScript, Type 1 fonts, approximation, Bézier curves.

Introduction

In 2001 I experimented with approximate conversion METAFONT Indic fonts to the Type 1 format by autotracing bitmaps with the TEXtrace program [11]. I was not satisfied with the results and decided to apply another, analytic approach, to achieve results more precise and also better optimized.

Conversion process

Our procedure consists of studying the font definitions in METAFONT and preparing encoding files. Then the glyph strokes produced by METAPOST are converted to outlines, and the font is assembled, optimized, and autohinted. Finally, it is generated as a Type 1 binary file with FontForge. After verification of visual proofsheet pages some steps are often repeated to correct or improve the final results.

Analysis of METAFONT sources We analyze the METAFONT source texts [7] of a font to select an appropriate strategy of conversion, to find the crucial parameters, like the font size, the italic angle, and the definitions of pens and strokes. Some parameters may be also hidden inside macros. Sometimes, a method for efficient conversion is not apparent. Therefore it is also important to know about the presence and number of METAFONT commands not available in METAPOST [5], for example, using operations with bitmap picture variables.

Creating encoding files Encoding files and encoding vectors define a mapping between the glyph names and their number codes. METAFONT definitions usually do not contain unique glyph names in an explicit form but only as comments. The glyph names are taken from these comments to produce an unambiguous list of PostScript names, i.e. we must find duplicated names and change them to be different. Our preliminary solution inherits the METAFONT comments closely to make glyph identification easier.

Running METAPOST Invoking METAPOST processes the METAFONT sources and produces EPS files. METAPOST together with a macro package `mfplain` ([5], p. 79) allows processing the original or modified (to eliminate METAFONT-specific commands) font sources written in METAFONT and to generate for each glyph a single file in the Encapsulated PostScript format, consisting only of PostScript commands like curves, strokes, affine transformations representing pens, etc., but no bitmap images, in contrast to METAFONT's usual output. Some metric data, e.g. the glyph widths and italic angles, may be lost; we shall restore them later.

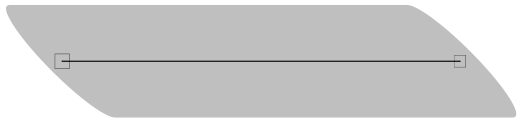


Figure 1: Result of METAPOST.

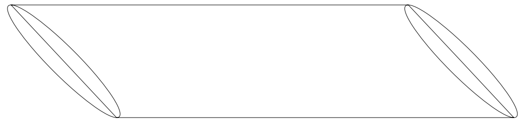


Figure 2: Primary conversion to outlines.

We also need to define a magnification factor, because we have to transform the glyph images to a 1000-unit glyph coordinate system (we use this usual space) with the units in PostScript big points (*bp*) and the font *designsize* in *pt* units. The transformation factor from *pt* to *bp* is 1.00375. Thus in general the magnification factor will be $1000 * 1.00375/\mathit{designsize}$. For $\mathit{designsize} = 10\,\mathrm{pt}$, this is $1000 * 1.00375/10 = 100.375$, for 8 pt it is 125.46875, for 17.28 pt 58.087384, etc.

So, a typical command to call METAPOST is:

```
  mpost '&mfplain \mode=localfont;' \
        mag=100.375'; input' dvng10.mf
```

These files may contain various stroked paths (see figures 1, 9). It is necessary to find contour curves for single strokes and then also common envelope curves for overlapping strokes.

The following lines from the PostScript produced by METAPOST correspond to fig. 1:

```
 0 79.06227 dtransform truncate idtransform
 setlinewidth pop [] 0 setdash
 1 setlinecap 1 setlinejoin 10 setmiterlimit
gsave newpath 119.50958 284.54501 moveto
398.36119 284.54501 lineto
[-0.98387 0.98387 -0.17888 -0.17888 0 0] concat
stroke grestore
```

The `lineto` operator describes the line segment, the `concat` operator applies the affine transformation represented by the preceding normalized matrix (in brackets) denoting the rotated elliptical pen, and `79.06227` ... `setlinewidth` is the scale factor defining the stroke width.

Converting METAPOST products to outlines

The results of METAPOST (strokes) are converted to "primary" outlines. Fitting curves with the least square method is a typical approach to calculate a curve approximation. This method is nothing new and may well have been used in conversion programs developed by Richard Kinch (MetaFog, [6]), Basil Malyshev [9], George Williams (FontForge, [13]) and others. We only apply a few additional conditions. We try to be more precise, but our attempts are still more fragile and unstable than the programs listed above.

All the calculations are in the non-integer value space. We check each segment for accuracy and subdivide it if a chosen limit is exceeded; insert all horizontal and vertical extrema nodes; keep all horizontal/vertical straight lines and control vectors to be exactly horizontal/vertical. The inner part of a contour curve of drawing a rotated elliptical pen even along a simple Bézier path without any intersections may have self-intersections. Therefore we try to find self-intersection points as precisely as possible, if it is possible at all. Unfortunately, sometimes this iteration does not converge. A simplest conversion to outlines is shown in figure 2.

For a given time of the path segment using the affine transformation matrix and its inverse matrix (for a usual pen they are always regular) we can calculate the displacement corresponding to the point lying on the right parallel outline curve (the left one is located symmetrically). Knowing the coordinates of points on the outline curves and also on the pen boundary we can fit them by a cubic Bézier approximation. But a problem is that we do not know whether the points are on the envelope curve, because parts of the outline curves may create loops of arbitrary size being inside a closed area. It depends on complex correlations between the path and the pen.

We also recognize quarter-circles, usually represented in METAFONT by two segments because METAFONT tends to divide curves into octants. To avoid further simplification problems, we do not preserve the 45 degree middle nodes and change the quarter-circles to the accurate single-segment PostScript representation with relative lengths of control vectors $4/3(\sqrt{2}-1) \simeq 0.552285$, cf. R. Kinch [6, p. 236] and Luc Devroye [2]. For an example of our circle approximation see figure 3.

To summarize, in the primary approximation straight lines and circles are represented by the minimal number of segments (because other nodes are unnecessary), and, on the other hand, other outline curves have redundant node points (to preserve a maximal starting accuracy). These intermediate results of the primary conversion to outline are demonstrated in figures 2 and 10.

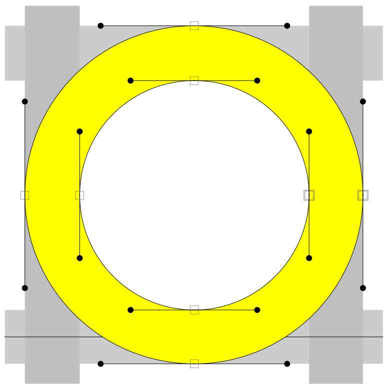


Figure 3: Representation of circles.

Creating a font with FontForge FontForge is a powerful open source font editor. Among its wide range of useful abilities is a "background layer", which may contain bitmap images and line drawings. So, we generate with METAFONT high resolution bitmaps for the font under study, either 2400 dpi (supre mode), or 7254 dpi. The 7254 dpi "device" corresponds to the relation 1 pixel in the PK bitmap ∼ 1 unit in the PostScript glyph space for a 10 pt design size:

```
% (72.27*1000.375/10dpi)=7254.1
mode_param (pixels_per_inch,4000+3254.1);
mode_param (blacker, 0);
mode_param (fillin, 0);
mode_param (o_correction, 1);
```

The resulting GF (or PK) files can be imported to the background as a set of gray pixels to compare with glyph images. (Sometimes, METAFONT with such a high resolution may fail, if the author did not design the font for an arbitrary resolution.)

Font composition We also run mftrace [10] with an appropriate encoding to make a PFB font file. From this file we build a frame for the created font, copy the glyph widths and glyph names, and move the outlines to the background layer (visible as green lines). During a subsequent processing of the font with FontForge we use its internal Spline Font Database format (SFD).

The high resolution bitmap is always huge, so we import it only before a comparison. But the outline contours of the font produced by mftrace are not large and we can store them in the working SFD files permanently. To the foreground layer we import the outlines from the EPS files calculated in the previous step from the original EPS files generated by METAPOST.

The high resolution pixel image gives a close visual bitmap representation of the original METAFONT source. Of course, information about contour curves, intersection points, corners, etc., virtually calculated by METAFONT has been lost. The font outlines autotraced by mftrace from similar bitmaps, despite the inevitable artifacts (bumps, holes, unrecognized corners, ...) give reasonably correct information about the glyphs. Our aim is to obtain another outline representation: more accurate and closer to optimal, minimizing the number of defects.

Having the font in SFD format built from the mftrace output, our next step with FontForge is **removing overlap and optimization** (simplification). We continue processing in the non-integer space to keep accuracy, in particular, not changing the slopes of the neighboring control vectors so as to preserve a smooth transition between segments.

Rounding to integer, hinting and Type 1 font generation FontForge allows for generating PostScript fonts with non-integer point coordinates and, PostScript RIP devices (usually) render these fonts properly. But we have three significant reasons to round coordinates to integers and to generate the Type 1 fonts in integer representation:

- Non-integer values in the PostScript charstring occupy 3 "items". Therefore the integer representation saves storage and the PFB files are smaller.
- The final Type 1 fonts do not need such accuracy after removing overlap and simplification.
- For hinting it would be inconvenient and impractical to use any discrete grid other than integers.

For example, the non-integer Type 1 command occupies 19 items:

```
18153 100 div 212 100 div
14437 100 div -407 100 div
7208 100 div -243 100 div
rrcurveto
```

and after rounding only 7 items:

```
182 2 144 -4 72 -2 rrcurveto
```

It is desirable to minimize the number of items because PostScript interpreters have internal memory limits per glyph. Exceeding these limits causes a **limitcheck** error and rendering fails.

The coordinates of the segments are rounded to integers by a more complex algorithm than a trivial rounding of all the values. First, we round the node points. Then we transform the control vectors according the changes of the nodes, and try to find the control points in the integer grid near the transformed control vectors. Even this sophisticated rounding to integer is not without problems. Sometimes, if the change in x or y in the segment is very

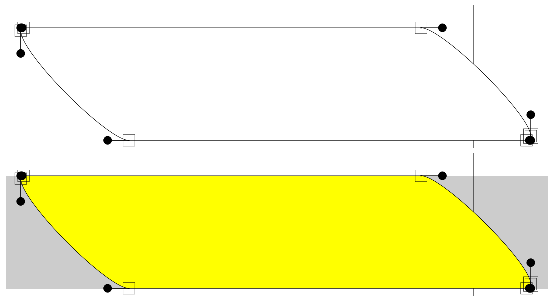


Figure 4: Final font in an outline form and a hinted proofsheet (clip).

small (e.g. about 1 unit) or a segment is too short (in both directions) no good selection may exist and a manual adjustment is then necessary, probably with the loss of closeness, accuracy or symmetry of the approximation.

No special additional program for hinting has been developed or applied. The automatic autohinting tool in FontForge is used, and any unsatisfactory results should be corrected manually.

Finally, we generate the Type 1 binary font with FontForge, rounded to integer coordinates and (auto)hinted as described. See figure 4.

Results

To make font auditing and verification quicker and more efficient, we have developed tools for generation of visual proofsheets in PDF. These support a fast overview of all glyph images, display of outline curves with node and control points and vectors, and hinting zones. They can also produce warnings about undesirable situations such as missing nodes at extrema, presence of inflection inside a segment, a non-smooth transition between segments, etc.

Our aim is to fulfill the *Type 1 conventions* [1]. Therefore we include the extrema nodes (they may be omitted if they are really redundant), exclude other unnecessary node points, and preserve smooth connections between the adjacent segments. We also keep the straight lines, corners and arcs after conversion, and avoid any false bumps, holes or steps absent in the original METAFONT sources.

In this paper, some selected figures have the node points (squares), the control points (bullets) and the control vectors enlarged for readability. In a real working process they are colored and small as in other proofsheets, when we zoom in on interesting details only if we need to check them.

The principal and auxiliary algorithms are still under development and adaptation for new fonts.

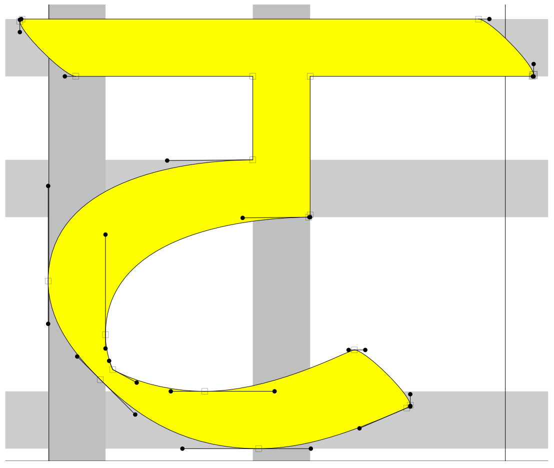


Figure 5: `dvng10`: tta of Frans Velthuis.

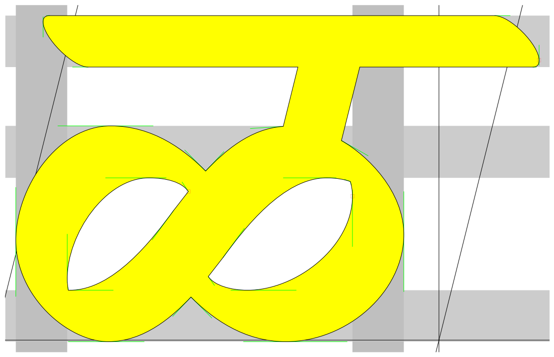


Figure 6: `dvngbi10`: lla of Frans Velthuis.

The programs are written in `awk` or `gawk` [3]. For Type 1 font handling the `t1utils` [8] are used.

Several pictures illustrate the intermediate and final results in the conversion of METAFONT fonts to the Type 1 format: figures 2, 4, 10, 11, 15, and 16.

Indic fonts A basic goal of the work is more precise outline versions of the free METAFONT Indic fonts available from CTAN: Devanagari, Sanskrit, Gurmukhi, Punjabi, Bangla, Sinhala, Malayalam, Telugu, Kannada, Tamil, and Tibetan. At the time of writing, not all the above fonts have been converted. Also, the Oriya fonts are impractical because they widely use METAFONT bitmap picture commands.

Figures 5, 6, 12, 13 (all Devanagari), and 14 (for Malayalam) show the results to date.

Chinese fonts We have also tried to convert two small single fonts with Chinese ideographs created in METAFONT: the Hóng-Zì font (128 glyphs) designed by Javier Rodríguez Laguna [12] (version 0.5 of 2005-03-23), shown in fig. 7; and `china10`, a font from the `china2e` package [4] containing Chinese

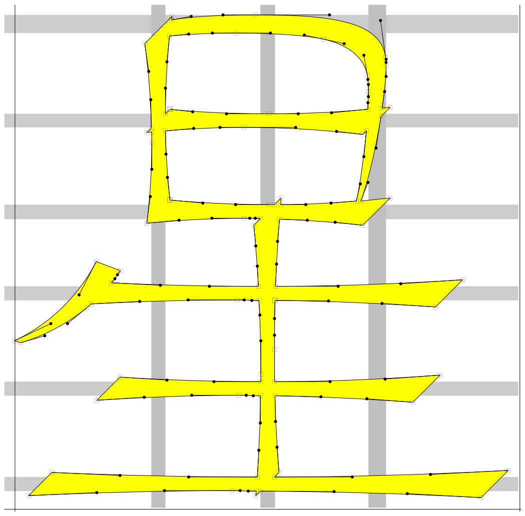


Figure 7: Hóng-Zì: xing1 of Javier Laguna.

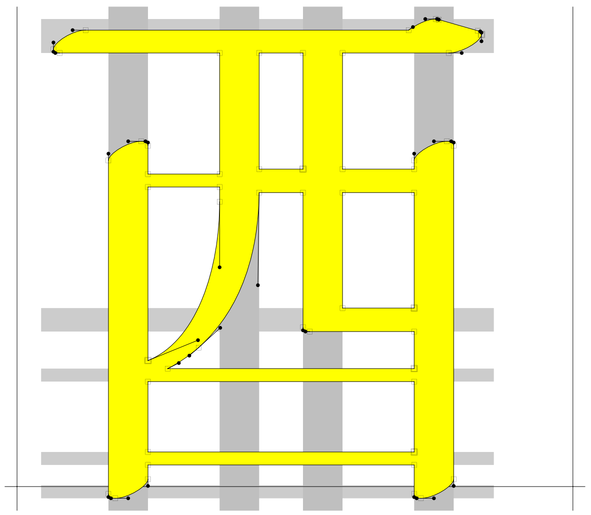


Figure 8: `china10`: yeu of Udo Heyl.

calendar symbols produced by Udo Heyl (1997), in fig. 8.

Conclusion

In this article we describe a font conversion process and briefly discuss some selected problems. Creating *precise* fonts is always difficult, time-consuming and never-ending work, regardless of the approach chosen. We plan to again verify all the glyphs to improve hinting and polish the outlines to remove tiny artifacts. It will also be useful to make the glyph names of the Indic glyphs common for all languages; this is not trivial because the fonts contain many various ligatures, special signs or variants not covered in the Unicode standards.

Acknowledgements

I would like to thank all the authors of the free conversion programs, the authors of the public METAFONT fonts for Indic languages, other sources and program packages used in this work.

References

[1] Adobe Systems Inc. *Adobe Type 1 Font Format.* Addison-Wesley, 1990.

[2] Luc Devroye. “Formatting Font Formats”, *TUGboat* **24**(3), pp. 588–596, 2003.

[3] Free Software Foundation. GNU awk, `http://www.gnu.org/software/gawk`.

[4] Udo Heyl. `china2e`, `CTAN:macros/latex/contrib/china2e`, 1997.

[5] John D. Hobby. *A user’s manual for METAPOST*. AT&T Bell Laboratories, Computing Science Technical Report 162, 1994.

[6] Richard J. Kinch. “MetaFog: Converting METAFONT Shapes to Contours”, *TUGboat* **16**(3), pp. 233–243, 1995.

[7] Donald E. Knuth. *The METAFONTbook.* Addison-Wesley, 1986. Volume C of *Computers and Typesetting.*

[8] Eddie Kohler. `t1utils`: Type 1 font utilities, `http://freshmeat.net/projects/t1utils`.

[9] Basil K. Malyshev, “Problems of the conversion of METAFONT fonts to PostScript Type 1”, *TUGboat* **16**(1), pp. 60–68, 1995.

[10] Han-Wen Nienhuys. `mftrace`, `http://www.cs.uu.nl/~hanwen/mftrace`.

[11] Karel Píška. “A conversion of public Indic fonts from METAFONT into Type 1 format with TEXtrace.” *TUGboat* **23**(1), pp. 70–73, 2002.

[12] Javier Rodríguez Laguna. Hong-Zi: a Chinese METAFONT, `http://hongzi.sourceforge.net`, 2005.

[13] George Williams. FontForge: an outline font editor, `http://fontforge.sourceforge.net`.

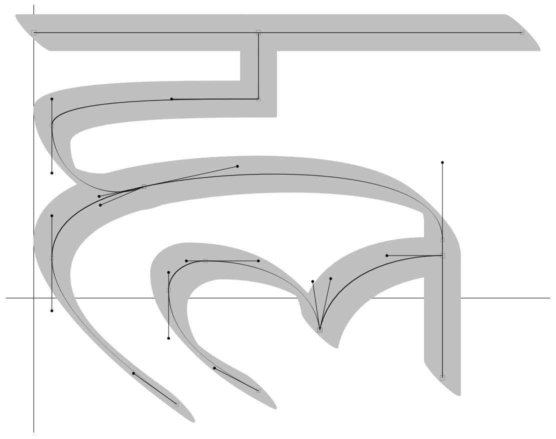


Figure 9: `dvng10` l_h: METAPOST output.

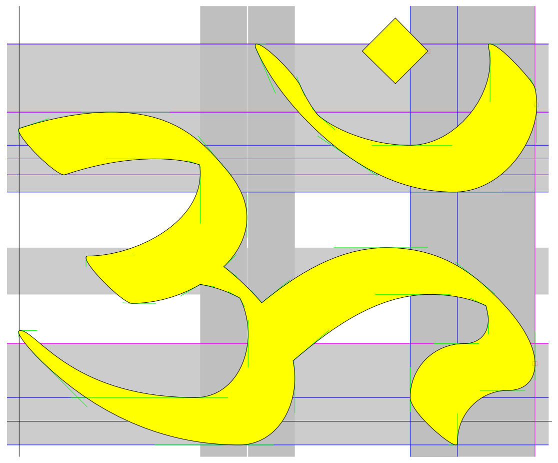


Figure 12: `dvng10`: om of Frans Velthuis.

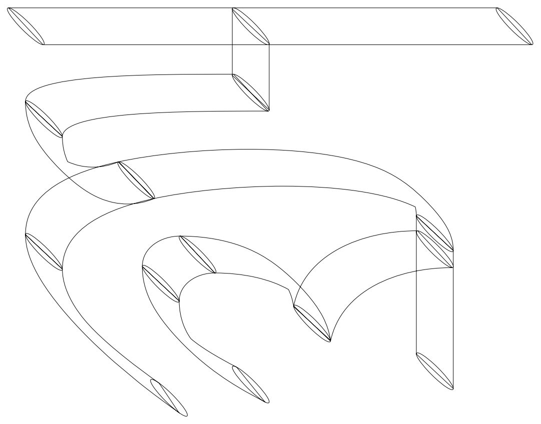


Figure 10: `dvng10` l_h: primary outlines.

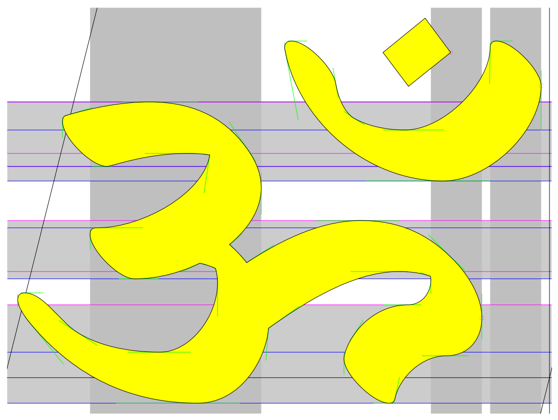


Figure 13: `dvngbi10`: om of Frans Velthuis.

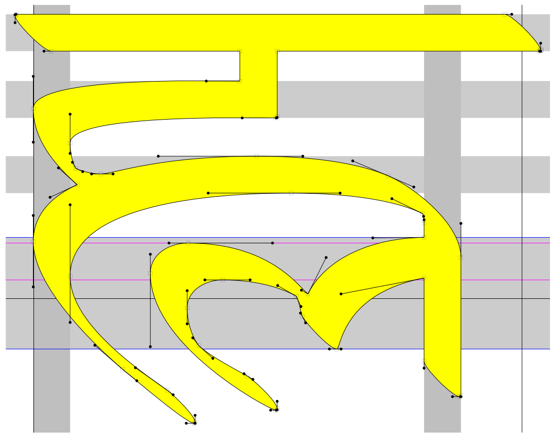


Figure 11: `dvng10` l_h: Type 1 font proofsheet.

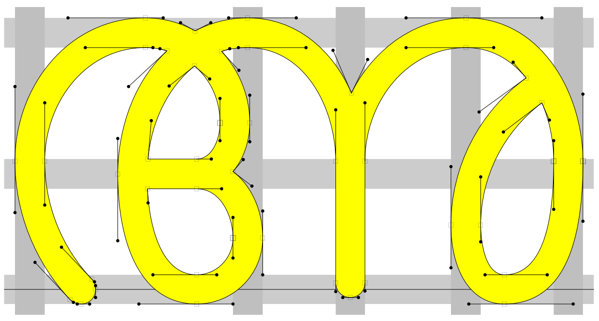


Figure 14: `mm10`: a of Jeroen Hellingman.

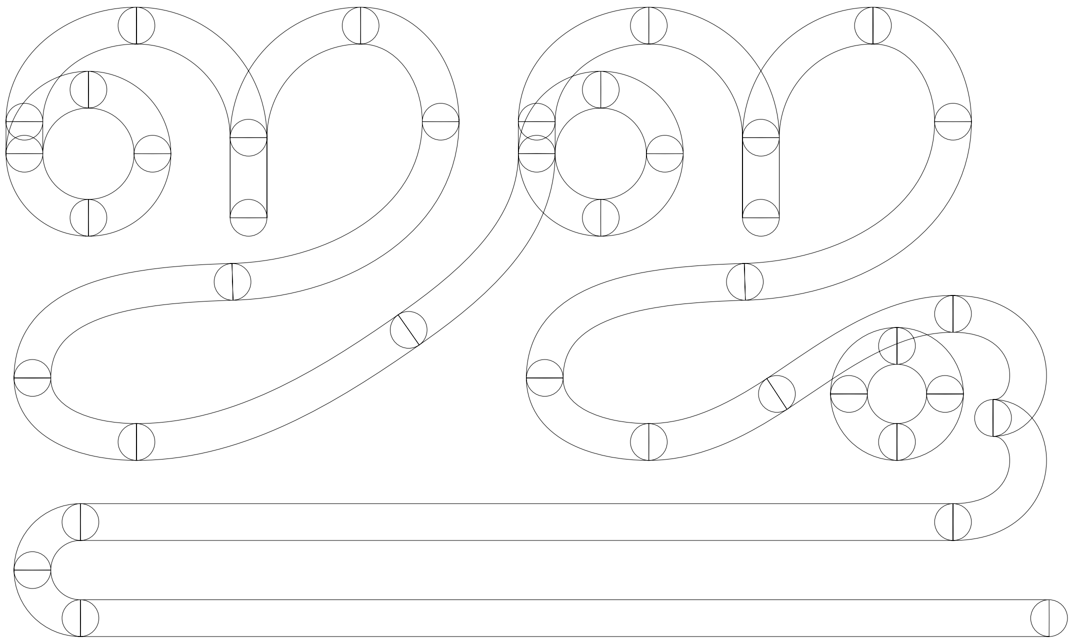


Figure 15: `mm10` j_juu: METAPOST output converted to primary outlines.

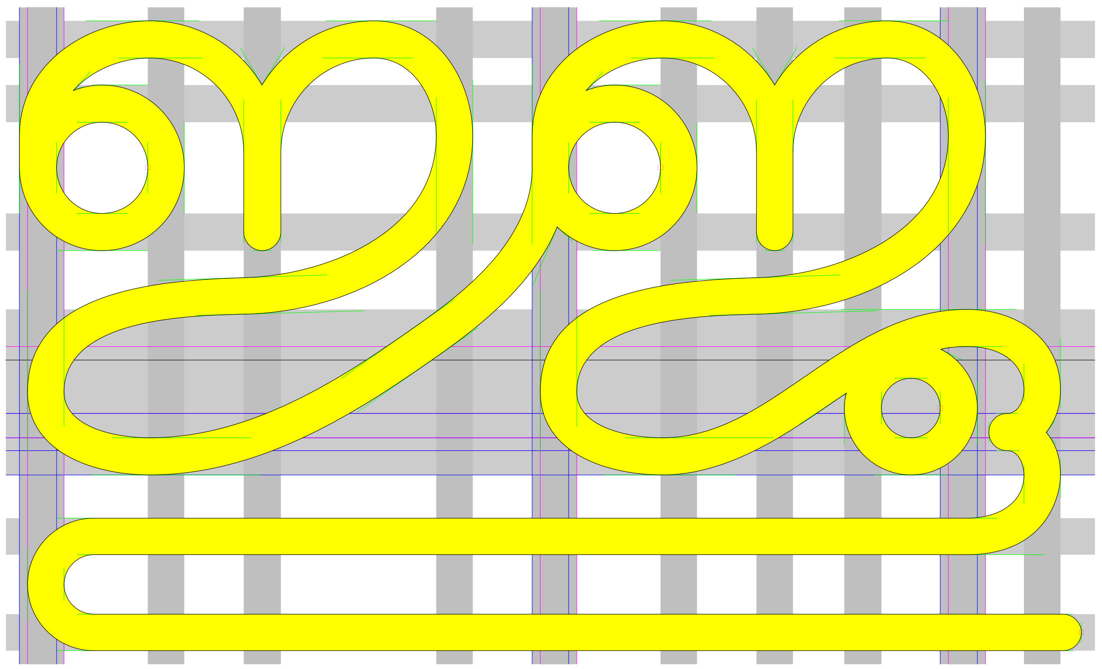


Figure 16: `mm10` j_juu: Type 1 font proofsheet with hints.

Moving from bytes to words to semantics

S.K. Venkatesan
TnQ Books and Journals,
8, Second Crescent Park Road,
Gandhinagar, Adyar, Chennai 600 020, India
skvenkat@tnq.co.in

Abstract

Starting from several bytes of ASCII or Unicode strings one can construct a typeset page readable by the community that understands that script. Unfortunately, it still remains unreadable by the larger community of people who don't understand the script. Instead, if this page had been coded at the level of a semantic word, with each word denoting a unique semantic identity, with sufficient markers (the curly bracket nesting being one such example) for grammar and flow, then it would be able to display itself in each language without ambiguity. The eccentricities of ligatures, capitalization, joining of letters could then be handled accurately. Hyphenation, for example, could then be based not on patterns but on the semantics of the word. For example, hyphenation in English tends to depend on whether the word is a noun or a verb. In this work we discuss the possible atomic words (atoms of course have their own protons, electrons, etc.) of a language and the semantic markups that could lead us to such an ideal.

"Somehow I too must find a way of making things; not plastic, written things, but realities that arise from the craft itself. Somehow I too must discover the smallest constituent element, the cell of my art, the tangible immaterial means of expressing everything."

—Rainer Maria Rilke

Introduction

At a superficial level, it is tempting to identify a language by the script (glyphs) it uses, or, in more modern terms, by the Unicode values of the characters used in the document. Both methods would fail. The first one would fail for scripts that are common to many language users and the second will fail miserably, as Unicode fonts are rarely used by word-processors and typesetters. Many of the e-mail transactions in Indian languages are done through their phonetic Latin script equivalents. Even the complex Chinese language can be phonetically written using the Latin script in the Pinyin system. There was an earlier attempt at Latinizing the Chinese language known as the Wade–Giles system, but due to its shortcomings the new Pinyin system was formed.

Human speech, especially its root-words and structure, is of a language formed much before the introduction of writing systems. Some languages like Vietnamese and Malay–Indonesian changed to their currently used Latin scripts only recently. My native language, Tamil, for example, has a Brahmic script that is similar to Sanskrit but there are even some claims that in its root-word and structural formations it is akin to the primitive Sumer, Elamite and Mande languages. Although the Japanese and Korean[1] languages use the Chinese Han characters (ideographs), the languages themselves have more root-word or structural similarities with the Dravidian languages than with the Chinese languages. For ease of pronunciation the Japanese language uses a smaller subset of simplified Han characters as its alphabet.

Chinese Han script is a good writing system—the characters can carry meanings beyond the spoken language limit, and reading ideographs is faster than reading phonograms such as alphabets, because ideographs directly indicate the meaning while phonograms are changed to pronunciation first and only then the meaning is recognised. People with dyslexia find it difficult to read phonograms but they can understand ideographs easily. It is clear that the Chinese ideographs can have wider applications. Most importantly, the Han script has managed to remain the script that links all the languages

[1] The Korean language was originally written using the Chinese characters; it is now mainly written in Hangul, the Korean writing system, optionally incorporating Hanja to write Sino-Korean words.

of China, creating a single language identity.

The Chinese languages are isolating languages in which the word order, with the help of distinct particles, creates the structure and meaning of the sentence.[2] In the Latin script, a similar artificial isolating language, `lojban`, was created and has generated considerable interest [1]. It is possible to associate an EBNF (a kind of SGML-DTD structure) to `lojban` that makes the language parseable, with words that have unambiguous semantic meaning.[3] `lojban` has clearly demonstrated that an attempt at precision does not make the system rigid. In fact, such attempts surprisingly add to the richness of the language. The root-words in `lojban` have also been carefully selected so as to maintain certain cultural neutrality by including elements from Turkish, Chinese, English, Indian, Russian, Spanish, French, Japanese, and German.

This paper here draws much of its inspiration from the `lojban` effort, and is an attempt to bring `lojban` within the context of the TEX paradigm. Unlike both Chinese and `lojban` we make no attempt at a speech level, conceding that area entirely to the natural languages.

Root-word formation in natural languages

Among the commonly used Dravidian group of languages of India, Tamil has managed to develop by inward growth, rather than by borrowing words from distinct languages such as Sanskrit. Relatively few languages in the world have remained isolated, and managed to avoid direct borrowing of words from other languages. In Europe, the Basque language has had such a self-sustained internal development, but Basque is an amalgamating language that is difficult to learn.

Tamil is an agglutinating language that is explicit and logical, with rules that are easy for children to learn. In Tamil, the formation of `brivla` (compound-words) from `gismu` (root-words) is quite logical and consistent. In Arabic and Hebrew, words are constructed by weaving vowels over root consonant patterns; in line with this in Tamil, the consonants are considered the 'true' (material truth) letters whereas the vowels are considered the ones that provide 'life' (spirit) to it.

It is also no accident that these inward-looking languages are also ones that belonged to matriarchal clan societies with higher in-breeding tendencies. The overthrow of these closed clan societies also meant the mingling and mangling of words used by these societies, leading to the present set of large complex words used in each language. It is however extremely difficult now to look back in time and reconstruct these morphologies in a coherent and consistent manner.

One inspired attempt is the attempt by Asko Parpola [2] who observed a link between the Dravidian languages and the Indus Valley script. The words fish (mIn), star (min-mini), lightning (min-al) identified as a fig-tree (al) with aerial roots from heaven through the astrological associations of Saturn, the slow moving dark planet with the Tortoise, the fish with a roof. The darkness indicating 'mai' associated with tortoise (á-mai). The words 'mal' darkness and 'vel' whiteness are associated with the deities Kannan and Murukan, this being the Tamil equivalent of the Yang and the Yin, at eternal mythological war with their opposites. There is also the undertone of overthrow of matriarchal Yin by the Yang. Like Tamil, perhaps the Chinese languages also have mystical beginnings that spring from tortoise shells, I-ching and soothsayers.

TEX as a paradigm for a new language formation

The industrial and scientific age has also introduced new sets of problems and solutions that require a drastically different outlook from that of the past. The TEX language has been supporting complex scientific symbols and macros making it an ideal platform for a fresh attempt to formulate a mechanism for a modern content-oriented language.

If you are familiar with TEX then these examples will make sense to you:

0. I go there (English)
= *Nan*(I) *ange*(there) *po*(go) (Tamil)

```
\go{0}{I}{there}
```

1. I went there = *Nan ange po-nen*

```
\go{1}{I}{there}
```

2. I am-going there = *Nan ange po-(ki)ren*

```
\go{2}{I}{there}
```

3. I am-going-to-go there = *Nan ange po-ven*

```
\go{3}{I}{there}
```

[2] Even the inflexional English language is showing some tendencies of becoming an isolating language.

[3] `lojban` uses only a subset (lower case) of basic Latin characters (specifically, the letters a to z excluding h, q and w), while uppercase letters are reserved for characters in words of foreign origin that require deviation from `lojban` phonology. We will follow this tradition here.

English, as it is practised today, doesn't have much of a future (tense[4]) for precision semantics, as it is loaded with ambiguity.

You can write Perl macros for writing the above three statements in TEX:

```
\perlnewcommand{\go}[4]
{
 my $smiti_0=$_[0]; my $smiti_1=$_[1];
 my $smiti_2=$_[2]; my $smiti_3=$_[3];
 if($smiti_0==0) { $klama=' go ';}
 if($smiti_[0]==1) { $klama=' went ';}
 if($smiti_0==2) { $klama=' am going ';}
 if($smiti_0==3) {
            $klama=' am going to go '; }
 if($smiti_2=~m/^[A-Z]/) {
          $klama=$klama . ' to ';}
 $text=$smiti_1. $klama . $smiti_2;
 return $text;
 }
\perlnewcommand{\po}[4]
{
 my $smiti_0=$_[0]; my $smiti_1=$_[1];
 my $smiti_2=$_[2]; my $smiti_3=$_[3];
 my $klama='po';
 if($smiti_0==0) { }
 if($smiti_[0]==1) { $klama.='nen';}
 if($smiti_0==2) { $klama.='(ki)ren';}
 if($smiti_0==3) { $klama.='ven'; }
 if($smiti_2=~m/^[A-Z]/) {$smiti_2.='ku';}
 $text=$smiti_1.' '.$smiti_2.' '.$klama;
 return $text;
 }
```

If you are a `lojban`ist then you would write:

```
\klama{#}{mi}{ta}
```

However, all this doesn't prevent you from talking non-sense:

```
\go{3}{I}{yesterday}
```

i.e., I am-going-to-go yesterday. The extra particle 'to' was not required here but it comes into play when we have a specific place, e.g. for

```
\go{0}{I}{Wuhan}
```

we have

> I go to Wuhan (English) = Naan Wuhan-ukku po (Tamil)

Tamil being an agglutinating language, the particle 'to' (-ukku) modifies the ending of the place-noun.

I have tried to illustrate by a simple example how languages belonging to very distinct families can be simplified by similar macros. English, like Chinese has a Subject-Verb-Object (SVO) order, while Tamil has Subject-Object-Verb (SOV) order,[5] but our SenseTEX way of writing has made it Verb-Subject-Object (VSO), more like Hebrew, Arabic and Celtic. We can relax this:

```
\SenseTeX{I,\go,Wuhan}
```

The SenseTEX environment will be discussed further in the next section.

SenseTEX

If words can be understood in terms of their underlying meanings, then they can be cross-sectioned, as first pointed out by George Thompson [3], using synonymity and antonymity to a smaller class. Clusters of related adjectives can also be formed. It is also possible to associate a unique number to this synonymous class of words known as the sense number. With more effort the cob-web of words (the cob-web being a graph, loosely speaking) can be cut down to a tree. This can be done by imposing a hypernym–hyponym hierarchy on all words. As this tree travels from the root to the leaves, it traverses from the abstract generalizations (groupings) to the concrete word (from the heaven to the earth, but upside down with roots in the clouds, like the sacred banyan tree).

Speaking of heaven, a search for the word "coke" in any search engines would get results for all the word-senses (coca-cola, charcoal, cocaine, etc.). It is not possible to be word-sense-specific unless the document has sense numbers indicated. (Although recently a search engine `http://beta.previewseek.com`, made by a company based in London, claims to do just that, with of course the usual ambitious claims about patented technology etc., to deter other search engines from reverse engineering it.)

Our approach here is quite straightforward. It uses no proprietary technology nor anything dificult to understand. It just adds value (sense numbers) to TEX when the author needs to do just that.

The user must associate either a sense-number or `lojban` word to each word he uses within the SenseTEX environment, e.g.,

```
\SenseTeX{coke}
```

would not parse — unless a `sensetex.cfg` file has either a valid `lojban` word or a sense-number (as in WordNet) associated with `coke`. More than one word can be used in SenseTEX, e.g.,

```
\SenseTeX{I go Wuhan}
```

[4] For instance, "I will go there" indicates future tense but "will" can add an extra emphasis meaning "I am definitely going there". Moreover, if you just wish to say "I go there tomorrow" — it is not possible.

[5] According to some linguists, Chinese is showing some tendency towards becoming a SOV language.

if each word has a unique entry in `sensetex.cfg`. If you are adept enough to make Λ, Ω, XƎTEX, or ConTEXt work for you, then you can use your own language's script instead of Latin.

We now add the SenseTEX macros:

```
\SenseTeX{\go{#}{I}{Wuhan}}
```

which can be trimmed down to this:

```
\SenseTeX{{I,\go{#},Wuhan}}
```

We can add further sentences, such as

```
\SenseTeX{{I,\go{#},Wuhan},\then,
{I,\go{#},Shanghai}}
```

which could mean "First, I go to Wuhan and then to Shanghai".

Curly brackets can be used to break sentences to logical pieces and double backslashes can be used as paragraph breaks. One must remember that except for the macros the SenseTEX environment behaves just like TEX. For predicate-words like `\go` and `\then`, not just a sense-number (or `lojban` word) but also a SenseTEX macro must be added in the `sensetex.cfg` file.

How are these sense-numbers indicated in print? One way is to print them on top of each word in a smaller point-size (here the `lojban` word 'klama'):

$$\overset{\text{klama}}{\text{go}}$$

If having SenseTEX change the printed output is not desired, then the **hypertex** package can be used and sense numbers (or `lojban` words) can be embedded in hyperlinks that leads to the entries in the sense dictionaries, either in your local system or on the web, e.g.

```
\href{http://www.ctan.org
/macros/sensetex/lojban.htm#klama}{go}
```

Conclusion

SenseTEX is a small beginning. Its future, like everything else, depends on the extensive effort required to build the `sensetex.cfg` file and a sense number based dictionary. From this small beginning one can then perhaps navigate the turbid waters of syntax using macros.

References

[1] `http://www.lojban.org`; see also Hong Feng, "The marriage of TEX and `lojban`", *TUGboat* **23**(1), 46–48, 2003.

[2] Asko Parpola, *Deciphering the Indus script*, Second paperback edition, Cambridge University Press, 2003, ISBN 0-521-79566-4.

[3] `http://wordnet.princeton.edu`

Abstracts

The design of TeX and METAFONT: A retrospective

Nelson Beebe, University of Utah

This article looks back at the design of TeX and METAFONT, and analyzes how they were affected by architectures, operating systems, programming languages, and resource limits of the computing world at the time of their creation by a remarkable programmer and human being, Donald E. Knuth. This paper is dedicated to him, with deep gratitude for the continued inspiration and learning that I've received from his software, his scientific writing, and our occasional personal encounters over the last 25+ years.

(This was also the keynote address at the Practical TeX 2005 conference, and was printed in *TUGboat* **26**:1. *Ed.*)

Practical use of `\special` commands in DVIPDFM*x*

Jin-Hwan Cho, University of Suwon

`\special` commands in TeX provide the only way to communicate arbitrary information to DVI drivers. DVIPDFM*x*, one such driver, translates the standard DVI output of TeX into the PDF format defined by Adobe for platform independent transmission of digital documents.

In this presentation, we discuss the `\special` commands supported by DVIPDFM*x* and show some practical applications for package designers as well as for TeX end users.

Strategies for including graphics in LaTeX documents

Klaus Höppner, DANTE e.V. and TUG

This talk presents strategies for including graphics into LaTeX documents. It shows the usage of the standard graphics packages of LaTeX as well as an introduction to different graphics formats. Some external tools for converting graphics formats are discussed.

(This paper was also presented at the Practical TeX 2005 conference, and was printed in *TUGboat* **26**:1. *Ed.*)

Wavelet transformations and Chinese font design

Hong Feng, CTUG and Ron's Datacom Inc.

Originally, the fonts used for the TeX system were designed with the METAFONT program. In the past two decades, the wavelet transformation has seen wide application, and it can also be applied in font design for TeX. METAFONT (or MetaPost) and the wavelet transformation can be mutually complementary in Chinese font design.

(We hope to publish the full paper in a future issue of *TUGboat*. *Ed.*)

LaTeX maintenance and development

Chris Rowley, LaTeX Project and Open University

This talk gives a brief history of the LaTeX Project, with some insights into what is involved in the enhancement and maintenance of a robust and widely used software system for the automated formatting of complex documents.

Grid-based typesetting in LaTeX

Philip Taylor, TUG and University of London

The first edition of Rosalind Gibson's *Principles of Nutritional Assessment* was jointly typeset by her husband Ian and myself in the years preceding its publication in 1990; the preparation of this edition was the subject of one of my very first talks at a TUG meeting. Now, fifteen years later, Ian and I have again collaborated in the typesetting of the second edition, which — unlike the first — is typeset in two columns on a strict grid. LaTeX is not easily coerced into grid-based typesetting, so the main thread of this talk will be the various measures we used to achieve the desired effect.

(We hope to publish the full paper in a future issue of *TUGboat*. *Ed.*)

Calendar

2005

Nov 29 – Dec 2	Seybold Seminars, Content Creation and Asset Management, San Francisco. For information, visit `http://www.seybold365.com/2005/`.
Dec 3	MaTeX (Hungarian TeX Users Group) Conference, Budapest. For information, visit `http://www.matexhu.org/`.
Dec 15	Deadline for submission to the annual typography and type design exhibition competitions of the Type Directors Club. For information, visit `http://www.tdc.org/calls/tdc522006` (typography) and `http://www.tdc.org/calls/tdc22006` (type design).

2006

Mar	DANTE 2006, 34th meeting, Technische Universität Berlin, Germany. For information, visit `http://www.dante.de/events/`.
Mar 1	**Deadline for submission of abstracts for EuroTeX 2006**. For information, visit `http://www.matexhu.org/`.
Mar 6 – 10	Rare Book School, University of Virginia, Charlottesville, Virginia. One-week courses on bibliography and electronic texts. For information, visit `http://www.virginia.edu/oldbooks`.
Mar 6 – 8	29th Internationalization and Unicode Conference, San Francisco, California. For information, visit `http://www.unicodeconference.org`.
Apr 1	**Deadline for submission of abstracts for Practical TeX 2006**. For information, visit `http://www.tug.org/practicaltex2006/`.
Apr 29 – May 3	BachoTeX 2006, 14th annual meeting of the Polish TeX Users' Group, GUST, Bachotek, Brodnica Lake District, Poland. For information, visit `http://www.gust.org.pl/BachoTeX/`.
Jun 5 – Jul 28	Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on type, bookmaking, printing, and related topics. For information, visit `http://www.virginia.edu/oldbooks`.
Jun 30	**Deadline for submission of papers for TUG 2006**. For information, visit `http://www.tug.org/tug2006/`.
Jul 3 – 4	"Jobbing printing — the stuff of life", joint conference of the Printing Historical Society and The Ephemera Society, University of Reading, UK. For information, visit `http://www.printinghistoricalsociety.org.uk/events/`.
Jul 4 – 9	ALLC/ACH-2005, Joint International Conference of the Association for Literary and Linguistic Computing and Association for Computers and the Humanities, "Digital Humanities 2006", Université Paris–Sorbonne. For information, visit `http://www.allc-ach2006.colloques.paris-sorbonne.fr/` or the organization web site at `http://www.ach.org`.
Jul 5 – 8	16th EuroTeX Conference, Hungary. For information, visit `http://www.matexhu.org/`.
Jul 11 – 14	SHARP Conference (Society for the History of Authorship, Reading and Publishing), "Trading Books — Trading Ideas", The Hague & Leiden, Netherlands. For information, visit `http://sharpweb.org/`. In conjunction with the 400th anniversary of Rembrandt's birth in Leiden; for information, visit `http://www.rembrandt400.com/`.

Status as of 16 November 2005

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at `http://www.tug.org/calendar/`.

Additional type-related events are listed in the Typophile calendar, at `http://www.icalx.com/html/typophile/month.php?cal=Typophile`.

Practical TEX 2006
Rutgers University, Busch Campus, Piscataway, New Jersey.

Jul 25 – 28	Pre-conference workshop.
Jul 30 – Aug 1	A user-oriented conference sponsored by TUG. For information, visit `http://www.tug.org/practicaltex2006/`.

Jul 30 – Aug 3	SIGGRAPH 2006, Boston, Massachusetts. For information, visit `http://www.siggraph.org/s2006/`.
Sep 29 – 30	American Printing History Association conference, "The Atlantic World of Print in the Age of Franklin", Philadelphia, Pennsylvania. For information, visit `http://www.printinghistory.org/htm/conference/`.
Oct 7	DIY (Do It Yourself) Book Festival, Los Angeles, California. For information, visit `http://www.diyconvention.com/`.
Oct 20 – 22	The Fourth International Conference on the Book, "Save, Change or Discard: Tradition and Innovation in the World of Books", Emerson College, Boston, Massachusetts. For information, visit `http://book-conference.com/`.

TUG 2006
Digital Typography & Electronic Publishing: Localization & Internationalization, Marrakesh, Morocco.

Nov 7 – 8	Pre-conference tutorials.
Nov 9 – 11	The 27th annual meeting of the TEX Users Group. For information, visit `http://www.tug.org/tug2006/`.

TUG'05 participants at Wudang Mountain, the birthplace of Taoism, in their traditional Taoist robes (hand-tailored as a conference gift).

Front row: Jiahui Wang, Devi Govalam, Robin Laakso (behind), Candy Yiu, Joseph Rajendra, Le Khanh Au Duong, Jonathan Kew, Karel Píška

Back row: Kevin Warnock, Maik Kündig, Hong Feng, Hartmut Henkel, Steve Grathwohl, Hans Hagen, Karel Skoupý, Phil Taylor, Jerzy Ludwichowski, S. K. Venkatesan

2005 TEX Users Group Membership Form

TEX USERS GROUP

Promoting the use of TEX throughout the world.

mailing address:
P.O. Box 2311
Portland, OR 97208-2311 USA

shipping address:
1466 NW Naito PKWY, Suite 3141
Portland, OR 97209-2820 USA

phone:	+1 503-223-9994
fax:	+1 503-223-3960
email:	`office@tug.org`
web:	`http://www.tug.org`

President	Karl Berry
Vice-President	Kaja Christiansen
Treasurer	David Walden
Secretary	Susan DeMeritt
Executive Director	Robin Laakso

TUG membership rates are listed below. Please check the appropriate boxes and mail the completed form with payment (in US dollars) to the mailing address at left. If paying by credit/debit card, you may alternatively fax the form to the number at left or join online at `http://tug.org/join.html`. The web page also provides more information than we have room for here.

Status (check one) ☐ New member ☐ Renewing member

	Rate	Amount
☐ **Regular membership** for 2005	$75	______
☐ **Special membership** for 2005 You may join at this special rate if you are a senior (62+), student, new graduate, or from a country with a modest economy. Please circle accordingly. See `http://tug.org/join.html` for more information.	$45	______
☐ **Subscription** for 2005 (non-voting)	$85	______
☐ **Institutional membership** for 2005 Includes up to eight individual memberships.	$500	______
☐ **Send me CTAN 2005 on CD** (shipped on DVD to everyone)	n/a	
☐ **If instead of TEX Live 2005** with your membership, you want the 2004 software delivered right away, check here.	n/a	
Last year's materials (in addition to 2005)		
☐ *TUGboat* volume for 2004 (3 issues)	$20	______
☐ TEX Collection 2004 2 CD's & 1 DVD with proTEXt, TEX Live, CTAN.	$20	______
☐ CTAN 2004 CD-ROMs	$15	______
Voluntary donations		
☐ General TUG contribution		______
☐ Bursary Fund contribution Financial assistance for attending the TUG Annual Meeting.		______
☐ TEX Development Fund contribution Financial assistance for technical projects.		______
Total $		______

Tax deduction: $40 of the regular membership fee is deductible, at least in the US.
Multi-year orders: To join for more than one year at this year's rate, just multiply.

Payment (check one) ☐ Payment enclosed ☐ Visa/MasterCard/AmEx

Account Number: ______________________ Exp. date: ______

Signature: ______________________

Privacy: TUG uses your personal information only to send products, publications, notices, and (for voting members) official ballots. TUG does not sell or otherwise provide its membership list to anyone.

Electronic notices will generally reach you much earlier than printed ones. However, you may choose not to receive any email from TUG, if you prefer.

☐ Do not send me any TUG notices via email.

Name ______________________

Department ______________________

Institution ______________________

Address ______________________

____

City ______________ State/Province ______________

Postal code ______________ Country ______________

Email address ______________________

Phone ______________ Fax ______________

Position ______________ Affiliation ______________

Institutional Members

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

Banca d'Italia, *Roma*, *Italy*

Center for Computing Science, *Bowie*, *Maryland*

Certicom Corp., *Mississauga*, *Ontario Canada*

CNRS - IDRIS, *Orsay*, *France*

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*

MacKichan Software, *Washington/New Mexico*, *USA*

Masaryk University, Faculty of Informatics, *Brno*, *Czechoslovakia*

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

Princeton University, Department of Mathematics, *Princeton*, *New Jersey*

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

Stanford Linear Accelerator Center (SLAC), *Stanford*, *California*

Stanford University, Computer Science Department, *Stanford*, *California*

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

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

University of Delaware, Computing and Network Services, *Newark*, *Delaware*

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

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

Uppsala University, *Uppsala*, *Sweden*

Vanderbilt University, *Nashville*, *Tennessee*

TeX Consultants

Ogawa, Arthur
40453 Cherokee Oaks Drive
Three Rivers, CA 93271-9743
(209) 561-4585
Email: `arthur_ogawa@teleport.com`
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TeX macros and LaTeX 2_ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LaTeX, TeX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: `boris@lk.net`
I provide training, consulting, software design and implementation for Unix, Perl, SQL, TeX, and LaTeX. I have authored several popular packages for LaTeX and `latex2html`. I have contributed to several web-based projects for generating and typesetting reports.
For more information please visit my web page: `http://users.lk.net/~borisv`.

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 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.

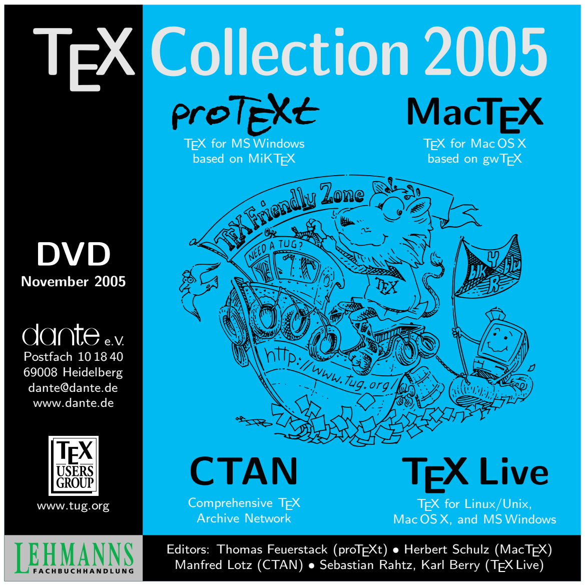
The TUG office mentions the consultants listed here to people seeking TeX workers. If you'd like to be included, or place a larger ad in *TUGboat*, please contact the office or see our web pages:

TeX Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97208-2311, U.S.A.

Phone:	+1 503 223-9994
Fax:	+1 503 223-3960
Email:	`office@tug.org`
Web:	`http://tug.org/consultants.html`
	`http://tug.org/TUGboat/advertising.html`

The latest TEX Collection (`http://tug.org/texcollection`) software distribution was produced in November 2005.

It includes four major components, as follows:



TEX Collection 2005

proTEXt
TEX for MS Windows
based on MiKTEX

MacTEX
TEX for Mac OS X
based on gwTEX

TEX Friendly Zone
NEED A TUG?
http://www.tug.org/

DVD
November 2005

dante e.V.
Postfach 10 18 40
69008 Heidelberg
dante@dante.de
www.dante.de

TEX USERS GROUP
www.tug.org

CTAN
Comprehensive TEX
Archive Network

TEX Live
TEX for Linux/Unix,
Mac OS X, and MS Windows

LEHMANNS FACHBUCHHANDLUNG

Editors: Thomas Feuerstack (proTEXt) • Herbert Schulz (MacTEX)
Manfred Lotz (CTAN) • Sebastian Rahtz, Karl Berry (TEX Live)

- **TEX Live** (`http://tug.org/texlive`): a comprehensive TEX system that can run directly from DVD or installed on hard disk. It includes support for most Unix architectures, including GNU/Linux and Mac OS X, and for Windows.
- **proTEXt** (`http://tug.org/protext`): an easy to install TEX system for Windows. It is based on MiKTEX, with additional tools.
- **MacTEX** (`http://tug.org/mactex`): an easy to install TEX system for Mac OS X. It is based on gwTEX, with TeXShop, XƎTEX, and Excalibur included. Other utilities are also available.
- **CTAN** (`http://www.ctan.org`): the Comprehensive TEX Archive Network, a set of servers worldwide making TEX software publicly available. This DVD contains a snapshot of the German CTAN node, `dante.ctan.org`.

The collection is available by joining TUG (see membership application in this issue, or `http://tug.org/join.html`), or by separate purchase from the TUG store (`http://tug.org/store`).

Most other TEX user groups also make it available to their members (`http://tug.org/usergroups.html`).

EuroTEX 2006: A Hungarian TEX Rhapsody

Announcement and Call for Papers

The 16th EuroTEX meeting, "A Hungarian TEX Rhapsody", will be held in Hungary, between July 6 and 9, 2006. MATEX (the Hungarian TEX User Group) together with the University of Debrecen have committed to undertake the conference affairs, and now announce the call for papers. This will be the first international TEX conference held in Hungary.

For more information about EuroTEX 2006, please visit `http://www.matexhu.org`.

Dates

- March 1, 2006 — Deadline for abstracts of presentations; email `eurotex2006@matexhu.org`.
- June 1, 2006 — Deadline for preprints of papers, for distribution at the conference.
- July 5–8, 2006 — Conference.
- August 15, 2006 — Deadline for final versions of papers; the proceedings will be published as an issue of *TUGboat*.

Topics

Topics include but are not limited to:

- TEX and so many friends, for automated typesetting
- Typography (digital or otherwise)
- Font design and technologies
- Publishing (electronic or otherwise)
- (Re)discovery of the Hungarian book tradition

Location

The place of the conference will be either Gödöllő or Debrecen.

Gödöllő is a nice town near Budapest. Debrecen is a town of universities known as the Calvinist Rome. It is near the biggest Hungarian National Park, Hortobágy, and a famous spa in Hajdúszoboszló.

There will be free time for you to have the opportunity to taste several types of Hungarian wines, and get to know tasty special Hungarian dishes. Hungary is a sunny country during summer, an ideal place for making excursions. There are several cultural programs in both towns, including jazz and classical music festivals, exhibitions and performances. And we especially invite you to bring instruments to create our own festival!

[redacted]

TUGBOAT Volume 26 (2005), No. 2 TUG 2005 Conference Proceedings

Table of Contents
(ordered by difficulty)

TUGboat

Volume 26, Number 2 / 2005
TUG 2005 Conference Proceedings

TUGBOAT Volume 26 (2005), No. 2 TUG 2005 Conference Proceedings

Table of Contents
(ordered by difficulty)