TUGboat

Volume 24, Number 3 / 2003
EuroTEX 2003 Proceedings

TEX Users Group

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

Memberships and Subscriptions

2004 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.

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

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/`

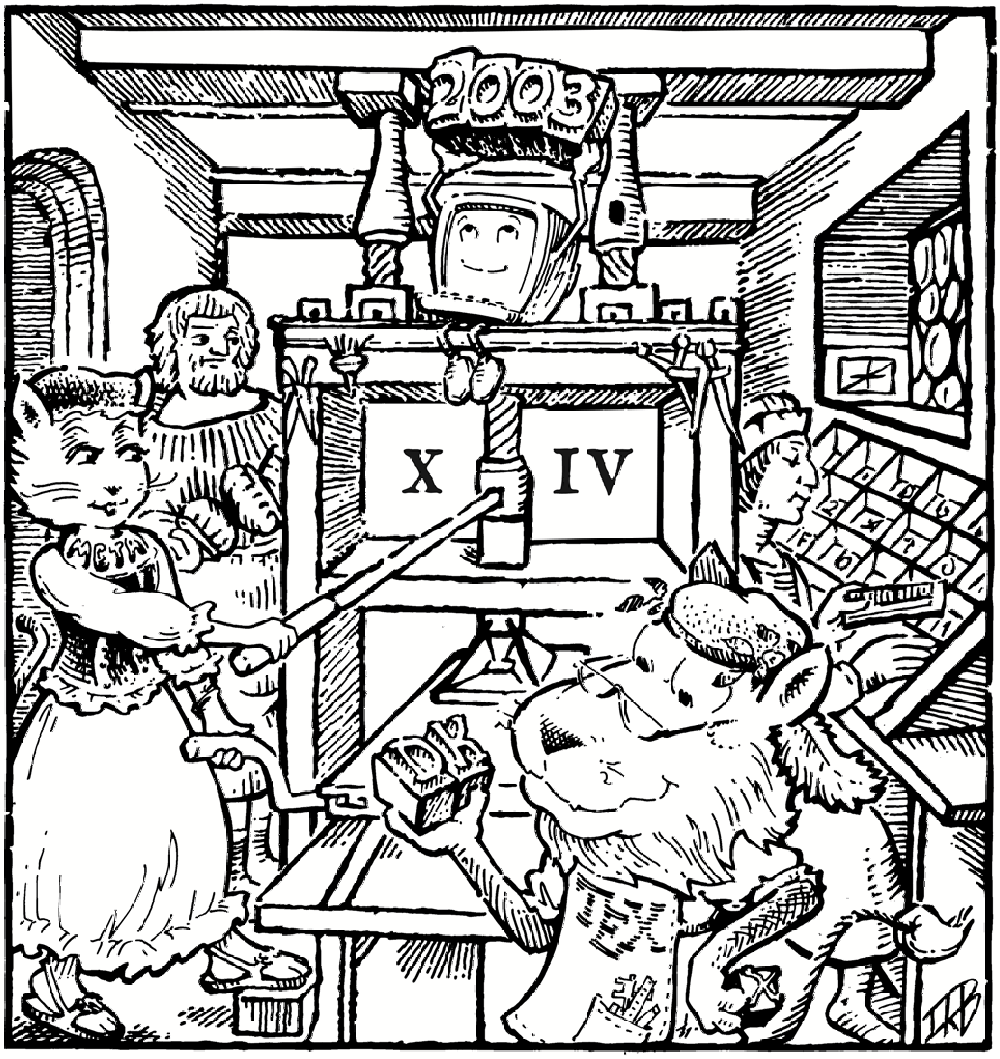
Problems not resolved?

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

[printing date: February 2005]

TUGBOAT

The Communications of the TEX Users Group



Volume 24, Number 3, 2003
EuroTEX 2003 Proceedings

EuroTEX 2003 Proceedings

Fourteenth European TEX Conference
Brest, Brittany, France
June 14–17, 2003

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGboat Editor Barbara Beeton
Proceedings Editor Yannis Haralambous

Volume 24, Number 3 • 2003
Portland • Oregon • U.S.A.

TUGboat

Volume 24, Number 3 / 2003
EuroTEX 2003 Proceedings

EuroTEX 2003 Proceedings

Fourteenth European TEX Conference
Brest, Brittany, France
June 14–17, 2003

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGboat Editor Barbara Beeton
Proceedings Editor Yannis Haralambous

Volume 24, Number 3 • 2003
Portland • Oregon • U.S.A.

The EuroTEX 2003 Conference

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
Yannis.Haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

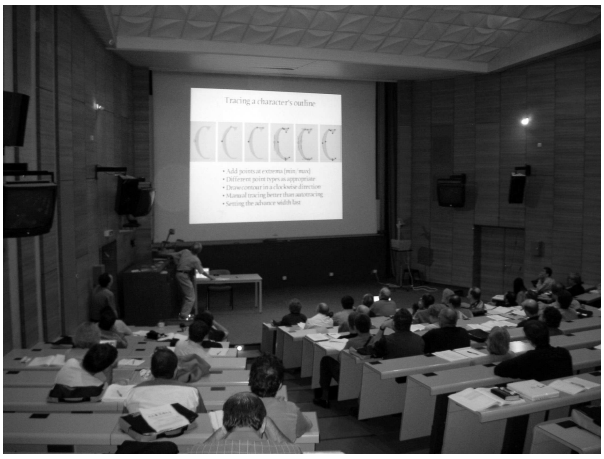
Who, Where, When

The Fourteenth European TEX Conference [1] was organized by the Computer Science Department of the "*École Nationale Supérieure des Télécommunications de Bretagne*" [2], a "grande école" (= French engineering school) located in Brest, Brittany (in western France). Forty-four talks were held over four days, starting on June 24th, 2003. More than 125 people attended the conference, coming from places as remote and diverse as Russia, Hong Kong, Australia, India, South Africa, Brazil, California, Canada, and of course... several European countries. Nevertheless we sadly missed most of the TEX & TUG veterans (Barbara Beeton, Karl Berry, Alan Hoenig, Mimi Burbank, Patricia Monohon, and many more), probably because of the TUG meeting in Hawaii held very shortly after Brest.

Topics

Academic conference or TEX users meeting? This is the usual dilemma when dealing with TEX, which brings together people doing academic research as well as practitioners, not to mention people simply doing it for fun. We tried to cover all aspects. The theme of this conference was "Back to Typography", intended as a reaction to previous conference themes dealing mainly with advanced technological aspects of TEX, like "TEX and the WWW", or "TEX in the era of XML and Unicode" that followed in 2004. The idea was to focus on what makes TEX unique: Don Knuth's (and his followers) passion for fine typography.

This thematic orientation gaved us the opportunity to welcome contributions outside the strict scope of TEX & friends. In fact this may have been the first TEX conference at which there were speakers who had never heard of TEX, and discovered it during the conference. Many papers were submitted in MS Word format, also a very unusual fact for a TEX conference.



(Taken by Tereza Haralambous)

Overview of the Talks

Although not clearly labeled as such, we tried to organize sessions by gathering talks in similar areas of interest. Day 1 was a very TEXnical day, with talks on present or future extensions of TEX ("superglue", Omega, ConTEXt, advanced output routines, font management, etc.). Not surprisingly some talks also dealt with XML.

The morning of Day 2 was a kind of continuation of Day 1, featuring talks on various topics around TEX. In the afternoon we switched to completely different subjects, namely Arabic calligraphy, semiology and history of typography, intertextuality. This was the part of the conference which was the farthest from TEX proper and the closest to the humanities.

Day 3 was also split in two parts. In the morning we dealt with two main issues: bibliography (Bibulus, mlBIBTEX, BIBTEX++) and METAFONT extension projects. In the afternoon once again we had "non-TEX" talks, but this time more technical and related to fonts (in general), history of typography and layout, collaborative DTP, and the like.

Day 4 started as a "font day", since more than half of the talks dealt with fonts, their history, technologies and applications. Only at the end of the day did the topic shift to multilingual support and critical editions.

(Taken by Tereza Haralambous)

Languages

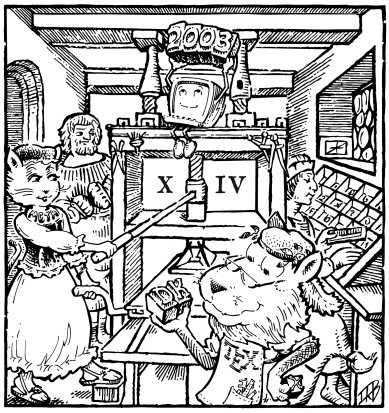
As was the case for the Parisian EuroTEX 1991, many talks were held in French. Fortunately we had a simultaneous translation team, made up entirely of colleagues from the Language Department of ENST Bretagne: Janet Ormrod, Mary Gravot and Patrick McLaughlin. They did an heroic job, not only translating at high speed but even explaining and commenting when necessary. We are extremely grateful to them!

Unlike Paris '91, the Brestian EuroTEX proceedings contain a significant number of papers written in French (almost a fourth of the papers, more than a third when counting pages). This may seem irritating to the average English-speaking reader, but let us not forget that even a mere fifty years ago it was quite common for papers in important scientific journals, even American ones, to be written in the author's native language, be it French, German, Italian or Spanish. And in the humanities this is still the case today.

Logo and (Typo)graphical Aspects

The TEX community likes tradition, and one of most beautiful traditions for the iconography of conferences (also book covers, t-shirts, mugs and whatever that can carry a picture) is to use drawings by Duane Bibby [3], showing the TEX lion, METAFONT lioness and their small computer companion in the context of the conference. For Brest '03 we could dress the lion as a sailor or a pirate (Brest is the favourite harbour for French pirate stories), or as an Obélix-like Celt carrying menhirs. We preferred to stick to the typography theme: the lion and lioness have become craftsmen in an 16th century printshop. The lion holds two character blocks in his right hand (the letters "DK": initials well-known to every TEX user) and one block in his left hand (the Greek letter "χ"). One of the most beautiful Bibby drawings!

Our fearless translators at the task (taken by Tereza Haralambous)



Although the conference was held in France, the visual style of its proceedings is rather British. The banner "Back to Typography" is typeset in Hoefler's *English Textura* [4] (from the collection "Historical Allsorts"), a font inspired by Caxton's blackletter, which brings us back not only to typography, but to the very origins of British printing:

Back to Typography

The font used for the text is the result of a monumental project of typeface digitization of William Caslon's historical collection of typefaces: Justin Howes has drawn 14 (!) different optical sizes, ranging from 8 to 96 points, in roman, italics and small caps. This font set is called *Founders Caslon* [5], and Justin has kindly allowed us to use it for this volume. Typewriter text and mathematical formulas are set in Computer Modern.

Social Events

In the late afternoon of the first conference day, the bookstore *Dialogues* [6], one of biggest bookstores in France, invited us for an "apéritif" cider session. For a few hours the bookstore was open only to conference attendees, allowing them to browse among hundreds of thousands of books.

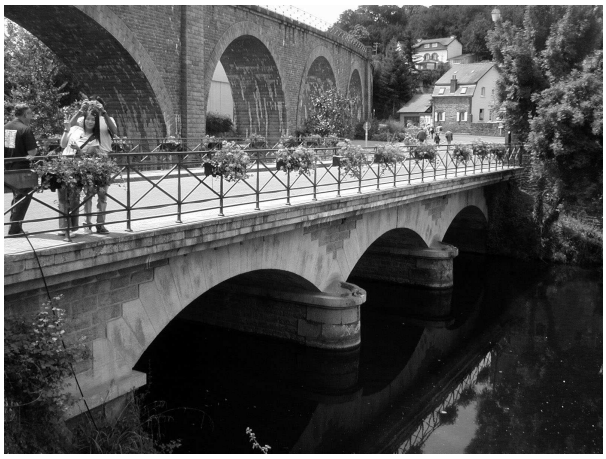
On day 2 we had our gala evening at a restaurant outside town. Before dinner we had a Breton folk dance session, where attendees had the chance to learn Breton dances from members of the local folk dance group in traditional costumes. Breton music was played by the *Kazimodal* [7] band, 2/3's of which were members of the conference programme committee. After dinner the music switched to rhythm & blues, also performed by colleagues at ENST Bretagne: the *Tom Chicago* band [8].

On Saturday, June 28th, we had our big excursion: the attraction was *La Vapeur du Trieux* [9], a train connecting Paimpol to Pontrieux, hauled by a 120-ton steam locomotive, built in Germany in 1912. Pontrieux is a very beautiful Breton village, and the weather was perfect — as we planned it.

Some heavy dancing going on (taken by Tereza Haralambous)

La Vapeur du Trieux (taken by Dag Langmyhr)

At *Dialogues* bookstore (taken by Dag Langmyhr)



A beautiful bridge in Pontrieux (taken by Dimitri Marakov)

Gratitude

A conference is a collective effort and many volunteers participated in this one, colleagues, students, and others. But a few people worked hard for a long time before the conference even started, and these people deserve a special thanks: Anne-Marie L'Hostis and Odile Ély managed and organized the food, dormitories and hotels, transportation, gala, excursion, t-shirts and mugs, registrations, financial issues, and so much more. Without them the conference would have been a failure and the author would have had a nervous breakdown.

Many thanks also to Andrzej Borzyszkowski and GUST for organizing the EuroTEX bus, a 5,000 km Warsaw-Brest roundtrip with stops at several German towns and Paris.

My dear wife, Tereza Haralambous, was the "official" conference photographer, but many attendees also took pictures. We have gathered more than 500 of them on the conference Web site [10].

The conference received funds from public institutions such as the Conseil général du Finistère [11], the Région Bretagne [12], the Communauté Urbaine de Brest [13] (afterwards renamed Brest Métropole Océane), and the Groupe des Écoles de Télécommunications [14]. It also received donations from TUG, GUT and DANTE. Air France and SNCF offered a discount on airway and railway tickets. Our warmest thanks go to all of them.

In Memoriam

The author could not finish this introduction to the EuroTEX 2003 proceedings without a thought for his father, Athanassios-Diomidis Haralambous, who passed away a few days before the conference. Besides being a loving father, he was both a scholar and a scientist, and a passionate book lover. As the eternal song says, *Venit mors velociter/ Rapit nos atrociter/ Nemini parcetur*.

The registration desk volunteers: Selma Matougui, Frédéric Miras, Kamal Gakhar, Anne-Marie L'Hostis (taken by Tereza Haralambous)

References

[1] http://omega.enstb.org/eurotex2003
[2] http://www.enst-bretagne.fr
[3] http://tug.org/tug2003/bulletin/bibby
[4] http://www.typography.com/catalog/historicals/more3.html
[5] http://www.hwcaslon.com/
[6] http://www.dialoguesenligne.com/
[7] http://kazimodal.trad.org
[8] http://tomchicago.free.fr
[9] http://www.vapeurdutrieux.com/
[10] http://omega.enstb.org/eurotex2003/photos
[11] http://www.cg29.fr/
[12] http://www.region-bretagne.fr/CRB
[13] http://www.mairie-brest.fr/cub/default.htm
[14] http://www.get-telecom.fr/fr_accueil.html

Our sponsors

Schedule — Programme horaire

Tuesday, June 24th, morning session

9:00 AM: Opening speech by Bernard Ayrault, Director of ENST Bretagne	*9h00 : Discours d'ouverture par Bernard Ayrault, directeur de l'ENST Bretagne*
Talks starting at 9:30 AM	*Début des exposés : 9h30*
Frank Mittelbach (EDS, Mainz) & Chris Rowley (Open University, London)	Superglue *Superglue*
Hans Hagen (Pragma ADE, Hasselt)	Can TEX beat DTP? *TEX peut-il battre la PAO ?*
Coffee break 10:30–11:00 AM	*Pause café : 10h30–11h*
John Plaice (UNSW, Sydney) & Yannis Haralambous (ENST Bretagne, Brest)	Generating multiple outputs from Omega *Génération de sorties multiples à partir d'Omega*
Gábor Bella & Anish Mehta (ENST Bretagne, Brest)	Using Omega and odvips with TrueType and OpenType fonts *Utilisation d'Omega et odvips avec des fontes TrueType et OpenType*
Yannis Haralambous (ENST Bretagne) & John Plaice (UNSW, Sydney)	XLATEX, a DTD/Schema which is very close to LATEX *XLATEX, une DTD ou un schéma, très proches de LATEX*
Lunch 12:30 AM-2:00 PM	*Pause déjeuner : 12h30–14h*

Tuesday, June 24th, afternoon session

Starting at 2:00 PM	*Début des exposés : 14h00*
David Kastrup (Bochum)	Output routine requirements for advanced typesetting tasks *Prérequis de routine de sortie pour tâches de composition avancées*
Simon Pepping (Elsevier Science, Amsterdam)	DocBook in ConTEXt, a ConTEXt XML mapping for DocBook documents *DocBook dans ConTEXt, une correspondance entre XML et ConTEXt pour les document DocBook*
José Grimm (INRIA, Nice)	Tralics, a LATEX to XML translator *Tralics, un traducteur de LATEX à XML*
Coffee break 3:30–4:00 PM	*Pause café : 15h30–16h*
Petr Sojka (Masaryk University, Brno) & Petr Olsák (CVUT, Prague)	The font management with OFS *Gestion des fontes avec OFS*
Vecsei Balázs (BME, Budapest)	Description of knowledge of mathematical programs with TEX and XML *Description de la connaissance dans les logiciels mathématiques avec TEX et XML*
Talks ending at 5:00 AM	*Fin des exposés : 17h00*
Reception organized by and held at the *Dialogues* Bookstore	*Pot organisé par, et tenu aux locaux de la librairie* Dialogues

Wednesday, June 25th, morning session

Starting at 9:00 AM	*Début des exposés : 9h00*
Marie-Louise Chaix (EDP Sciences, Paris) & Fabrice Popineau (Supélec, Metz)	The XEmTeX Project *Le projet XEmTeX*
Jérôme Laurens (Université de Bourgogne)	iTeXMac, integrated TeX software for Mac OS X *iTeXMac, un logiciel TeX intégré pour Mac OS X*
Azzedine Lazrek (University of Marrakech)	CurExt, a tool for typesetting variable-sized curved symbols *CurExt, un outil pour la composition de symboles curvilignes de taille variable*
Coffee break 10:30–11:00 AM	*Pause café : 10h30–11h*
Karel Horak (Academy of Sciences, Prague)	Geometric diversions with TeX, METAFONT and METAPOST *Jeux géométriques avec TeX, METAFONT et METAPOST*
Péter Szabó (BME, Budapest)	Inserting external figures with GraphicP *Insertion de figures externes avec GraphicP*
Petr Olsák (CVUT, Prague)	The second version of encTeX: UTF-8 support *encTeX version 2 : le soutien d'UTF-8*
Lunch 12:30 AM-2:00 PM	*Pause déjeuner : 12h30–14h*

Wednesday, June 25th, afternoon session

Starting at 2:00 PM	*Début des exposés : 14h00*
Thomas Milo (DecoType, The Netherlands)	Ali Baba and the 40 Unicode characters — towards the ideal Arabic working environment *Ali-Baba et les 40 caractères Unicode — un pas vers l'environnement de travail arabe idéal*
Vlad Atanasiu (EPHE, Paris)	Simulating ancient scripts: The role of contextual allographs *Simulation d'écritures anciennes : le rôle des allographes contextuels*
Emmanuël Souchier (ENST, Paris)	"Typography as a servant", articulation of a certain praxis and discourse [*Talk held in French, with simultaneous translation into English*] *La «typographie servante» — articulation d'une pratique et d'un discours*
Coffee break 3:30–4:00 PM	*Pause café : 15h30–16h*
Ghassan Mourad (Paris IV — Sorbonne)	New reading and writing signs for electronic documents: Example of the pointed index finger and hypertext signs [*Talk held in French, with simultaneous translation into English*] *Nouveaux signes de lecture et d'écriture pour les documents électroniques. Exemple de l'index pointé et des signes de l'hypertexte*
Ioannis Kanellos (ENST Bretagne, Brest)	Intertextuality and digital typography *Intertextualité et typographie numérique*
Talks ending at 5:00 AM	*Fin des exposés : 17h00*
Banquet at *Moulin de Traon Lez*	*Dîner de gala au* Moulin de Traon Lez

Thursday, June 26th, morning session

Starting at 9:00 AM	*Début des exposés : 9h00*
Thomas Widmann (Glasgow)	Bibulus — a Perl/XML replacement for BibTeX *Bibulus — un remplacement de BibTeXécrit en Perl/XML*
Jean-Michel Hufflen (Université de Franch-Comté, Besançon)	European bibliography styles and MlBibTeX *Styles bibliographiques européens et MlBibTeX*
Coffee break 10:00–10:30 AM	*Pause café : 10h00–10h30*
Ronan Keryell (ENST Bretagne, Brest)	BibTeX++: Towards higher-order BibTeXing *BibTeX++ : vers une utilisation de BibTeX d'ordre supérieur*
Serge Vakulenko (Cronyx Engineering, Moscow)	The Metatype project: creating TrueType fonts based on METAFONT *Le projet Metatype : création de fontes TrueType basée sur METAFONT*
George Williams	Font creation with PfaEdit *Création de fontes sous PfaEdit*
Thierry Stoehr (AFUL, France)	Eight minutes for Knoppix *Huit minutes pour la Knoppix*
Lunch 12:38 AM-2:00 PM	*Pause déjeuner : 12h30–14h*

Thursday, June 26th, afternoon session

Starting at 2:00 PM	*Début des exposés : 14h00*
Luc Devroye (McGill University, Montréal)	Formatting font formats *Formation de formats de fontes*
Isabelle Turcan (Université de Lyon)	The digitization of collections of ancient books: ancient typography [*Talk held in French, with simultaneous translation into English*] *Numérisation de collections de livres anciens : typographie ancienne*
Ghassan Mourad (Paris IV — Sorbonne)	Comments on the graphical — and functional — origins of the comma [*Talk held in French, with simultaneous translation into English*] *Quelques constatations sur l'origine graphique — et fonctionnelle — de la virgule*
Coffee break 3:30–4:00 PM	*Pause café : 15h30–16h*
Michel Cacouros (EPHE, Paris)	Could [and should] digital books benefit from page layout practices previous to the 18th century? [*Talk held in French, with simultaneous translation into English*] *Le livre électronique pourrait-il [et devrait-il] bénéficier des pratiques de mise en page antérieures au 18e siècle ?*
Jacques André (IRISA, Rennes)	The Cassetin project — encoding of ancient types in Unicode [*Talk held in French, with simultaneous translation into English*] *Le projet Cassetin : codage des types anciens en Unicode*
Yves Maniette (UNESP, São Paulo)	Automated system for co-authoring of books *Système automatisé de co-rédaction de livres*
Talks ending at 5:30 AM	*Fin des exposés : 17h30*

Friday, June 27th, morning session

Starting at 9:00 AM	*Début des exposés : 9h00*
Christian Paput (Imprimerie Nationale, Paris)	French typographic patrimony, conservation and teaching [*Talk held in French, with simultaneous translation into English*] *Présentation du Cabinet des poinçons, son contenu et ses activités*
Gerry Leonidas (University of Reading)	Teaching and learning typeface design: early thoughts on an agenda for design education in a digital environment *Enseigner et apprendre la création de fontes : réflexions en avant-première d'un agenda d'éducation au design dans un environnement numérique*
Gyöngyi Bujdosó (University of Debrecen)	About some new types *Sur certains caractères typographiques nouveaux*
Coffee break 10:30–11:00 AM	*Pause café : 10h30–11h*
Jef Tombeur (Paris)	"Schoolbook fonts" and "handwriting typefaces," from decorative to functional fonts for the teaching of reading *«Fontes scolaires» et «manuscrites», du décoratif aux fontes fonctionnelles de l'enseignement de la lecture*
Oliver Corff (FU Berlin)	The source and the legacy: The MF→Perl translation project *La source et le legs : le projet de traduction de MF en Perl*
Boguslaw Jackowski (Gdansk), Janusz M. Nowacki (Grudziadz), Piotr Strzelczyk (Sopot, Poland)	Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating *Programmation de fontes PostScript de type 1 en utilisant MetaType1 : vérification, optimisation, création*
Lunch 12:30 AM-2:00 PM	*Pause déjeuner : 12h30–14h*

Friday, June 27th, afternoon session

Starting at 2:00 PM	*Début des exposés : 14h00*
Sivan Toledo (University of Tel Aviv) & Zvika Rosenberg (Masterfont Ltd., Tel Aviv)	Experience with OpenType font production *Expériences de production de fontes OpenType*
Candy L.K. Yiu, Kelvin C. F. Ng, Wai Wong (Hong Kong Baptist University)	Typesetting rare Chinese characters in LaTeX *Composition de caractères chinois rares avec LaTeX*
Primoz Peterlin (University of Ljubljana)	The free UCS outline fonts project — an attempt to create a global font *Le projet libre de fonte vectorielle UCS — une tentative de créer une fonte globale*
Coffee break 3:30–4:00 PM	*Pause café : 15h30–16h*
Jean-Pierre Sutto (Universitá di Pisa)	The use of MauroTeX in building the critical edition of Francesco Maurolico's mathematical oeuvre [*Talk held in French, with simultaneous translation into English*] *L'utilisation de MauroTeX pour la construction de l'édition critique des Œuvres mathématiques de Francesco Maurolico*
B.V. Venkata Krishna Sastry (Hindu University of America, Orlando)	Enhanced font features for the future multilingual digital typography with sound-script-language attribute integration *Fonctionnalités de fontes avancées pour la typographie numérique multilingue de l'avenir avec intégration d'attributs son-écriture-langue*
Apostolos Syropoulos (Xanthi)	Replicating archaic documents: A typographic challenge *Réplication de document archaïques : un défi typographique*
Talks ending at 5:30 AM	*Fin des exposés : 17h30*

Participants at EuroTEX 2003
June 24–27, 2003
ENST Bretagne, Brest

Jacques André
Vlad Atanasiu
Alain Aubord
Marwan Auger
Paul Bartholdi
Kenneth R. Beesley
Gábor Bella
Javier Bezos-Lopez
Andrzej Borzyszkowski
Thierry Bouche
Benjamin Buffereau
Gyöngi Bujdosó
Michel Cacouros
Lance Carnes
Kevin Carnes
Brian Carnes
Marie-Louise Chaix
Kaja Christiansen
Oliver Corff
Fabien Dagnat
Benoit Deckmyn
Christine Detig
Émilie Devriendt
Luc Devroye
Luzia Dietsche
Jean-Marc Dumont
Bouabid El Ouahidi
Daniel Flipo
Pierre Fournier
Kamal Gakhar
Alexandre Gaudeul
Paul Geffroy
Jean-Marie Gilliot
Hans Ginzel
Michel Girard
Pavel Grams
Steve Grathwohl
José Grimm
Jean-Philippe Guérard
Michael Anthony Guravage
Hans Hagen
Yannis Haralambous
Klaus Höppner
Henning Heinze
Yvon Henel
Marcel Herbst
Morten Høgholm
Karel Horák
Jean-Michel Hufflen
Johannes Idsol
Bogusław Jackowski
Marius Jøhndal
Jean-Paul Jorda
Palle Jorgensen
Ioannis Kanellos
David Kastrup
Ronan Kuryell
Thomas Koch
Reinhard Kotucha
Johannes Küster
Dag Langmyhr
Jérome Laurens
Azzeddine Lazrek
Thomas Le Bras
Jacques Lefrère
Philippe Lanca
Gerry Leonidas
Bogusław Lichonski
Knut Lickert
Thomas Lotze
Lars Madsen
Yves Maniette
Dimitry Marakov
Stephane Marchand
Selma Matougui
Anish Mehta
Thomas Milo
Frédéric Miras
Frank Mittelbach
Jean-Luc Moan
Ghassan Mourad
Joël Mouric
Michel Mouric
Thomas Neumann
Richard Nickalls
Johann Nonat
Janusz Nowacki
Robert Ogor
Petr Olšák
Christophe Osswald
Gérald Ouvradou
Christian Paput
Günter Partosch
Elliott Pearl
Benoit Peccatte
Simon Pepping
Primoz Peterlin
Karel Piska
John Plaice
Fabrice Popineau
Ghislain Putois
Bernd Raichle
Christian Rossi
Christopher Rowley
Venaktakrishna Sastry
Jalil Hassan Sayed
Volker Schaa
Hilmat Schlegel
Joachim Schrod
Martin Schröder
Raymond Seroul
Gagan Sharma
Tadeush Sheibak
Karel Skoupy
Petr Sojka
Emmanuël Souchier
Thierry Stoehr
Jean-Pierre Sutto
Chris Swanepoel
Apostolos Syropoulos
Péter Szabó
Sivan Toledo
Sigitas Tolusis
Jef Tombeur
Isabelle Turcan
Benoit Vaillant
Serge Vakulenko
Balázs Vecsei
Gérard Verroust
Ulrik Vieth
Zofia Walczak
Dorothea Wand
Thomas Widmann
George Williams
Wai Wong
Candy Yiu

French Typographic Patrimony, Conservation and Teaching

Christian Paput
Cabinet des Poinçons
Imprimerie Nationale
c.paput@libertysurf.fr
http://www.imprimerienationale.fr

Abstract

Presentation of the Conservatory of Punches, its content and its activities. Punchcutting today, its usefulness, its future, its developments, for the "Imprimerie nationale" as well as in the framework of the trade of typography. Practical and research possibilities of the different elements that compose its collections. What can be done today in hopes of saving this patrimony and the knowledge that resides there.

Résumé

La gravure du poinçon typographique aujourd'hui, son utilité, son avenir, ses développements, à l'Imprimerie nationale ainsi que les pratiques possibles dans le cadre privé de l'exercice des métiers de la typographie. Les possibilités d'utilisation ou de recherche des différentes pièces qui composent ses collections. Comment peut-on aujourd'hui espérer sauver ce patrimoine et les savoir-faire qui s'y rattachent.

Introduction

The Conservatory of Punches is the nucleus of the French National Printing House (*Imprimerie nationale* — IN). There are two punchcutters at the IN, Nelly Gable and myself.

This punchcutting workshop (creation, restoration) is also the place where the IN's engraving collections are held. The collection consists of over 500,000 items; a simplified inventory:

- 230,000 steel punches (the earliest examples date back to François 1st) employed to print our own IN publications;
- 28,000 steel engraved die-stamps (for medals);
- 14,000 steel punches for printing of musical scores from engraved plates;
- 224,000 Chinese wood-engraved ideograms;
- 15,000 wood-engraved types for poster printing;
- 1,300 wood-engraved blocks for book illustration;
- 3,000 copper-engraved plates for book illustration;
- and 2,500 gold-blocking tools.

In 1946, Raymond Blanchot, director of the IN, had the typographic punch collection listed as a "Historical Monument", and Christian Paput, your modest engraver servant to the Conservatory of Punches, had this classification extended in 1994 to all of the punches and engraved material found since 1946, including all the engraved coins belonging to the IN.

The classification "Historical Monument" by the French Ministry of the Culture registers a collection or a work, ensures its protection and security, and prevents against any possible sale of all or any part of this collection.

The oldest punches at the Conservatory

The collection of western typographic punches is well known through the famous lineage of Garamont, Grandjean, Luce, Didot, Marcellin-Legrand, Jaugeon and Gauthier. But many other lesser-known characters are also in use at the IN, for example, calligraphic character punches of "ronde", "coulée" and "bâtarde", cartography signs, music signs, and chemical and mathematical symbols. The Conservatory of Punches houses those as well as punches by Jacquemin, a certain number of Bodoni punches, some Gothic punches engraved for Arthur Christian, director in 1906, which bear his name, etc.

This general listing of the material, which does not claim to be exhaustive, is almost complete when one considers the Oriental collection belonging to the Conservatory of Punches, including the first wood-engraved Chinese types created under the direction of Fourmont, on the order of the Regent between 1723 to 1730, as well as characters as varied as Hebrew, Arabic, Telugu, Nagari, Cuneiform, etc. These characters, mainly engraved during the course of the 19th century, constitute a treasure whose value cannot be calculated, relating to a wide range of wholly different cultures. Today, seventy languages are represented in one hundred different forms in this collection.

Since the 1980s and 1990s, there have been many developments.

First, the restitution of the last remaining punches of the Peignot collection that had been bequeathed to the Conservatory of Punches by the Haas Switzerland Typefoundry. Subsequently, Neufville Typefoundry Barcelona donated types to the Conservatory of Punches. Punches of the company Plon-Nourry have also been recovered; these are essentially "decorative vignettes". This was followed by the purchase of the Tanturri Workshop, which has enriched our collections with punches of music, as well as with the complete set of tools required for this type of engraving and printing.

Various productions of the French National Printing House, particularly of the Conservatory

A collection of books has been published by the IN. All are composed by hand from metal types.

A workshop is supported, specializing in producing artists' books and original prints, and also catering to the needs of other publishers, in the fields of lithography, etching, engraving and collotype. It is also in this workshop that letterpress printing is practised.

This is why it is the mission of the Conservatory of Punches to preserve typographic punches in perfect condition, suitable for supplying characters for printing.

In practical terms, the composing workshop orders its supply of characters from the typefoundry. The foundry casts type according to the needs of the compositors using the matrices in its extensive type library, as long as the condition of the actual matrix permits. Otherwise the request goes back through the system of production to the Conservatory of Punches which supplies the original punch necessary for the restriking of the defective matrix. Where a punch is damaged, it is completely recut, identical to the original, using the time-honoured methods passed down to us by our forebears. This is the restoration sector of our activity.

The Conservatory of Punches today pursues several aims in order to show its worth, and build its archive outside of the employment of these hand-engraved pieces:

- We participate in various exhibitions, in France and abroad, also loaning material. Notably, we have lent punches to the Seville Universal Exhibition as well as for a display accompanying a conference in Athens. Nearer to home, the Didot exhibition allowed us to present a selection of punches, while an exhibition about Lebanon showed a composition of classic Phoenician in lead type. The Print Museum of Lyon organized an exhibition to celebrate the three hundred year anniversary of the first utilization of the "Romain du Roi" and, for this occasion, borrowed a number of historical items from the IN.
- More recently, we have had the opportunity to increase our collections from donations and, whenever possible (we have no special budget for this purpose), we have also bought historical material.
- Study and experimentation (through lack of time) of traditional techniques — such as hand-composing, music engraving, printing and casting — is undertaken whenever possible, to preserve a dying craft.
- We have created and we continue to increase a library of type specimens, thanks to our suppliers and contacts with the trade.
- Passing on of our knowledge of engraving is sometimes dispensed at the Conservatory of Punches through internships and organized visits. Approximately one trainee per year has been received in our workshop since 1989.
- The creative side to our work, apart from the creation of the typeface "Gauthier" in the 1980s, is presented by the engraving of characters missing from ancient fonts; it is especially evident lately through various projects and creations (character "Humane" and expressive vignettes).
- Through my work as lecturer at École Estienne, I pass on my knowledge of the history of handwriting and typography.
- Inventories of our collections have been undertaken to allow the Historical Monument classification. Today these inventories need to be specified qualitatively to be able to be properly exploitable.

In a few lines, I trace the present situation of the Conservatory of Punches. We know that the French National Printing House must move; for this reason, over the last six months we have resumed collating all of the engraving collections in order to reference and prepare them for transportation to a new installation site. The collection should go into a workshop-museum at Choisy-le-Roi in the months to come.

I have obtained for this preparation the services of an ex-student from École Estienne, qualified in engraving and etching. He will assist me until June 2003, then he will stay with the Conservatory of Punches during his training in the art of typographic punchcutting (three years are envisaged). This training is possible thanks to the action of the Ministry of Culture, in the context of the scheme of "Maîtres d'Art", initiated by them, to safeguard crafts.

Thus, a young engraver is allowed the chance to obtain rare skills in the area of typography which he can use in his future profession, which, for him, seems to be the creation of handmade artist's books.

This experience, designed to be self-perpetuating, should permit this craft to continue both within and without the IN.

References

[1] Fred Smeijers, *Counterpunch*, Hyphen Press, 1995.

[2] Christian Paput, *La lettre — La gravure du Poinçon typographique*, édition bilingue Français-Anglais, TVSO éditions, 1998.

[3] Christian Paput, *Vocabulaire des Arts Graphiques, de la Communication, de la PAO, etc.*, TVSO éditions, 1997–2002.

[4] *Le Cabinet des Poinçons*, Imprimerie Nationale, 1963.

[5] *Les Caractères de l'Imprimerie Nationale*, Imprimerie Nationale, 1990.

The Cassetin Project — Towards an Inventory of Ancient Types and the Related Standardised Encoding

Jacques André
Irisa/Inria-Rennes
Campus de Beaulieu
F-35042 Rennes Cedex, France
Jacques.Andre@irisa.fr

Abstract

The Cassetin Project purposes are to inventory all of the types used in Europe for centuries, to standardize their encoding or naming and to propose this list as a Unicode addendum (or at least as a private area).

Résumé

Nous décrivons le projet Cassetin d'inventaire des types utilisés en typographie européenne en vue de la normalisation de leur codage ou de leur nommage, voire de leur intégration dans Unicode (au pire dans une zone privée).

Introduction

Character encodings such as Unicode [15] make a strong difference between characters (abstract linguistic entities) and glyphs (the rendition of characters) and so ignore the "types" effectively used when composing books. Typical examples are ligatures (such as "ct"), abbreviations or vanished characters (such as the old French "ç ę ŋ"). This absence makes it difficult to standardize OCR outputs and quite impossible to get genuine plain text from electronic editions of old books (especially Renaissance or even 18th century ones). Strangely, far more complicated texts, such as medieval manuscripts, are now electronically editable thanks to encoding projects.

We first show how encodings have been applied to medieval manuscripts, then describe the equivalent, and not favorable, situation for old books. We follow with a presentation of the Cassetin project: its aims are to inventory all of the European types, to name them and to propose this list as a Unicode candidate.

Fig. 1: Manuscript of De Ventadour (France, 12th century)

[...] chanter et vint conter et enseigner.
[...]to sing and came singing and teaching.
2-a: Modern translations

[...]chantar et venc cortez et enseingnatez.
2-b: Modern composition

[...] châ-
tar 7 venc cortef 7 ēfeingnatz.
2-c: Diplomatic edition

Fig. 2: Various editions of figure 1 (last two lines of text)

Electronic Editing of Manuscripts

More and more manuscripts are edited on the Web or on CDs: medieval charts, writers' drafts, registers of births, etc. Most of them are only available in image mode. Some are accompanied with the corresponding text, or rather with the corresponding "texts" when the edition is undertaken by humanities scholars. Indeed they are different views of the same text.

Let us take as an example a manuscript of Bernard

```
... ch&an;<br>tar &et7; venc corte&longz; &et7;
&en;&longs;eingn<unclear reason="stain">a</unclear>tz.<br>
```

Fig. 3: TEI encoding of figure 1

GLYPH	MUFI		ISO	UNICODE	
	ENTITY	NAME	ENTITY	CODE	NAME
	&con;	SIGN CON		0254	SMALL OPEN O
	✗	SIGN CROSS		271D	LATIN CROSS
	&ed;	SIGN ED	;	003D	SEMICOLON
	&est;	SIGN EST		F150	HOMOTHETIC
	&etslash;	SIGN ET WITH SLASH			

Fig. 4: Example of MUFI proposal for encoding Nordic medieval abbreviations

de Ventadour[1] and even let us consider just the last two lines of Figure 1. They can be encoded in various ways. In reverse order, we can first translate these lines into modern French or even into modern English (figure 2-a). A student in medieval philology would prefer a form closer to the genuine language, like figure 2-b, where each sign is composed with modern types, without respect to specific handwritten signs. A way[2] closer to the manuscript is to edit the text with types close to the handwritten signs (figure 2-c) and to respect the layout (hyphenation, etc.).

In the end, a researcher needs far more, such as ligatures or signs used, unclear letters, who wrote or annotated the original text and who has translated or encoded it. Many projects have been launched to access electronic versions of medieval manuscripts.[3] They generally use SGML-like tags to indicate either the structure of the document or the actual characters used. Figure 3 shows how our two lines could be encoded.[4] Tags like `&et7;` or `&an;` indicate special characters ("et" and "an") and are like the ones used by Project Charrette at Princeton University.[5] Other projects use other tags (e.g., `&abar;` instead of `&an;`). That is why some new projects, such as MUFI[6] (figure 4), try to inventory all of these signs, to give them names (used as entity names), to propose[7] them as Unicode characters and finally to propose a font offering all of these glyphs.

From now on, thanks to such projects, the various electronic editions of manuscripts should follow a standardized encoding, hence become portable and usable by any researcher.

1. A French troubadour (12th Century). Here is an example of a version of his *Quand vei la laudeta mover...* (When the skylark wings...), a song written in "Provencal" (an Old French dialect).

2. This is close to the so called "diplomatic" version, generally used for modern writers' drafts, that takes care as well of the position (orientation, length, etc.) of lines in pages.

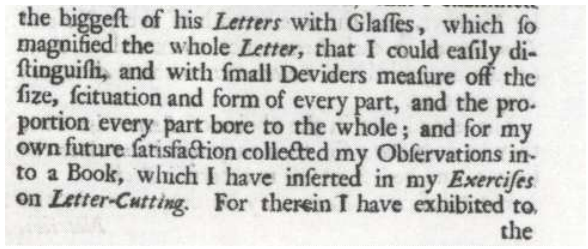
3. Such as Digital Scriptorium (`http://sunsite.berkeley.edu/Scriptorium/`), EAMMS (`http://www.csbsju.edu/hmml/eamms/`), or the TEI based project MASTER (`http://www.cta.dmu.ac.uk/projects/master/`).

4. That form is rich: it can be translated to forms 2.a to 2.c, while the opposite way is generally impossible.

5. The list of entities may be found at `http://www.mshs.univ-poitiers.fr/cescm/lancelot/keys.html`.

6. Medieval Unicode Font Initiative, `http://www.hit.uib.no/mufi/`.

7. See section "Unicode" below for the position of the Unicode Consortium regarding glyphs and characters.


the biggeſt of his *Letters* with Glaſſes, which ſo magnified the whole *Letter*, that I could eaſily diſtinguiſh, and with ſmall Deviders meaſure off the ſize, ſcituation and form of every part, and the proportion every part bore to the whole; and for my own future ſatisfaction collected my Obſervations into a Book, which I have inſerted in my *Exerciſes* on *Letter-Cutting*. For therein I have exhibited to the

the biggest of his *Letters* with glasses, which so magnified the whole *Letter*, that I could easily distinguish, and with small dividers measure off the size, situation and form of every part, and the proportion every part bore to the whole; and for my own future satisfaction collected my observations into a book, which I have inserted in my *Exercises* on *Letter-Cutting*. For therein I have exhibited to [...]

5-a: Modern transliteration

```
the bigge&st; of his <ital>Letters</ital> with
Gla&ss;es , which &longs;o<br>
 magni&fi;&ed the whole <ital>Letter</ital>,
that I could ea&longs;ily
di&hyphen;<br>&stlig;tingui&shlig;, [...]
[...]
&longs;atisfa&ctlig;ion collec&ctlig;ed  [...]
```

5-c: TEI-like encoding

FIG. 5: Extract from Moxon's *Mechanick Exercises* (1683), (volume 2 *Applied to the Art of Printing*) followed by a modern transliteration (5-a) and a TEI-like encoding (5-c)

Electronic Editions of Old Books

Probably because printed characters are thought to be more readable than handwritten ones, quite often digitized books are edited as images and/or as plain text encoded with modern characters. If some other types are used, no attempt is made to standardize their encoding.

Moxon's Mechanick Exercises Let us consider first the famous Moxon's book (1683, a pioneer in the matter of printers' manuals [4]) *Mechanick Exercises* [14]. Figure 5 shows an extract and the corresponding "plain text" version of it: it is a modern version, with modern words, no emphatic caps, no ligatures, etc. As for manuscripts, it is possible to encode this text (figure 5-c) to take care of specific characters (like ct), hyphenation (tag `÷`), etc.

However, many people would like an edition (like figure 6) that is both more legible than TEI-encoding (figure 5-c) and more authentic than a traditional plain text.[8] Alas, if one looks in minute detail at these glyphs, one can see that if ligatures such as "fi," "long s + i," "long s + t" or "ct" are present, other ones are absent, such as

8. Furthermore, figure 5-a does not allow any research on the use of the long s or of old words (e.g., scituation).

the biggeſt of his...
magnified the whole...
diſtinguish, and with ſmall...
ſize, ſcituation...
...
own future ſatisfaſtion...

FIG. 6: Moxon text (figure 5) edited with convenient glyphs[9]

FIG. 7: Fertel lower case (1723)

"sh": it is quite impossible to find a font with all the usual (old) glyphs! It is not a matter of difficulty to design these characters, but simply that designers do not know them!

Note that today OCRs still have problems recognizing old types; however, current research allows one to think that very soon OCRs will be able to recognize Renaissance characters and their ligatures. What will be the output when recognizing the ligature "ct"? A code (which one? nnn_8, $xxxx_{16}$?) or a name? Will it be "ct" (if so, any study on ligatures will be impossible)? Or "`&ctlig;`"? Or "`&ct;`", or "`\ct`", or ...? Standardized naming or encoding is required!

Another example: Fertel's case Figure 7 exhibits the lower case used by Fertel [8, page 12] in the first quarter of the 18th century. Among other special characters (such as ligatures "si", "ssi" or "ct" and typically French characters like "ç" and "œ"), it offers an "ę" (left upper corner) that has nothing to do with the "e ogonek" used in eastern Europe. It is an echo of a former system to write French (by a grammarian and poet, Jean-Antoine de Baïf, 16th century) and this kind of "breve e" was used for less than a century.[10] Obviously, this character is not an isolated example and you can find many characters that are neither in today's encodings nor in font character

9. Here, as in the rest of these proceedings, the *HW Caslon* font is used, designed by Howard Jones.

10. 50 years after Fertel, Diderot's *Encyclopédie* shows the same case, however the "ę" is replaced by an "é" while the genuine box for "é" is empty!

sets. Yet, they are present in foundry specimens, books or even in grammars ...

Unicode, Characters and Glyphs

Unicode [15] makes a strong difference between characters (abstract linguistic entities) and glyphs (a possible physical stylistic representation or rendition of these entities). Very few clever papers give a good explanation of those concepts; let us cite here one by Ken Whistler, the technical director of Unicode [2] and one by a typographer, John Hudson [12]. On the other hand, there are also good papers that say that Unicode made the wrong choice and that characters and glyphs are not so easily different [10, 11]. We would like to add that "types" (with the usual typographic meaning) are neither characters, nor glyphs.[11]

An important point is that the Unicode principle that separates glyphs and characters has been historically violated by another one: Unicode is based on previous encoding systems (proprietary or international standards) where ligatures were present. If Unicode was clean, even the sign "&" should not be there! However we can be suspicious why "long s" and even "ligature st" have been very recently added and not "ligature ct"!

Imagine the dialog:
– "How could I describe[12] Fertel's case (figure 7) and its ę using Unicode?" I ask.
– Answer from Unicode specialist: "Use `latin small letter e with ogonek`, U+0119."
– "No, I say, Fertel's character is not that character, there is the same glyph resemblance as with `latin capital a` and `greek capital alpha`, but they are different characters and Unicode encodes them separately."
– "Why don't you encode this character as letter e and a combining diacritic ogonek?"
– "For it is not an ogonek, rather a kind of breve," I answer.
– "OK," he says, "your ę is a glyph of some `latin small letter with breve`."
I disagree, it's not the same breve as the one used by Fertel in another case: "ĕ", so it's not the same character ... And now, if you look at the alphabet given by the same Baïf, you can see an "`a with raising tail`" that is rather a nasal O (its place in the alphabet is just before the P letter). Let us restart the same dialog ...

Last point: Unicode knows old languages such as the Runes or Ogham. Why should it ignore old European languages and their writing used for centuries?

The Cassetin Project

Being involved in digitization projects such as Fournier's *Manuel typographique*,[13] I am continuously confronted with such problems of coding or naming old[14] characters. Discussions with many people involved in such tasks pushed me recently to undertake a project[15] to inventory these types and try to establish a standardized list of names or ... codes.

Its main aims are:

Inventory of types Prepare an inventory of all types used in texts[16] printed in European[17] languages.

Typical characters are

- Ligatures, such as the ones already quoted here (sh, si, st, ...) and many other ones (like the Hungarian gz ...).
- Vanished characters, such as the "ę," the tailed A, etc.
- Accented characters (like the old Spanish consonants).
- Abbreviations.
- Special characters such as verset and respons (these two are in Unicode, but many other special characters are not).
- Historical typographical characters[18] (that are not already in Unicode) such as raised letters.

This inventory is based on

- Previous studies, such as [3, 4, 5, 7], including Web pages such as Bolton's on cases [6].
- Specimens published by foundries.
- Ancient books.
- The MUFI project for manuscripts!

11. There are many stylistic variants of our "ę"! On the other hand, Unicode speaks about rendition of abstract characters. However, what about the other way: when scanning documents, printed characters exist before the corresponding "abstract" character, they are not only images of abstract characters, they are characters by themselves at an intermediary level between glyphs and linguistic entities.

12. Even if "[t]he Unicode Standard is explicitly *not* aimed at being a system for facsimile representation of text" [2], one may need to quote such a character. Actually, it is not only a Unicode problem!

13. Like Moxon's, a famous 18th century book on type-cutting and typefounding. See [9, 13] and `http://www.irisa.fr/faqtypo/BiViTy`.

14. Old means here before DTP! A typical example is the use, still current in 1950, of the abbreviation "crossed K" that represents the Breton "ker" occurring in many names.

15. Temporarily called CASSETIN: "cassetin" is the French name of case boxes. It can stand for "CASSE Type encodING" ... See also [1].

16. One problem not yet solved: should we consider all types, even the ones used outside of plain text, such as ornaments and rules? I do not think so, however the limits are not yet fixed!

17. This is again an unsolved question: Which languages do we consider? Latin ones? What about Cyrillic, Greek, Hebrew, Arabic, Syriac, etc.? Actually, today it is only a matter of specialists working in this project ...

18. We do not dare to speak about small caps!

Naming and Encoding Each identified type will enter a file with typical glyphs, usages, etc. and a name will be given. This name will be usable as an XML entity (&xxx;), a TEX name (\xxx), an (Adobe or OpenType) glyph name, etc.

Each type will as well be assigned a number: the Unicode number when it exists; if not, a new number that could be an entry in some Unicode *user private area*.

Note that these glyph/character names permit not only standardizing output from OCR, but as well a standardized way to type in these special signs for rendition.

Experimental Font As MUFI does, an existing font should be upgraded to offer all of these, say, glyphs: editing "facsimile" texts like those in figures 5 or 6 should be easier! Again, it's not only a matter of actual glyphs, but rather, to be standardised, a matter of table encoding. In that way, TEX LMs or Ω could be good candidates!

Calendar This project is still a private undertaking, however, I'd like to make it an international project[19] (with, e.g., European Union help).

A quick glance at the already published cases shows that the number of new characters to inventory is not very large and the list should not be as large! So this work should not last years ...

Conclusion

Glyphs or not, characters or not, types belong to a class that is not recognized by Unicode. Historians of books, of languages, etc. do need a standardization of their names, even of their encodings, in such a way that the increasing number of sites offering digitized books can be researched in a portable manner.

References

[1] André J. (2003), "Numérisation et codage des caractères de livres anciens", *Numérisation et patrimoine* (B. Coüasnon éd.), special issue of *Document numérique*, vol. 7, num. 3–4, 2003, pp. 127–142.

[2] Andries P. (2002), "Entretien avec Ken Whistler, directeur technique du consortium Unicode", *Document numérique*, vol. 6, 3-4, pp. 13–49. http://hapax.iquebec.com/hapax/

[3] Barber G. (1969), *French Letterpress Printing. A list of French printing manuals and other texts in French bearing on the technique of letterpress printing*, Occasional publication num. 5, Oxford Bibliographical Society.

[4] Baudin F. (1994), *L'effet Gutenberg*, Éditions du Cercle de la Librairie, Paris.

[5] Bigmore F.C. and Wyman C.W.H. (1978), *A Bibliography of Printing with Notes & Illustrations*, Holland Press Ltd (London) and Oak Knoll Books (USA).

[6] Bolton D. (2002), *Type cases*, The Alembic Press, http://members.aol.com/typecase/

[7] Dreyfus J. (1972), *Type specimen facsimiles*, London.

[8] Fertel M.D. (1723), *La science pratique de l'imprimerie* ..., Saint Omer. Facsimile by Libris editions, 1998.

[9] *Fournier le jeune* (1764 et 1766), *Manuel typographique utile aux gens de lettres et à ceux qui exercent les différentes parties de l'art et de l'imprimerie*, Paris, chez Barbou, 2 tomes.

[10] Haralambous Y. (2000), "Unicode, XML, TEI, Ω and Scholarly Documents", 16th International Unicode Conference, Amsterdam, 24 p.

[11] Haralambous Y. (2002), "Unicode et typographie : un amour impossible", *Document numérique*, vol. 6, n° 3-4, p. 105–137.

[12] Hudson J. (2002), "Unicode, from text to type", *Language Culture Type — International Type design in the Age of Unicode* (Berry ed.), ATyPI/Graphis, pp. 24–44.

[13] Mosley J. and Carter H. (1995), *The Manuel Typographique of Pierre-Simon Fournier le jeune*, 3 volumes, Darmstadt.

[14] Moxon J. (1683), *Mechanik Exercises or the Doctrine of Handy-works*, printed for Joseph Moxon on the Westside of Fleet-ditch, at the Sign of Atlas. Misc. eds. by Davis and Carter (Dover, 1978), De Vinne (1886), Johnson and Gibson (Oxford University Press, 1978), etc. Note that the second volume, *Applied to the Art of Printing*, is more difficult to find.

[15] The Unicode Consortium (2003), *The Unicode Standard, Version* 4.0, Addison-Wesley, ISBN 0-321-18578-1. See also http://www.unicode.org.

19. One goal of this paper is a call to participation! Don't hesitate to mail to the author.

Replicating Archaic Documents: A Typographic Challenge

Apostolos Syropoulos
366, 28th October Str.
GR-671 00 Xanthi, Greece
apostolo@ocean1.ee.duth.gr

Abstract

The replication of archaic documents on paper using modern typographic tools is a quite challenging task. This task is usually broken in two sub-tasks: the design of a font that fully respects all aspects of the handwritten document and the design of software tools (macros for the digital typesetter and/or external routines) that are used to set the archaic document. Here we report on the creation of a font and the accompanying typographic tools for the replication of Epi-Olmec stelae. In addition, based on the experience we gained from our work on the tools for the Epi-Olmec script, we developed a font and similar typographic tools for the replication of the Philokalia book, which we also report.

Résumé

La reproduction de documents archaïques en utilisant des outils typographiques modernes est une tâche ardue. Nous l'avons subdivisée en deux sous-tâches : la création d'une fonte qui respecte tous les aspects du document manuscrit et le développement d'outils logiciels (macros pour le compositeur numérique et/ou routines externes) utilisés pour le document arachaïques. Nous présentons ici la création d'outils typographiques pour les stèles Épi-Olmec. D'autre part, nous appuyant sur l'expérience acquise par notre travail sur les outils de l'écriture Épi-Olmec, nous avons développé une autre série d'outils typographiques pour la reproduction du livre *Philokalia*, outils également présentés ici.

Introduction

Different people view the same things in different ways. So a book or a pamphlet crafted in the pre-typographic era is just a text for some people. For others it is something more—the binding, the material the text was scribed on and the scribing style. We have to admit that we are fascinated by digital typography and writing and its history. Since writing is a system of human intercommunication by means of a set of visible marks with a conventional reference [2], the arrangement of symbols is an essential part of writing. Consequently, we tend to view "documents" as streams of little pieces of art arranged in a particular way on paper, a stone, a pot, etc.

Pre-typographic documents are now available in print. However, they are not readily available in their "original" form, but in a modern version of their "original" form. Thus, it is quite difficult to have copies of the "original" documents. In order to fill this gap, certain publishers produce exact replicas of the "original" documents by scanning the text and printing the images on a type of paper that gives the reader a particular feeling. On the other hand, there are documents that have been carved on stone and so it is not possible to produce replicas of these documents. So one can create a computer font that contains the symbols of the script and thus provide a means to have access to the "original" form of the text. Obviously, the font is not enough. In addition, we need typographic tools that will provide the means of replicating the text using the font. But is it easy to create such fonts and the accompanying typographic tools?

Our answer to the question above is affirmative and here we report the results of our endeavor to create both the font and the typographic tools for two different documents: the Epi-Olmec steles and the Philokalia book. Since we describe the creation of a font for a non-alphabetic script, we start by describing how writing systems are classified. Next, we describe the various steps that were necessary for the fabrication of a font for the Epi-Olmec script (an ancient Mesoamerican logosyllabic script, see Table 1) plus the various accompanying typographic tools. We continue with the presentation of the Philokalia fonts that were fabricated using the procedure developed for the Epi-Olmec font. Although the Greek book entitled *Philokalia* was published in 1782, we present the corresponding font as a demonstration of our technique. We conclude with some remarks concerning our work and our future plans.

Table 1: The Epi-Olmec symbol table.

Classification of Writing Systems

Historically, writing is the only permanent means of intercommunication between humans. Naturally, nowadays other permanent means of intercommunication exist (e.g., tapes, CD, DVD, etc.), but still writing is the primary means to convey ideas, information and knowledge. In general, there are two forms of writing: *lexigraphy* or *glottography* (i.e., visual symbols that make a permanent record of speech) and *semasiography* (i.e., sign systems that communicate information between human beings) [2, 3]. Examples of semasiographic systems include pictographs and articulated symbolic systems (e.g., mathematics and musical notations). There are three categories of lexigraphic writing: *logographic* writing (i.e., symbols represent significant elements of speech such as words or phrases), *phonographic* writing and *logosyllabic* writing (i.e., symbols are either *logograms*, that is symbols that denote words, or *syllabograms*, that is symbols that indicate syllables). Furthermore, there are two forms of phonographic writing: syllabography (i.e., writing with symbols that represent syllables of words) and alphabetic writing (i.e., symbols are not individually pronounceable yet when formed in sequential combinations are able to indicate with surprising accuracy the sounds of spoken language).

Typesetting Epi-Olmec Texts

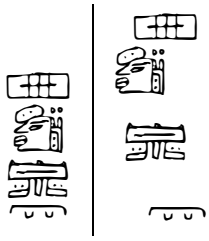
Epi-Olmec is an ancient Mesoamerican logosyllabic script which has been recently deciphered by Terrence Kaufman and John Justeson. A complete description of the

Fig. 1: The la Mojerra stele.

script can be found in [1]. Table 1 shows all the symbols used in the Epi-Olmec script to the best of our knowledge. People of the Mesoamericas used the vigesimal numbering system and the first two rows of the table contain the digits of the system. Note that the Mayas had a designated symbol to indicate zero, but there is no evidence whether zero was known to the Olmec people. Epi-Olmec texts were written in columns either from left to right or from right to left. For example, the text in Figure 1 follows both writing directions. In particular, the left part of text follows the first writing direction and the right part the second writing direction. Note also that the text on Figure 1 is the largest piece of Epi-Olmec text that has been found so far.

Let us now describe how we created the Epi-Olmec font. Since we had no pictures of any Epi-Olmec document we had to resort to [1] to get the shapes of the symbols of the script. For each symbol we generated a bitmap from a screenshot of a magnified image of the symbol. The screen shots were made with the GNU Image Manipulation Program (GIMP). Then, we traced the generated bitmaps with autotrace. This step was necessary in order to create a truly scalable font. However, since the lines of the symbols were not smooth, we had to simplify the lines. For this purpose, we had to resort to a commercial product (namely Adobe Illustrator). With this program we visually manipulated the images of each symbol and generated EPS files. These files were transformed to AI-EPS files with a program called aimaker. The AI-EPS files were the basis for the font that was created with pfaedit. We had to perform a number of different tasks (e.g., reducing the number of control points, correcting the direction of lines, etc.) in order to bring each glyph to a decent form. The result of our work is shown in Table 1.

Naturally, a font is not enough in order to typeset text with it, especially when it comes to the typesetting of text in a "peculiar" writing direction. This particular "peculiar" writing direction dictated the use of Ω as our basis for the design of the typesetting tools that accompany the Epi-Olmec font. Although the reader may expect that we relied on Ω's capabilities to typeset text in various writing directions, still we realized that this capability was not helpful at all. In the two examples that follow, the text on right was typeset using these extra capabilities, while the text on the left uses the conventional capabilities of TEX:



However, we must note that these extra capabilities of Ω yield the expected results in some other cases. This led us to believe that there is some problem with our font, but we have not figured out the problem yet.

Since the Epi-Olmec script is a logosyllagraphy we need some practical way to access the symbols of the script. One solution is the development of an Ω translation process that will map words and "syllables" to the corresponding glyphs of the font. In this way one obtains a natural way for typing in Epi-Olmec texts. In addition, in order to avoid the problem mentioned above, we need a wrapper that will typeset the text vertically. For short texts `\shortstack`s are quite adequate, while for longer texts, one can use a `multicols` environment inside a relatively narrow `minipage`. We have partially implemented this solution, but at the same time we investigate some other ideas, which might produce better results.

Typesetting with the Philokalia Fonts

The Philokalia fonts consist of three fonts: one that contains the normal typeface, one that contains the ligatures

FIG. 2: Text from the original Philokalia book (top) and the text as it is typeset with the Philokalia fonts and LaTeX (bottom).

and one that contains the special ornament characters that decorate the beginning of each chapter. The glyphs were generated from scanned images of the book pages. However, the scans were made in low resolution, which prevented us from doing quick and dirty work. My collegue Yannis Gamvetas spent many hours working on the smoothing of the curves of the letters with Adobe Illustrator.

When this work was finished, we had to create an Ω translation process plus an Ω virtual property list file to handle the numerous ligatures of the font. Another problem was the handling of the ornament glyphs. For this purpose we used the lettrine package, which we felt was quite adequate for this particular task.

Figure 2 shows a little piece of text from the Philokalia book and how LaTeX typeset the same text using the Philokalia fonts. The reader is invited to judge whether the fonts meet our original goal. In addition, the reader is invited to judge whether the resulting fonts produce aesthetically pleasing results.

Conclusions and Future Work

We have presented a procedure by means of which one can create fonts from archaic or old documents. These fonts accompanied by a set of macros can be used to digitally reproduce the original documents. In this way it is possible to replicate old documents in a very efficient and systematic way. In the near future, we plan to release both the fonts and the macros presented here so that people can use them. In addition, we plan to create a font for the script of the famous disk of Phaistos[1] and possibly a font with all the known symbols of Linear A.

Acknowledgements

The author thanks Yannis Gamvetas for his excellent work and the anonymous referee for his helpful comments and suggestions.

References

[1] Terrence Kaufman and John Justeson. Epi-Olmec Hieroglyphic Writing and Texts. Available from `http://www.albany.edu/anthro/maldp/EOTEXTS.pdf`, 2001.

[2] Barry B. Powell. *Homer and the Origin of the Greek alphabet*. Cambridge University Press, Cambridge, U.K., 1991.

[3] Javier Serrano Urcid. *Zapotec Hieroglyphic Writing*. Dumbarton Oaks Public Service, Washington, D.C., U.S.A., 2001.

1. This font has been completed since the time of writing and is available from CTAN in `fonts/phaistos`.

CurExt, Typesetting Variable-Sized Curved Symbols

Azzeddine Lazrek
Department of Computer Science, Faculty of Science,
University Cadi Ayyad, P.O. Box 2390, Marrakech, Morocco
Phone: +212 44 43 46 49 Fax: +212 44 43 74 07
lazrek@ucam.ac.ma
http://www.ucam.ac.ma/fssm/rydarab

Abstract

The main goal of this contribution is to present a program that allows the composition of variable-sized curved symbols such as those occurring in mathematics. This application, called CurExt, extends the capabilities of the well-known TEX system designed by D. E. Knuth for typesetting. Big delimiters, such as brackets, or special curved symbols, such as the Arabic mathematical summation symbol, can be built automatically according to the size and the shape of the concerned mathematical expression. CurExt will make it possible to stretch Arabic letters according to calligraphic rules in order to draw the kashida. It follows a useful tool for justifying texts written with the Arabic alphabet. Unlike in Latin alphabet based writing, where the justification is done through inserting small blanks among characters, cursive writing fills in the space between characters with the kashida.

Résumé

L'objet principal de cette contribution est de présenter un programme qui permet la composition de symboles curvilignes extensibles comme ceux qui se présentent en mathématique. Cette application, appelé CurExt, étend les capacités du système bien connu TEX développé par D. E. Knuth pour la composition de document. Les grands délimitants, comme les parenthèses, ou les symboles curvilignes spéciaux, comme le symbole arabe pour la somme, peuvent être composés automatiquement suivant la taille de l'expression mathématique contenue. CurExt permettra également l'allongement de certaines lettres arabes, de façon curviligne, en accord avec des règles de la calligraphie. Un corollaire important de cette possibilité de traiter la kashida sera de permettre la justification du texte arabe. Rappelons que là, il s'agit d'un problème profond dû au fait que la justification dans une écriture cursive, comme l'arabe, se fait à l'aide de la kashida, à l'opposé du moyen de justification des textes à écriture non cursive, qui se fait au moyen d'une distribution adéquate de blancs entre les mots de la ligne.

The problem

Variable-sized symbols Besides *fixed size* symbols, such as characters of the Latin alphabet in a given font or basic mathematical symbols (e.g., +, −), there are symbols with a context dependent size. Some mathematical symbols, such as delimiters or Arabic characters, are examples of these *variable-sized* symbols. The variation of size can concern:

- the width of the symbol, such as in:
  - the various parts of some symbols (e.g., $\sum$ or $\overline{\Big\langle}$, $\Rightarrow$). A horizontal stretching, according to the expression covered by the operator, is sometimes necessary;
  - the kashida for some characters of the Arabic alphabet (e.g., بـ , جـ) or certain Arabic mathematical symbols such as those found in the abbreviations of usual functions (e.g., مجـــــ , جـــذ , نهـــا) in mathematical expressions. The kashida ـــ, a small curve, is used to stretch some characters in order to cover the concerned mathematical expression or to break the line when the left margin is reached;
  - some diacritics or accents (e.g., $\underline{abc}$, $\overline{abc}$, $\widehat{abc}$, $\widetilde{abc}$, $\overleftarrow{abc}$, $\overrightarrow{abc}$, $\overbrace{abc}$, $\underbrace{abc}$ or
    $\underbrace{\text{ابح}}$ ، $\overbrace{\text{ابح}}$ ، $\overrightarrow{\text{ابح}}$ ، $\overleftarrow{\text{ابح}}$ ، $\widetilde{\text{ابح}}$ ، $\widehat{\text{ابح}}$ ، $\overline{\text{ابح}}$ ، $\underline{\text{ابح}}$).
- the length of the symbol, such as in:
  - delimiters (e.g., ⟨, (, |, [, ||, {, }, ||,], |,), ⟩);
  - symbols of operators (e.g., $\int$ or $\left.\right\rangle$, $\left\{\right.$, $\Uparrow$).
- the width and the length of the symbol, such as in some mathematical symbols: (e.g., $\sqrt{}$ or $\sqrt{}$).

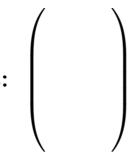
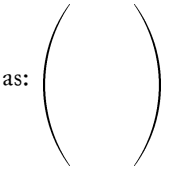
Production of the variable-sized symbols Several different approaches can be adopted to produce variable-sized symbols:

- *Through measuring*: glyphs are made on the basis of measurements directly taken from the context of the symbol. This manner leads to a very high precision but is *a posteriori*. A second processing of the text is absolutely necessary, after a first one where measurements of sizes are taken and recorded. The glyphs can be produced once the measurements are made. This way of proceeding makes it possible to have one glyph of sign per size through dynamic fonts.
- *Ready to wear*: standardized sizes are determined. Glyphs are then drawn according to this set of sizes. No second processing will be necessary of course. The precision cannot reach the level attained through the previous manner. A glyph of sign per interval of sizes, through static fonts, is then produced *a priori*.
- *Semi-finished*: a combination of the two previous ways.

There is a difference between producing these glyphs and using them. The production can be done through programs, with parameters to be determined, with METAFONT or PostScript. A particular system of editing will be necessary to make use of the glyphs. This system will be used to send the required parameters to the previous program for generating the fonts.

Curvilinearity of the variable-sized symbols Besides symbols drawn with *segments* (e.g., [,], ⇒), there are variable-sized symbols composed with small *curves* (e.g., (,), ∫). Thus, the extensibility of the symbol should be done with respect to this curvilinearity. The task becomes difficult for the shapes of the curves that vary according to the size. Then, the problem is not limited to stretching or lengthening these curves. It consists of producing curves according to different sizes. As far as we know, up till now, there is no system that allows the production of variable-sized curved symbols through parameterized dynamic fonts. Say, for example, a curvilinear variable-sized parenthesis. Neither TEX [5] nor MathType[1] offers such a possibility. An attempt had been made with the Math-Fly[3] font [1] in the system Grif/Thot[4] [10]. It didn't go far from initial experimentation, nor has it been added to the system. The system Ω[6] doesn't offer the possibility of producing such symbols. An extension to ditroff/ffortid[7] allows the abilities to stretch letters themselves with dynamic PostScript fonts [3].

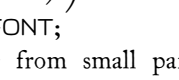
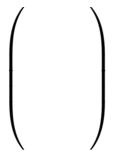
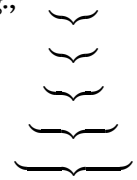
A very detailed survey on the ways for producing variable-sized symbols and the general problem of the optical scaling can be found in [1].

The parentheses in big sizes, say, for example, those of matrices offered by TEX, look like this:  . Can we get curved parentheses such as:  ?

In the same way, one can wonder how to produce kashida so as to get the correct writing: بـــجـــ, instead of the simple straight-line lengthening: بـــجـــ.

Variation of symbol sizes in TEX

TEX handles the problem of producing variable-sized symbols through two different ways at the same time, with the Computer Modern font developed by D. E. Knuth in METAFONT [6]:

- through the production, a priori, of glyphs up to some sizes[8] (e.g., ), using the primitive charlist from METAFONT;
- through composition, starting from small parts, whenever the size goes beyond some level (e.g., ), using the primitive extensible from METAFONT. Horizontal or vertical segments are then added to get the desired size (e.g., 

1. MathType is an equations processing system, including Equation Editor, from Design Science, Inc.[2]

2. http://www.mathtype.com or http://www.dessci.com

3. Math-Fly is a parameterized PostScript type 3 font.

4. Thot is an interactive system for the production of structured document. It is an evolution of the Grif system developed by the Opera team with the INRIA and the IMAG. Amaya, the W3C browser,[5] is based on this system.

5. http://www.w3.org

6. http://omega.enstb.org

7. ditroff/ffortid is a system for formatting bi-directional text in Arabic, Hebrew and Persian developed by J. Srouji and D. M. Berry [11].

8. Some other sizes can be obtained by yhmath package, which extended math fonts for LATEX, developed by Y. Haralambous: CTAN:/macros/latex/contrib/yhmath

or $\left\{\left\{\left\{\left\{\right.\right.\right.\right.$.

When a certain size limit is reached, some symbols can't be extended any more[8] (e.g., $\widehat{a}$, $\widehat{ab}$, $\widehat{abc}$, $\widehat{abcd}$, $\widehat{abcde}$).

The primitives `\Bigg`, `\bigg`, `\Big` and `\big` (e.g., `\Bigg( ... \Bigg)`) denote various possibilities of extension that the user can specify to get the desired size. The primitives `\left` and `\right` (e.g., `\left( ... \right)`) allow an automatic determination of the size according to the context.

The TEX compiler keeps room for each character. The character is considered as a rectangular box, with a width and length, on a baseline, passing between the height and the depth. These values are taken from the corresponding `TFM` files of the fonts in use.

The font should be determined a priori in the preamble of the document. No specification of size will be allowed while the document is processed. A possibility of *magnification*, for all text, is allowed a priori, in the preamble of the document. This magnification consists of a reduction or an enlargement of the font.

Curvilinear extensibility

Parameters of dynamic font Hereafter, a solution for obtaining variable-sized curved symbols is proposed. The particular case of parentheses, brackets of a mathematical expression, will be presented in detail, as an example of *vertical* extensibility. The kashida, as an example of *horizontal* extension, will be presented also. It goes without saying that all other variable-sized curved symbols can be handled in the same way.

The size of such symbols is determined a posteriori according to the context. Instead of taking the size of the symbol from the `TFM` files of the specified font, this size is computed starting from the size of the mathematical expression covered by the symbol. The required room is then reserved (using the TEX primitives `\hbox` and `\vbox` with `\wd`, `\ht` and `\dp`). Then, the program METAFONT is called to generate the fonts according to the computed sizes. Thus, dynamic fonts are built.

The necessary repetition of processing is not a problem, for TEX will already be called two times at least for the table of the contents, the bibliography, the index, etc.

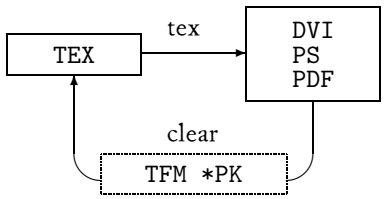
This way of processing will lead to the following constraints:

- there is a need for one font per character. TEX can use simultaneously up to 16 math font families. This can be a restriction of the number of variable-sized symbols that can be processed;
- for every variable-sized symbol, a file for storing the sizes computed after the first compilation is required. The number of files allowed by TEX for the input/output (by using the primitives `\read` and `\write`) is limited to 16. That also limits the number of variable-sized symbols to deal with;
- the program METAFONT allows up to 256 symbols per font processed by TEX. This seems to limit the number of different sizes of glyphs. This is not a true limitation because the width of a symbol is already limited by the width of the sheet and likewise for the length.

Actually, the number of required files can be reduced through the use of the same file for each pair of delimiters: the closing bracket is generally required wherever an opening one appears. Thus, the same file will be used for the two brackets. Another reduction in the number of required files may be carried out through recording a size only once for all the occurrences of a given symbol at a given size. TEX doesn't allow the use of a file for both input and output. A file opened for output will be overwritten whenever it is called once again for input and conversely. This leads to the use of an intermediate temporary file to look for any possible existence of a given size. This file will be used for all variable-sized symbols handled.

As was said before, the number of sizes of a variable-sized symbol is limited to 256. Beyond this limit, the system will use the smallest size higher than the required one. The system holds the biggest size as the default size. The intermediate file used previously will be used to determine this size of substitution. This will be done for all variable-sized symbols.

The size of an extensible symbol of an expression can be given if the expression itself does not comprise another extensible symbol. A problem arises when several extensible symbols are overlapping in the same expression. TEX and METAFONT should be called as many times as variable-sized symbols overlap. The `TFM` and `*PK` files must be cleared in order to compute them every time with the new sizes as the font's parameters, as in the following:



Glyph parameters During the development of such a dynamic font of variable-sized symbols, some difficulties arise in determining:

- the shape of the glyph according to the dimensions of the character (e.g., the curvilinearity, the level of concavity or convexity, of a bracket);
- the shape of the glyph at both small sizes and big sizes;
- the position of the glyph, points of control, according to the dimensions of the character;
- the dimensions, the width and the length (height plus depth), of the box of the character;
- the position, the height and depth, of the character compared to the baseline;
- the position of the character with respect to the other characters of the same line, the space between characters;
- the position of the character compared to the expression covered by this character;
- etc.

Choices should be made with respect to:

- the nature of the symbol that determines the space both before and after the composed symbol (e.g., `\mathinner`);
- the lengthening to be added to the initial length (respectively the width) of the expression covered by the brackets (respectively the kashida) under the terms of the rules of the typography;
- the form of boundaries of kashida in other to join it with the preceding part and/or the following part (e.g., for the symbols for sum, product and limit);
- etc.

For example, the parameters that determine the brackets are:

- the level of curvilinearity of the bracket;
- the level of thickness of the middle of the bracket;
- the level of thickeness of the ends of the bracket;
- the shape of the ends of the bracket, as they are to be identical at the top and the bottom;
- the shape of the bracket depends on the size of the covered expression.

Examples:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 & 4 \end{pmatrix}$$

The kashida always has the same thickness, but the concavity of the kashida varies within limits fixed by the rules of the calligraphy of the style *Naskh* [4].

The symbol ص obtained by the command `\lsum` from the system RyDArab[9] [8, 9], is a composition of the fixed part ص, obtained with the `xnsh` font from ArabTEX[10] [7], and of the extensible rectilinear part ـــ whose effective size depends on an automatic examination of the context.

The symbol ص obtained by the command `\csum` from the system RyDArab with the present package CurExt, is a composition of the fixed part ص , of the font NasX[11], and the variable-sized curvilinear part kashida ـــ whose effective size depends on an automatic way of the context. The problem of drawing these parts of a component symbol arises then.

Examples:

ص ص ص

$\sum_{ن = ط - ف - ح - 1}$ $\sum_{ن = ط - ف - 1}$ $\sum_{ن = ط - 1}$

Syntax of commands Hereafter, some commands offered by the CurExt package.

The syntax for the combined parentheses command is:

```
$\parentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\left(\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}\right)$$

The syntax for the open (or left) parenthesis command is:

```
$\openparentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\left(\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}\right.$$

The syntax for the close (or right) parenthesis command is:

```
$\closeparentheses
 {\matrix{1 & 2 & 3\cr
          4 & 5 & 6\cr
          7 & 8 & 9\cr
 }}$
```

$$\left.\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}\right)$$

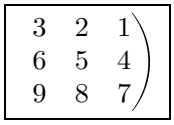
The syntax for the open (or right) parenthesis, in an Arabic mathematical presentation, command is:

9. The extension RyDArab is a system for processing mathematical documents in an Arabic presentation. It allows the composition of mathematical expressions with specific symbols spreading out from right to left according to the Arabic writing. It has been developed by the author.

10. The extension ArabTEX is a system for processing Arabic textual document. It has been developed by K. Lagally.

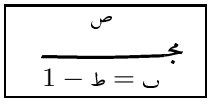
11. The font NasX is a kernel of an Arabic mathematical font. It offers some Arabic literal symbols. It has been developed by the author.

```
\arabmath
$ {\openparentheses
 {{\matrix{1 & 2 & 3\cr
           4 & 5 & 6\cr
           7 & 8 & 9\cr
 }}}}$
```



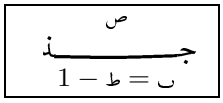
The syntax for the Arabic sum command is:

```
\arabmath
$ {\csum_{b=T-1}^{s}}$
```



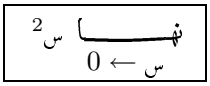
The syntax for the Arabic product command is:

```
\arabmath
$ {\cprod_{b=T-1}^{s}}$
```



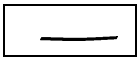
The syntax for the Arabic limit command is:

```
\arabmath
${\clim_{c \to 0}
 c{{}^2}}$
```



The syntax for the Arabic kashida command is:

```
\arabmath
$ {\kashida{9mm}}$
```



Conclusions

The application CurExt allows composition of variable-sized curvilinear symbols. This system will make it possible to compose automatically delimiters of mathematical expressions that can vary in a bi-dimensional way. It also allows the composition of kashida in Arabic mathematical symbols such as the symbols of *sum*, *product* and *limit*.

The number of various sizes for the brackets or the kashida is limited to 256. Beyond this limit, the system will use the smallest size higher than the required one or the biggest size already computed. The number of occurrences of the same size is unlimited.

A choice of the parameters to compose brackets is made with respect of typographic and calligraphic rules in use. A compromise between certain parameters is necessary. In some cases, the choice is subjective.

The system CurExt will allow looking after the typography of the variable-sized curvilinear symbols such as brackets or the integral symbol. It also makes it possible to take into account the compliance with the rules of the calligraphy of a cursive writing such as Arabic. The kashida will be carried out in the strictest respect of Arab calligraphy.

A significant application of such a system will be the justification of a text in a cursive writing through complying with the calligraphic rules.

In addition to the previous limits, METAFONT fonts under an ASCII encoding generate many problems with the new formats of multilingual electronic documents. A study has been started (and since completed, see [2]) for PostScript Type 1/3 or OpenType fonts under a Unicode encoding.

References

[1] Jacques André and Irène Vatton, *Dynamic optical scaling and variable-sized characters*, EP—ODD 7 (1994), no. 4, 231–250.

[2] Mostafa Banouni, Mohamed Elyaakoubi, and Azzeddine Lazrek, *Dynamic Arabic mathematical fonts*, International Conference on TEX, XML, and Digital Typography — TUG 2004, Xanthi, Greece, Springer Lecture Notes in Computer Science (LNCS), vol. 3130, Springer-Verlag, 2004, http://www.springerlink.com/index/URHRT2EYKYHH1RPA, pp. 149–157.

[3] Daniel M. Berry, *Stretching Letter and Slanted-baseline Formatting for Arabic, Hebrew and Persian with ditroff/ffortid and Dynamic PostScript Fonts*, Software—Practice & Experience (1999), no. 29:15, 1417–1457.

[4] Mohamed haCm al XTaT, *Les règles de la calligraphie arabe, Ensemble calligraphique des styles d'écritures arabes*, Univers des livres, Beyrouth, Liban (1986), in Arabic.

[5] Donald Ervin Knuth, *The TEXbook*, Addison-Wesley, 1984.

[6] ———, *The METAFONTbook*, Addison-Wesley, 1986.

[7] Klaus Lagally, *ArabTEX — Typesetting Arabic with Vowels and Ligatures*, EuroTEX'92, Prague (1992).

[8] Azzeddine Lazrek, *A package for typesetting Arabic mathematical formulas*, Die TEXnische Komödie, DANTE e.V. 13 (2001), no. 2/2001, 54–66.

[9] ———, *Aspects de la problématique de la confection d'une fonte pour les mathématiques arabes*, Cahiers GUTenberg 39–40, Le document au XXIe siècle (2001), 51–62.

[10] Vincent Quint and Irène Vatton, *Grif: an interactive system for structured document manipulation*, Text Processing and Document Manipulation, ed. J. C. van Vliet, Cambridge University Press, Cambridge, UK (1986), no. 4, 200–213.

[11] Johny Srouji and Daniel M. Berry, *Arabic formatting with ditroff/ffortid*, EP—ODD 5 (1992), no. 4, 163–208.

Allographic Biometrics and Behavior Synthesis

Vlad Atanasiu
Massachusetts Institute of Technology
USA
atanasiu@mit.edu
http://mywebpage.netscape.com/atanasiuvlad/frq/

Abstract

This essay deals with the role of allographs in the general framework of the study of writing. It explains why allographs are important (writer individuality), how they can be measured (frequency and contexts (graphic, semantic, non-graphic)) and what applications to expect from them (biometrics, script restoration, artistic creation, cultural history).

Résumé

Cet essai traite du rôle des allographes dans le cadre général de l'étude de l'écriture. Il explique pourquoi les allographes sont importants (l'individualité de l'écrivain), comment les mesurer (fréquences et contextes graphiques, sémantiques et non-graphiques) et à quelles applications s'attendre (biométrie, restauration de l'écriture, création artistique, histoire culturelle).

One of the common elements used to identify a typeface or a handwriting is the shape of the script. Sometimes, however, individual characters may have alternate forms — in those cases the rules and habits governing their usage characterize the behavior of the typesetter or writer in a more subtle way than shape does. Allographs provide inconspicuous signatures to a text.

This paper intends to raise the issue of the study of allographs — allometry — , for the pleasure of its intricacies as well as for its many practical applications. I will proceed by discussing four diagrams, representing some of the central issues of the subject. Although my discussion is based on Arabic handwritten examples — a script where allographs are an inherent part of the writing system — , the same principles can be applied to non-Arabic hand- or machine-produced scripts.

Measuring frequencies

The two text blocks in Fig. 1 represent a double-page of a manuscript copied in Sicily in the 14th century [1]. On the right side we see the original script (Fig. 1a), whereas the left side (Fig. 1b) is a replacement folio, added at a later date, after some folios went missing. Some attempts were made to keep the aspect even: the dimension of the text-block and the line number is the same and both scripts are in the *naskh* style. But even an eye unaccustomed to Arabic script will feel the presence of crude restoration work.

The paleographic expertise can be automated; the magnitude of a 2D Fast Fourier Transform of the pages will reveal two main differences. The axis-angle of the cross-like spectra are different, which means that the vertical strokes of the replacement script are more slanted than the original ones (Fig. 1c has a nearly horizontal axis, Fig. 1d a more oblique one; the vertical axis comes from the regular spacing of the 13 lines). The four bright spots away from the center of Fig. 1d suggest a stronger linearity of the replacement script, whereas the dispersion in Fig. 1c points to a rounder and more irregular handwriting in the original script).

But there is an even more powerful distinguishing feature: two allographs of the *kaf* letter are used by the replacement, the oblique one (marked by a circle, 13 times in Fig. 1f) and the long one (marked with a disc, 7 times in Fig. 1f), whereas only the oblique appears in the original (19 times in Fig. 1e). It is also remarkable that most (4) come in the context of the same word (*kana* "to be"). Defined by quantities and contexts never obvious to the eye, the allographs offer a reliable security filigree for testing the integrity of the manuscript.

Applications

Figs. 2a and 2b are excellent examples of the principal application of allometry: *biometrics*, or the spatio-temporal identification of scripts and their attribution to individuals or traditions (a subject of interest to police and intelligence agencies, the judicial system, the art market, historians, et al.). The examples presented here are written by two calligraphers for a competition and transcribe the same text [2, p. 31–32]; however, the number of long *kaf*-s is 6 and 23, respectively. Thus the allographs of the elongated type clearly play a very different role as "visual attractors" from one individual to

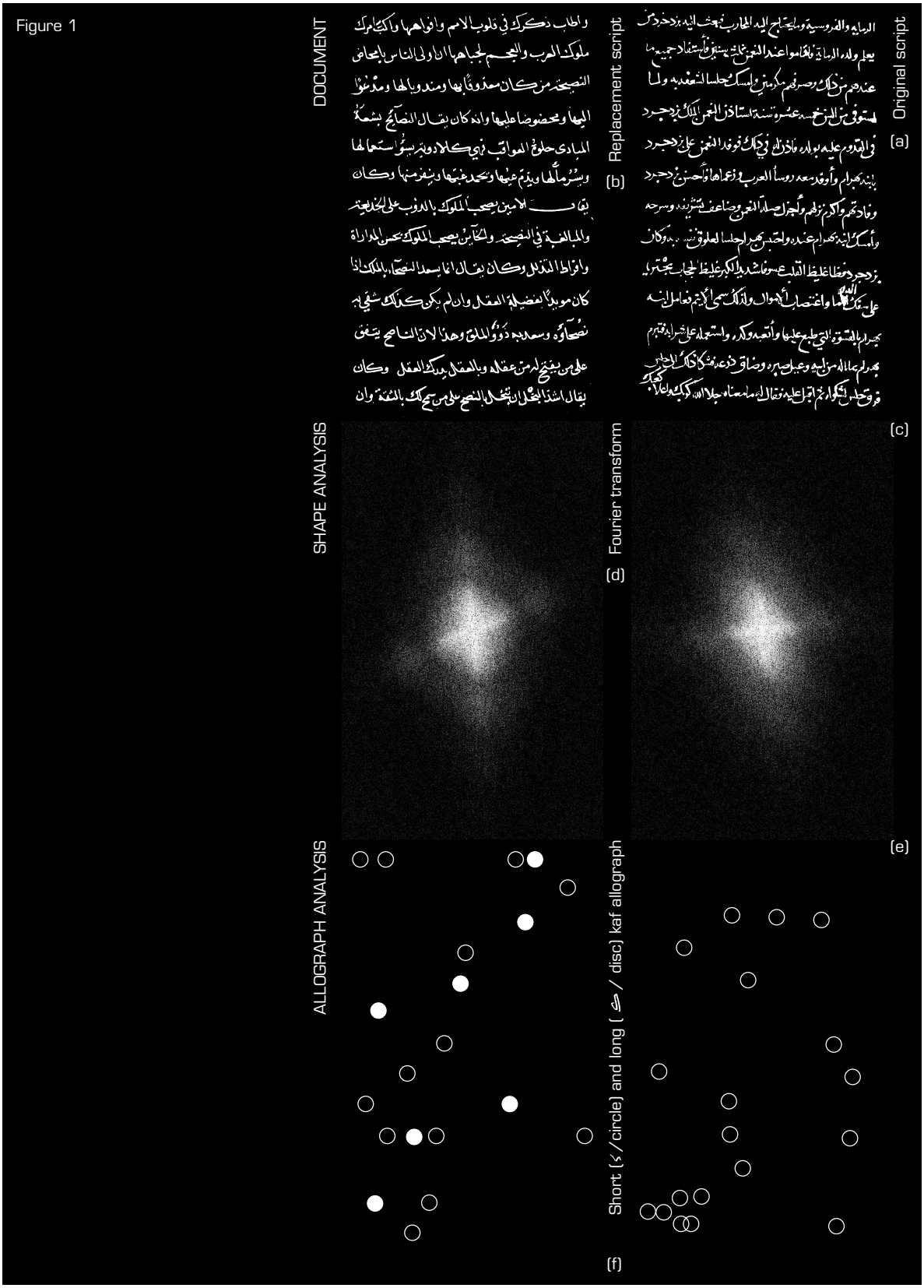
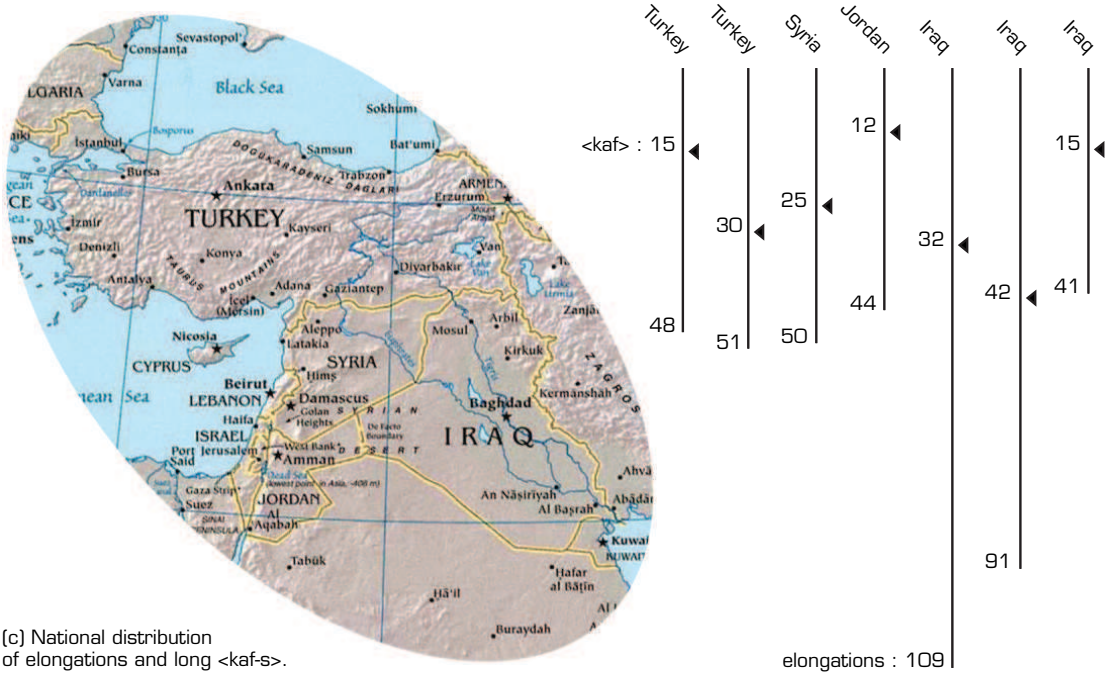


Fig. 1: Letters beyond shapes: quantity and contexts

Figure 2 (b) (a)
Strong and weak visual attractors:
for the same text, but for different calligraphers the usage of the rare long <kaf> allograph is 6 for 23.



(c) National distribution
of elongations and long <kaf-s>.

Fig. 2: Allographic behaviours: individuals and traditions

another. The same can be observed at the level of traditions. On the basis of five other participants in the calligraphic competition, we see that Turks globally use a very similar number of elongations (but differ individually in their attribution to specific graphemes). They are closely connected to the Syrian, Jordanian and part of the Iraqi trends. In Iraq, however, two traditions prevail, with one making an extensive use of elongations, very unlike Turkish calligraphers.

Once the script analysis is done, it can serve as a basis for a *cultural history*. Thus, the coherence of the Turkish style is remarkable, witness to the force still extant today of a tradition hundreds of years old. The allographic differences are the expression of an Arabic style independent of the Turkish school — an anti-monopolistic tendency anchored in the political, economic and social make-up of the Middle East. Given that the Iranian calligraphic styles need a good deal of elongation, one also has to ask what the influence of Iran is on its neighboring Iraqi calligraphy. Arabic calligraphy still being an important artistic player in Islamic countries, such analysis can reveal some preferential channels of influence and provide clues for policymaking.

Another active use of allometry is *behavioral synthesis*; after having extracted information about the way a person or tradition handles allographs, it can be used to predict future behavior, whether in known textual situations (typically the restoration of a damaged or missing part of a writing) or for virtual texts, never written by the individual (these are revival texts, "in the style of ..."). Models of behavior should be seen as an important aspect of typographic automation for its capacity to adapt graphic output to specific contents or lectorates (an Ottoman text with an Arabic allograph frequency would be historically odd and certainly disapproved of by the calligraphic establishment).

Finally, allometry can be used to enhance the visual aspect of a text in ways too complex or tedious to be feasible by human minds and hands. For example, allographs are known to be used for producing graphic figures or rhythms, but they occur almost always locally, at the page level, and stretch very rarely across a manuscript of several hundred pages (which thus becomes more a suite of chamber music than a symphony). However, with a program for *computer aided script*, a designer could concentrate on modeling the figures and rhythms, fine tuning and harmonizing them, and leaving the detail rendering to the machine. Just as sound synthesizers help composers, in the same way could computers generate written texts of a far more complex structure than anything seen before.

At a methodological level, allography leads us to define the intellectual goals of the study of writing (*graphonomics*[1]) as follows. The graphic *analysis* is a required step of script identification. It enables the building of a *model* for the reproduction of a specific writing behaviour and in a later stage its modification and use in the process of new graphic *creations*. Script recognition leads also to the understanding of the non-graphical aspects that shaped its existence: Beside its *physical aspects* (spatial, temporal, force, acoustics, ...), this graphical ecology is *technological* (materials, tools, process), *informational* (mathematical, statistical, informational properties of writing systems), *biological* (physiological, neurological, psychological) and *cultural* (social, economic, artistic, literary, customary, religious, ..., aspects of writing).

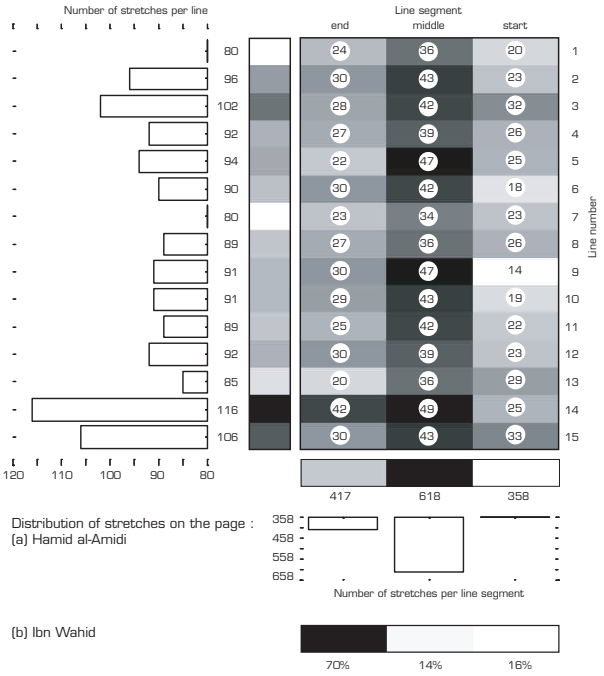


Fig. 3: The simultaneous contexts problem

Measuring semantic contexts

In a calligrapher's mind the presence of a word with a strong meaning (for himself or within the context of the text) implies using a special graphic output for that word. In an Arabic working text this happens generally by using less frequent allographs. In Fig. 3 we see the example of a word containing the letter *kaf*, *kufr*, which means "impiety". In the religious context of the text — Qur'anic citations, holy sayings, moral advice — this word triggers in 70% of cases the usage of an alternate allograph in the group of calligraphers studied (the same as for Fig. 2).

1. Graphonomics is a field of study different from graphology, in the same way that astronomy is distinct from astrology. It has its own association, conference and publication; see `http://www.cedar.buffalo.edu/igs`.

In contrast, *malaka* "to possess", a word less relevant to the text, is only in 20% of the cases attributed the long, exceptional, *kaf*. A side effect of the transfer from lexical meaning to visual display of exceptional status, is the creation of a cultural bond between the otherwise random association of a meaning and a specific shape. Poets have started to use the expression *kaf-i kufr* "the *kaf* of impiety" and explain the elongated shape of the letter by comparing it to the "narrowness of heart" of some people. The metaphor strengthens the practice of the calligrapher by wrapping it in a cultural halo and a web of meanings. The semantic triggering of an allograph exemplifies the second most important aspect of an allograph, after its frequency: the contexts where and why it is used.[2]

Problems and limits

Fig. 4 presents a case of space management — that is, the various techniques for fitting a specified amount of text on a finite inscribable surface. Here I analyze the behavior of a writer regarding the elongation of graphic segments in the particular setting of the Qur'ans of the Ottoman tradition called "*ayet ber kenar*", where a verse ending must coincide with the end of a page, thus forcing the calligrapher to carefully consider not only the line, but also the page justification (more references to this calligraphic feat in [4]). The superposition of all elongations of part of a manuscript [5, p. 2–48] show a fluctuation in the quantity of elongations from line to line. This means that the text was not studied so as to let the elongations spread uniformly across the page and that the writer needs an adjustment period observed in the first third of the page before settling to a more constant quantity and then again overusing the elongations during the two lines before last to fit his text. He shows, however, a stronger ability of prescience — or faithfulness in his credo of uniform justification — than the Mamluk calligrapher of Fig. 4b, who comes to the conclusion that he needs an elongation in order to justify only at the very end of lines [6, vol. 3, fol. 99v°0–166v°]. So the Turkish calligrapher starts thinking ahead earlier, generally in the middle of a line. The low quantity of elongations in the first part of lines indicate that they are primarily used as justification devices. Technical treatises of Arabic calligraphy and typography confirm these findings.

However, if we want to quantify the justification context, we have to consider that what we see on a page is the result of the simultaneous presence of several contexts, each having specific demands on the graphic output. For example, we saw in Fig. 1 that the long *kaf* is an important part of all elongations, thus the uneven distribution of elongations could be partly due to a particular distribution of words containing the letter *kaf*. In fact it is not possible to consider just one context at a time, and a theoretic framework has to be developed to disentangle the contribution of each context.

The principal hurdle comes from the number of potential contexts that can coexist (consider also other aesthetic reasons that could have led the calligrapher to make the choices he did, such as line or word balance, black vs. white equilibrum, etc.), from the "melting" of contexts (how to know which ones actively influenced the choice?) and the knowledge that we can have about them (some are "hidden" — such as page justification would probably be for a casual reader –, yet others are "forgotten" — such as the mood of the writer, which certainly was a factor in his performance). These are the frontiers of our present knowledge — some to be left unchallenged until time machines and nano-submarines for brain exploration are invented.

References

[1] Ibn Zafir, *Sulwan al-muta' fi 'udwan al-ittiba'*, TRI Publishing, Kuwait, 1984.

[2] Mohammed Tamimi, *Catalogue of Winners' Plates in the Second International Calligraphy Competition*, Research Centre for Islamic History, Art and Culture, Istanbul, 1991.

[3] Vlad Atanasiu, *Hypercalligraphie. Le phénomène mamluk à l'époque du sultanat mamluk* (*Moyen Orient, XIII^e-XVI^e siècle*), thèse de doctorat, École pratique des Hautes Études, Paris, 2003, `http://mywebpage.netscape.com/atanasiuvlad/the/`.

[4] ——, "Visible *i'ajaz* — Three graphic particularities in the Qur'an copies", in *Actes du colloque de Bologne 2003 sur les Manuscrits coraniques*, François Déroche ed., E.J. Brill, Leiden, 2003, `http://mywebpage.netscape.com/atanasiuvlad/pubs/atanasiu2003iajaz.pdf`.

[5] *Qur'an*, calligraphed by Hamid al-Amidi, Hizmet Vakfi, Istanbul, 1974.

[6] Ibn Wahid, *Qur'an*, Cairo, 1304–6 (London, British Library, ms. Or. Add. 22408).

2. A classification of allographic contexts for the Arabic script and its comparison with Latin script can be found in my doctoral thesis [3, chap. 10, "La grammaire graphique"].

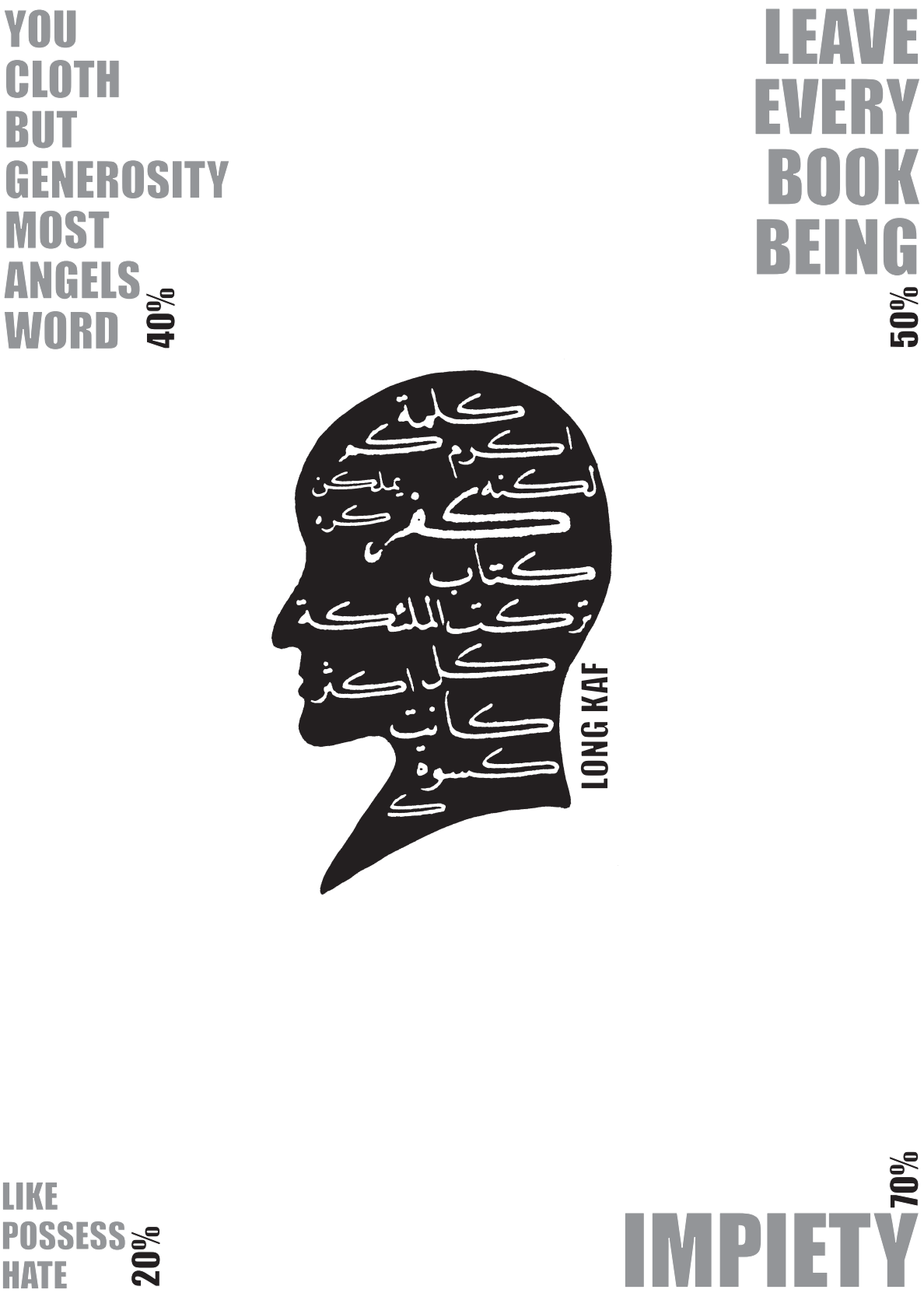


Fig. 4: Contextual aspects: association strength between meaning and shape

La virgule viendrait-elle de l'écriture arabe ? Constatations sur ses origines graphique — et fonctionnelle

Ghassan Mourad
Laboratoire LaLICC (Langage, Logique, Informatique, Cognition et Communication, UMR 8139)
96, bd Raspail, 75006 Paris, France
Ghassan.Mourad@paris4.sorbonne.fr
http://www.lalicc.sorbonne.fr/~mourad

Abstract

This paper examines the origins of the graphical form of the most commonly used punctuation sign, the comma. It is a multi-functional sign, whose functions depend on the context. Its presence or absence can change the meaning of a text. As with any other sign, according to the broader meaning of the term, the comma has a meaning in a graphical stream, which affects and modifies the meaning of a textual segment.

I have reached certain conclusions about the origin of the comma, based on a study of Arabic texts and more particularly of the particle *wāw* — its functionality and its similarity to the comma in certain writings. The comma as we know it did not exist before the invention of the printing press. It came to us thanks to the Italian Renaissance publishers who were inspired by the *wāw* to create the form of a comma. This similarity pertains not only to the graphical domain but also to the functionality and use of the two signs.

The paper concludes with some remarks on the punctuation in Arab writing.

Résumé

Les questions qui m'interpellent portent sur l'origine de la forme graphique de la virgule.

En effet, la virgule est le signe de ponctuation le plus utilisé dans l'écriture ; c'est un signe polyvalent et multi-fonctionnel. Ces fonctions dépendent du contexte. Une virgule change le sens d'un énoncé selon sa présence ou son absence. La virgule a comme tout autre signe, dans le sens large du terme, une signification dans une chaîne graphique ; ce signe affecte et infléchit le sens d'un segment textuel.

D'après l'étude que j'ai menée, j'ai pu exprimer certaines constatations sur cette origine. Celles-ci résultent d'une observation de textes arabes et plus précisément de la particule arabe *wāw*, de sa fonctionnalité et de la similitude qu'elle présente avec la virgule dans certaines écritures. La virgule telle que nous la connaissons n'existait pas avant l'invention de l'imprimerie. En effet, elle nous serait venue grâce aux imprimeurs italiens de la Renaissance qui se seraient inspirés de la particule *wāw* pour réaliser l'aspect actuel de la virgule. Cette similitude ne concerne pas uniquement le domaine graphique ; mais elle implique également la fonctionnalité et l'utilisation de ces deux signes.

Enfin, nous notons quelques remarques sur la ponctuation dans l'écriture arabe.

Je vous avais dis : supprimez tout un morceau, si une virgule vous déplaît dans le morceau, mais ne supprimez pas la virgule, elle a sa raison d'être.
BAUDELAIRE[1]

Introduction

Nous mettons en évidence dans cet article l'étude de l'origine de la forme graphique de la virgule.

En effet, la virgule n'est connue dans sa forme graphique actuelle qu'après l'invention de l'impri-merie. Pour cette raison, nous nous intéresserons en premier lieu à l'origine de la forme graphique de la virgule qui, selon nous, a été inspirée par celle d'une particule sonore arabe à savoir la lettre *wāw* (و). En deuxième lieu, nous ferons une synthèse du fonctionnement du *wāw* et de sa liaison avec la virgule actuelle selon différents aspects. En effet, l'étude de la virgule proprement dit et de sa complexité nécessite une étude plus approfondie qui dépasse le cadre de cet article.

1. Lettre de Baudelaire à Charpentier, directeur de la Revue Nationale.

Note de la rédaction: Il est possible que cet article sera repris dans un prochain numéro de *TUGboat*, avec de legères améliorations des exemples arabes. Pour l'instant, veuillez accepter le texte arabe tel quel.

Constatations sur l'origine de la virgule (forme graphique et similitude fonctionnelle)

La virgule, du moins dans sa forme graphique, nous serait venue d'Italie. Elle aurait été introduite dans l'écriture en Occident par les imprimeurs italiens soucieux de diffuser les valeurs de la Renaissance. Parmi ces imprimeurs figure la famille des Manucio (plus connu sous le nom de Aldes) dont Alde l'ancien à qui l'on doit notamment le caractère italique et Alde le jeune qui dirigea l'imprimerie du Vatican. La révolution culturelle de la Renaissance s'appuya notamment sur les œuvres antiques dont l'œuvre des philosophes grecs. Nous savons à quelle point la contribution des savants arabes fut importante dans ce domaine, notamment par leurs interprétations des œuvres d'Aristote et par leurs propres œuvres dans le domaine de la médecine (Averroès, Avicenne, ...). Ceci joua un rôle considérable en Europe jusqu'au XVII^e siècle. Ainsi les imprimeurs italiens de la Renaissance qui travaillaient sur l'impression et la publication d'anciens manuscrits grecs, latins et écrits dans d'autres langues dont l'hébreu et l'arabe, n'ont pas échappé à cette influence. Étant souvent confrontés à des écritures différentes, ils s'en sont inévitablement inspirés dans leur production de textes. Par conséquent nous pensons que la virgule actuelle (,) a été empruntée à l'écriture arabe. Elle serait la transcription de la lettre arabe *wāw*. Nous nous efforcerons dans ce qui suit de démontrer la plausibilité de cette hypothèse.

Constatation 1 D'après Ibn Al Nahhās, le calife Abū Bakr (VII^e siècle) rencontra un jour un bédouin et lui demanda de lui vendre son chameau. Ce que le bédouin refusa de faire en répondant :

لَا عَفَاكَ ٱللّٰهُ

La traduction de cet énoncé serait :

«Que Dieu ne te pardonne pas»

Abū Bakr lui répliqua en rectifiant cet énoncé des deux manières suivantes :

لَا ( ) عَفَاكَ ٱللّٰهُ
لَا (و) عَفَاكَ ٱللّٰهُ

La traduction de ces deux énoncés peut être :

Soit : «Non, que Dieu te pardonne !»
Soit : «Non et que Dieu te pardonne !»
Soit : «Non, et que Dieu te pardonne !»

Le bédouin dans sa réponse n'aurait pas respecté la pause nécessaire après la particule de négation *lā*. En effet cette pause sépare cet énoncé en deux car elle constitue une ellipse («non je ne veux pas te vendre mon chameau et que Dieu te pardonne»). De même la particule *wāw* sépare l'énoncé en deux en marquant une reprise énonciative. Ainsi le non respect de la pause équivaut à la non utilisation de la particule *wāw*, ce qui rend l'énoncé du bédouin ambigu voire incorrect. De même si nous observons la traduction des deux énoncés proposés par Abū Bakr, nous constatons que la pause (silence) et la particule *wāw* peuvent être traduits de la même manière par une virgule, par la conjonction «et», ou par les deux à la fois.

Constatation 2 Notons que la particule *wāw* en arabe est un connecteur, dont un des rôles est celui de la conjonction et comme nous le verrons par la suite.

Constatation 3 Dans l'écriture arabe la virgule est utilisée actuellement, mais pour ne pas la confondre avec le *wāw*, on l'écrit avec une rotation.

Constatation 4 Dans son utilisation actuelle, la virgule en arabe n'a aucune valeur syntaxique. Sa valeur est uniquement pausale. En ce qui concerne les différentes valeurs de la virgule, elles sont, pour la plupart, exprimées par la particule *wāw*.

Constatation 5 La particule *wāw* peut dans certains cas avoir le rôle de connecteur, tels que «alors», «alors que», «car», etc.

Différentes valeurs de la particule wāw *et sa similitude avec la virgule et d'autres signes de ponctuation*

La particule *wāw* de coordination a plusieurs valeurs :

- Valeur d'union — tout type d'union (مُطلَق ٱلجَمع)

  Ex. 1 :

  تَرتَدِي مَلَابِسَ مُلفِتَه لِلنَظَر تَدليلًا عَلَا ٱهتِمَامِهَا بِٱلمُوضَا و مُتَابَعَتِهَ لِأَخبَار ٱلنُجوم و مَيلِهَا لِلغَرب

  Traduction : «Elle s'habille de manière voyante pour montrer son intérêt pour la mode, pour l'actualité des stars, et son penchant pour l'Occident.»

- La coordination dans un énoncé peut concerner des noms.

  Ex. 2 :

  قَوس قُزح مُؤلَّف مِن ٱللَون ٱلأَحمَر و ٱلخَضَر و ٱلزرَق وٱلخ

  Traduction : «L'arc-en-ciel est composé des couleurs rouge, bleu, vert, etc.»

- La coordination dans un énoncé peut concerner des groupes propositionnels.

  Ex. 3 :

  رسمتُ بِقَلَمِ ٱلحبر و بِقَلَمِ ٱلرَصَاص و بِقَلَمِ ٱلتَلوين

. *Al khata' wa al i'tināf*, cité par Abd Al Sattār Ibn Mohammad Al Awnī dans [1, p. 265–318].

Traduction : «J'ai dessiné avec un stylo, avec un crayon à papier et avec un crayon de couleur»

- La coordination dans un énoncé peut concerner les deux termes d'une alternative.

  Ex. 4 :

  إِمَا شَارِبَ و إِمَا آكَلَ

  Traduction : «Soit en buvant, soit en mangeant.»

- La coordination dans un énoncé peut concerner des propositions relatives.

  Ex. 5 :

  رَيتُ ٱلرَجُلَ أَلَّذِي كَتَب ٱلمَقَال و أَلَّذِي نَشرَه و أَلَّذِي قَرَءهُ

  Traduction : «J'ai vu l'homme qui a écrit l'article, qui l'a publié et qui l'a lu.»

- Dans certains emplois, la particule *wāw* implique une conséquence où elle est précédée d'un certain type de proposition, à savoir qu'il y a des conditions modales. En effet, la proposition qui précède doit être non assertive (ordre, interrogation, souhait, etc.).

  Ex. 6 :

  زُرني و ازُورَكَ
  رَاسلهُ و يُرَاسلُكَ

  Traduction : «Visite-moi, je te visiterai», «Écris-lui, il t'écrira.»

  Nous constatons que du point de vue de la coordination, il existe une similitude entre les emplois de la virgule et du *wāw*. Ainsi, comme nous l'avons vu dans les exemples, les *wāw* sont traduites par des virgules et inversement.

- Autre que la valeur de coordination, la particule *wāw* peut avoir une valeur énonciative. En effet, le *wāw* peut introduire une incise.

  Ex. 7 :

  لتَقَينَا بِفُلَان و هوَ كَاتبُ مَشهور ثُمَ ذَهَبنَا آلَى ٱلمَقهَى

  Traduction : «Nous avons rencontré un tel, écrivain célèbre, puis nous sommes allés au café.»

- Elle peut aussi avoir une valeur de reprise.

  Ex. 8 :

  ذَلِكَ و مَن يُعَظِمْ شعَاإِرَ ٱللهُ فَإِنَهَ مِن تَقوا ٱلقُلُوب . . .

  (سُورَة ٱلحَج)

  Traduction [9, sourate XXII] : «(33) Il en sera ainsi. Celui qui observe les divers rites de Dieu, tels les offrandes, fait une action qui provient de la piété dans le cœur.»

  Dans ce verset la particule *wāw* sert à lier deux énoncés indépendants. En effet, Kasimirski a traduit ce *wāw* par un point, ce qui indique clairement une reprise énonciative.

  Ex. 9 :

  و لَمَا ذَهَبَ رَضوَان قَالَ آدهَم لِنَفسِهِ (. . .)
  و وقَفَ آدهَم يومًا يَنظُرُ ءلا ظاقِلِهِ (. . .)
  نجيب مَحفُوظ (أولَاد حَارفتنَا)

  Traduction [10, p. 19] : «Lorsque Radouan s'en fut, Adham se dit à lui-même [...]» — «Un jour Adham se tenait debout regardant son ombre [...]»

  Nous pouvons dire que dans ces deux phrases les *wāw* indiquent un début de paragraphe. Ce *wāw* agit au niveau du texte. En effet il sert à lier des énonciations distinctes et peut se traduire par un point ou un alinéa.

  Ex. 10 :

  جَاءني زَيدٌ و ٱلشَمسُ طَالعٌ

  Traduction : «Zaed est venu alors que le soleil brillait.»

- Le *wāw* en arabe est un connecteur, il joue le rôle de «alors que», «et», «donc», «car» (cf. Ex. 6 : «écris-lui, il t'écrira») où il peut être traduit par l'expression «alors» («écris-lui alors il t'écrira»).

  Ex. 11 :

  يُرِيدُ ٱللهُ آن يُخَفِفَ عنكُم و خَلَقَ ٱلآنسَانَ ضَعيفًا

  Traduction [9, sourate IV, p. 91] : «(32) Dieu veut rendre son joug léger, car l'homme a été créé faible.»

  Dans ce verset, le *wāw* est traduit par la conjonction «car» qui introduit une explication (c'est un *wāw* explicatif).

Nous constatons que le *wāw* assure une jonction tant au niveau de la phrase qu'au niveau du texte. En effet, les connecteurs jouent un rôle complémentaire aux signes de ponctuation dans un texte. Dans cette perspective, ils jouent le rôle de marqueurs d'intégration linéaire. Pour ce qui est de l'arabe, nous n'avons pas accès à la plupart des manuscrits anciens. Les éditions sont responsables de la mise en page des textes classiques dont nous disposons aujourd'hui.

Mais, nous avons eu accès à des copies de manuscrits anciens (cf. Al Th'alibi, p. 15–20) où nous avons pu constater deux choses :

. Comme nous le savons, il en est de même pour la plupart des manuscrits classiques français.

1. l'existence de quelques signes de ponctuation, dont le point «en haut» fait partie pour marquer la cadence de la phrase dans la prose rimée ;
2. l'existence de signes qui marquent ce qui pourrait être un alinéa. Le premier mot qui suit ce signe est écrit en gras. L'emploi du *wāw* se place sur trois niveaux : la proposition, la phrase, et le texte (paragraphe) (فصل, en gras, p. 19, Al Th'alibi, fig. 1).

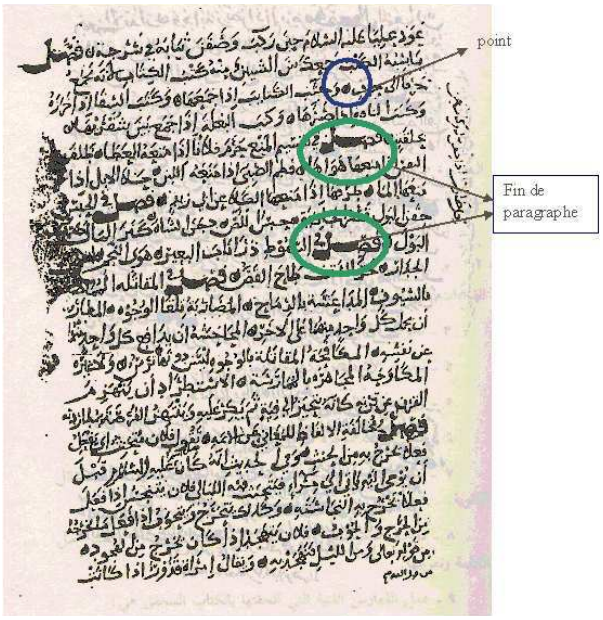


Fig. 1 : Al-Th'alibi, *Fiqh al Lougha* (x^e s.), p. 19.

Pour résumer, on trouve seulement trois sortes de signes qui nous paraissent être vraisemblablement des signes de ponctuation. C'est ainsi que les connecteurs deviennent indispensables à la linéarité du texte.

Ainsi, nous dirons qu'il est probable que le découpage en paragraphes est du fait des éditeurs et ne se trouve pas dans les manuscrits initiaux. Après avoir fait plusieurs observations empiriques, en comparant un ouvrage du x^e siècle et un ouvrage du xx^e siècle, nous avons constaté que dans l'autore classique avec sa mise en page moderne, l'usage des connecteurs en début de paragraphe est très fréquent, alors que dans l'ouvrage moderne, il est rare de trouver des connecteurs entre les paragraphes. Nous remarquons que l'usage des connecteurs est très fréquent pour séparer les segments et assurer leur enchaînement et leur intégration dans les textes arabes anciens. Il en est de même concernant les textes médiévaux en Occident, comme l'avait signalé Catach, où l'on constate un usage abondant de connecteurs («et», «alors») [5, p. 7].

Aujourd'hui, dans les écrits arabes, les auteurs notamment dans la presse ont un penchant pour la ponctuation importée de l'Occident. Étant importée, et appliquée sans règles précises, elle ne peut qu'être artificielle dans l'organisation textuelle.

Notre propos devient pertinent si on observe, dans l'ouvrage d'Al-Th'alibi (p. 158, fig. 3), la mise en page du manuscrit qui se trouve à la page 19 du même livre (fig. 2). On constate que l'auteur (Saleh)) a utilisé des signes de ponctuation (, ; : ! « »).

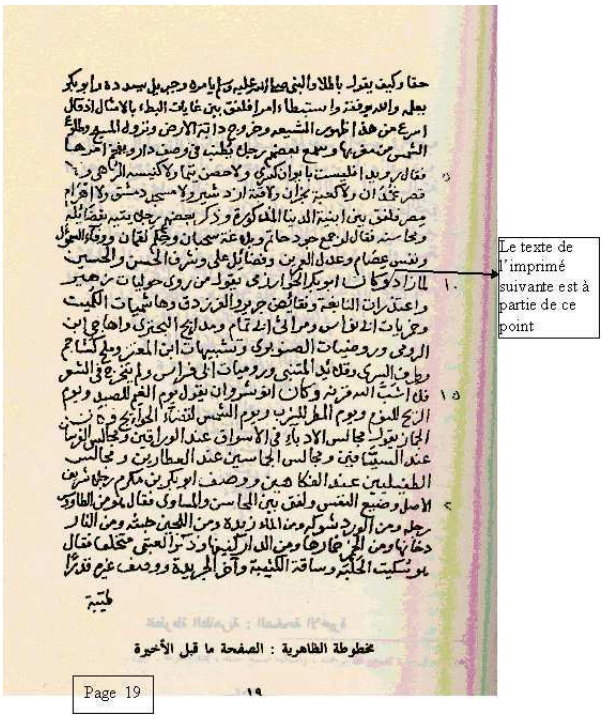
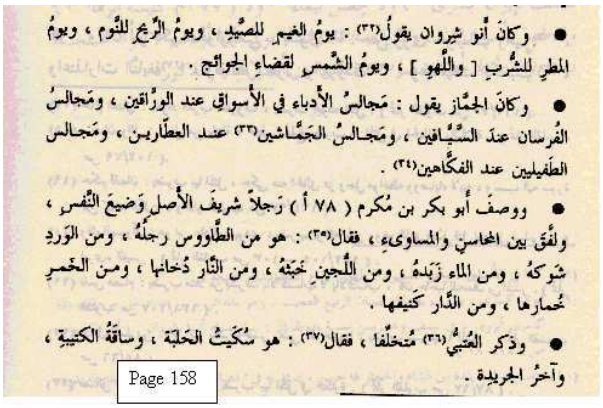


Fig. 2 : Al-Th'alibi, *Fiqh al Lougha* (x^e s.), p. 19.

. Déjà dans le texte coranique il y avait des marques pour indiquer une pause pour la récitation du Coran. Dans les versions du Coran du Moyen Âge la fin d'un verset était marquée par les signes «o», «O», «*». De même, on trouvait ces signes dans différents manuscrits désignant la fin d'une phrase comme dans le manuscrit de Al-Th'alibi, x^e s., *Fiqh al Lougha*.

Les marques de ponctuation utilisées aujourd'hui sont le résultat de la Renaissance arabe au xix^e s. grâce à l'imprimerie, à la confrontation avec l'Europe et au mouvement de traduction d'œuvres étrangères. Les marques de ponctuation sont celles du système d'écriture européen, mais n'ont pas pour autant les mêmes valeurs, en particulier la virgule qui n'a pas toujours la fonction de coordination. D'autre part, le point en arabe n'est souvent utilisé que pour marquer la fin d'un paragraphe, alors que la virgule est utilisée pour déterminer la fin d'une phrase. En revanche les autres signes de ponctuation tels que les guillemets, les parenthèses, les points d'exclamation et d'interrogation, les trois points, etc. ont la même valeur que ceux des langues européennes.

À notre connaissance, il n'y a que trois études traitant de la ponctuation en arabe ; un livre de Ahmad Zaky (1912), un autre de Abed Al Rauf Albasri (1932) et un livre récent de F. Isaak qui selon nous n'est pas approprié à la langue arabe par le fait qu'il y applique le système de ponctuation européenne [11].


● وكانَ أنو شيروان يقولُ[٣٢] : يومُ الغيمِ للصَّيدِ ، ويومُ الرِّيحِ للنَّوم ، ويومُ المطرِ للشُّربِ [واللَّهوِ] ، ويومُ الشَّمسِ لقضاءِ الحوائج .

● وكانَ الجمَّاز يقول : مَجالسُ الأدباء في الأسواقِ عند الورَّاقين ، ومَجالسُ الفُرسان عندَ السَّيَّافين ، ومَجالسُ الجَمَّاشين[٣٣] عند العطَّارين ، ومَجالس الطَّفيليين عند الفكَّاهين[٣٤] .

● ووصفَ أبو بكر بن مُكرم (٧٨ أ) رَجلاً شريفَ الأصلِ وَضيعَ النَّفسِ ، ولفَّقَ بين المحاسنِ والمساوىءِ ، فقال[٣٥] : هو من الطَّاووس رجلُهُ ، ومن الوَردِ شَوكُهُ ، ومن الماء زَبَدُهُ ، ومن اللَّجين خَبَثُهُ ، ومن النَّار دُخانها ، ومن الخَمرِ خُمارها ، ومن الدَّار كَنيفها .

● وذكرَ العُتبيُّ[٣٦] مُتخلِّفا ، فقال[٣٧] : هو سُكَيتُ الحَلبَة ، وساقَةُ الكتيبةِ ، وآخرُ الجريدة .

Page 158

Fig. 3 : Al-Th'alibi, *Fiqh al Lougha* (x^{e} s.), p. 158.

Or, le *wāw* qui servait de connecteur dans le manuscrit a été conservé.

Ainsi, la ponctuation n'apporte rien de plus au texte qui aurait pu garder sa linéarité sans l'ajout de ces signes. Le seul changement concerne la mise en page (le texte est plus aéré), bien que dans chaque début de paragraphe le *wāw* demeure.

Deux questions s'imposent ici : Si l'auteur a gardé les connecteurs dans le texte, est-ce pour rester fidèle au manuscrit ? Ou bien a-t-il jugé ces connecteurs nécessaires à la cohérence du texte ?

À l'inverse, on s'interroge alors sur le besoin d'utiliser des signes de ponctuation qui s'avèrent ici superflus.

En effet, la ponctuation devient pertinente et signifiante quand elle remplace les «connecteurs ponctuationnels», comme dans le cas de l'énumération. Aujourd'hui des auteurs arabes remplacent le *wāw* coordonnant dans l'énumération de substantifs par une virgule. Cela paraît naturel au lecteur de voir une suite de mots sans recours à la conjonction *wāw*, car il existe dans la langue arabe une règle qui autorise l'absence du *wāw* quand il s'agit de l'énumération d'une série d'adjectifs épithètes.

Ex. 12 :

جَاءني رَجُلٌ شَاعرٌ مُقرَئٌ كَاتِبٌ

Traduction : «Un homme poète, lecteur et écrivain, m'a rendu visite».

Ainsi, la langue arabe ayant sa spécificité doit adopter un système de ponctuation qui lui est propre et qui répond aux exigences liées à l'évolution technologique.

Il est désormais manifeste que dans l'activité langagière, les langues s'influencent mutuellement (introduction de mots d'origines différentes d'une langue à une autre). Ceci semble moins évident dans le domaine de l'écrit, même si l'histoire de l'écriture nous donne des exemples concluants (l'adoption de l'alphabet arabe par les Perses, de l'alphabet latin par les Turcs, etc.). Nous posons la même question concernant la particule *wāw* qui, de lettre sonore en arabe, serait devenue caractère muet en latin. De plus, le «connecteur ponctuationnel» *wāw* (en plus de sa forme graphique), dans son emploi pour la coordination, pour l'introduction d'incises et pour la reprise énonciative, ressemble à la virgule. Dans l'écriture arabe la virgule semble ne pas être toujours nécessaire : le *wāw* joue ce rôle.

References

[1] Revue *Alam Al Fikr*, Koweit, 1987.

[2] Brun, J., A. Doppagne, *La Ponctuation et l'art d'écrire*, Baude, Bruxelles, (1957).

[3] Catach, N. (sous la direction de), «Recherches historiques et actuelles sur la ponctuation», dans *Actes de la table ronde internationale CNRS*, Paris, Publications CNRS-HESO, 1977–1979.

[4] Catach, N., «La Ponctuation», *Langue française*, n° 45, Larousse, Paris, février 1990.

[5] Catach N., *La ponctuation*, Presses Universitaires de France, Paris, 1994.

[6] Doppagne, A., *Majuscules, abréviations, symboles et sigles*, Duculot, Louvain-la-Neuve, 1991.

[7] Doppagne, A., *La Bonne ponctuation*, 2^{e} éd., Duculot, Louvain-la-Neuve, 1992.

[8] Fonagy, I., *Structure sémantique des signes de ponctuation*, Bulletin de la Société de Linguistique de Paris, 1980.

[9] Biberstein-Kasimirski, A., *Le Coran*, 1970, éd. Garnier-Flammarion.

[10] Mahfouze, N. (éd.), *Dar Al Adab*, Beïrout, 1986

[11] Mourad, G., «La Segmentation de textes par l'étude de la ponctuation», *Actes de CIDE'99, Document Électronique Dynamique*, p. 155–171, Damas, Syrie, 1999.

[12] Recanati, F., *Oratio obliqua, oratio recta. The semantics of representation*, Bibliothèque du CREA, Paris, MIT Press, 1999.

[13] Védénina, L., *Pertinence linguistique de la présentation typographique*, Paris, 1989.

Quelques remarques sur le sens et la servitude de la typographie — pratiques, discours et imaginaires

Emmanuël Souchier
École Nationale Supérieure des Télécommunications, Paris
Emmanuel.Souchier@enst.fr

Abstract

How does meaning enter the typography (an aspect which so often goes unnoticed by the reader) of a text? How do typographers themselves view this question? And what role does talking about it (typography) play? In what part of our imagination does the body of "industrial text" show itself?

These are some of the questions we will try to touch upon.

Résumé

Comment le sens vient-il à la part typographique du texte, cette part souvent ignorée des lecteurs ? Comment les praticiens envisagent-ils la question ? Et quelle part doit-on concéder aux discours qui accompagnent le fait typographique ? Dans quel imaginaire s'est déployé le corps du texte industriel ?

Quelques questions que l'on tentera d'ébaucher au fil de notre propos.

Écriture & typographie

La typographie est cet art, cette technique qui depuis un demi millénaire déjà, nous donne à voir la part visuelle du texte industriel. La «*part typographique*» [1, p. 59–74] des textes est essentielle à la lecture, en ce qu'il ne saurait y avoir d'écriture sans sa livrée graphique, sans une matérialité susceptible de la révéler, au sens propre du terme. Le dessin, le graphisme — ou sa version technique, le typo-graphisme — a pour fonction première de donner à voir le texte, de le rendre perceptible, quel que soit le système d'écriture envisagé : alphabétique, idéographique, pictographique, mixte ...

Avant que de s'adresser à notre intellect, à travers la culture qui l'aura modelée, l'écriture s'adresse à nos sens. Matière ou trace sur un support ou un espace, elle se donne avoir ou à tracer et relève de l'œil aussi bien que de la main.

Aussi l'écriture est-elle avant tout duale, cristallisant deux énergies essentielles et contradictoires en une expression unique où le souffle de la langue s'allie à la geste de l'image. Deux énergies irréductibles, du «geste et de la parole» [2], indissolublement liées en un seul et même mouvement. L'écriture donne à voir ce que l'oreille ne sait entendre. Et c'est de ce lien et de ce déchirement, de cette tension primordiale, que sourdent ses différentes manifestations qui, suivant l'histoire et la géographie, se sont exprimées selon les registres variés de la figuration ou de l'abstraction. Peinture ou signes d'écriture, la variation s'exécute du dessin à l'alphabet.

Média Janus, l'écriture est double. Son identité est marquée par cette dualité constitutive, tout a la fois linguistique et iconique. Mais elle n'est ni linguistique ni iconique. Elle procède des deux à la fois et doit répondre, à travers cette identité complexe, à la fonction de communication qui est la sienne. L'écriture est un média à part entière doué d'un registre d'une très grande richesse qui va de l'expression artistique à la simple circulation d'information.

La typographie est quant à elle ontologiquement liée à l'écriture en ce qu'elle la donne à lire, la révèle à son tour. Elle en est l'empreinte, l'image, le double industriel qui s'en est peu à peu affranchi. S'intéresser à la question de la signification de la typographie, c'est donc fatalement revenir à l'histoire de l'écriture et réinterroger sa fonction autant que sa constitution. Historiquement, le terme est attesté en 1554 (*tipographe*), il entérine l'industrialisation de l'écriture. Mais pour industriel qu'elle soit, la typographie n'en est pas moins écriture, affirmant par son étymologie la dimension visuelle qu'elle reniera dans son discours d'accompagnement : au type (*tupos*), frapper (empreinte en creux ou en relief que laisse la frappe d'une matrice), elle allie la marque, la figure, l'image (*graphein)* et l'activité qui concerne l'art d'écriture ou de dessiner (*graphikos*).

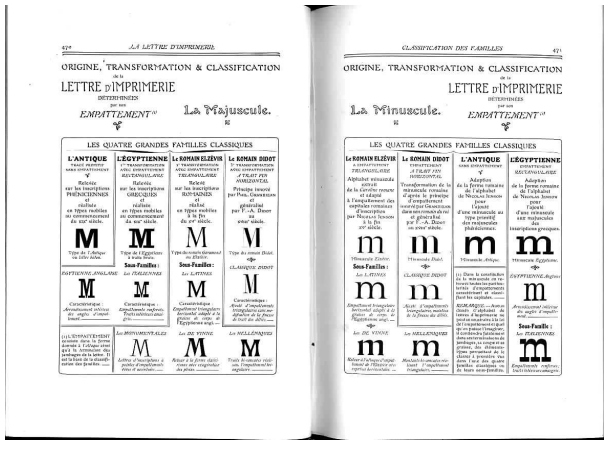
Voici donc une activité technique et industrielle qui relève de la médiation visuelle. Mais cette caractéristique technique et industrielle ne fait que se substituer à la pratique artisanale et manuelle. La typographie est à l'écriture ce que la voiture est à l'hypomobile : un substitut technique qui ne change pas la fonction première du média (la communication ou la locomotion), mais transforme les conditions de son usage et sa réception sociale.

Sens & typographie

L'une des questions que posent les études sur la typographie à trait à ses modalités de signification, partant à la

signification de l'image [3]. Comment le sens advient-il à l'art typographique ? Quelle différence y a-t-il entre un texte composé en Baskerville ou en Didot, par exemple ? Et sur quels critères objectifs puis-je faire reposer mon interprétation ? Car, si comme l'écrit Raymond Gid «*la lettre vit ...et signifie*» [4, p. 21], comment parvient-elle à signifier ?

Pour en comprendre la signification, les théoriciens (à l'origine souvent praticiens) ont commencé par classer les objets auxquels ils avaient à faire. Les traits pertinents qui ont alors été retenus pour distinguer les familles de caractères l'ont été, la plupart du temps, sur des critères historiques et formels. Je ne retiendrai pour éclairer mon propos que les classifications les plus significatives et leurs critères méthodologiques. Ainsi de la première classification de Thibaudeau (1921) [5, p. 470] où se croisent l'élément graphique (l'empattement de la lettre comme élément discriminant[1]) et l'élément historique (chacune des quatre grande familles étant définie en fonction de sa période d'apparition ou d'usage dominant : l'Antique, l'Égyptienne, l'Elzévir et le Didot).



Vox, pour sa part, fera adopter sa classification par l'ATypI en 1962[2] en récusant le caractère «*trop abstrait*» de son prédécesseur et en intégrant les «*catégories empiriquement acceptées en langue anglaise*». Il définit en outre «*une valeur biologique*» qu'il voulait «*d'usage universel*» [6, p. 122]. Le critère de «*valeur biologique*», qui ne cesse d'être problématique, s'inscrit dans une idéologie qui prônait alors un universalisme de bon aloi fondé sur un formalisme rejetant tout particularisme historique, culturel ou géographique et dont le parangon typographique devait être l'*Univers* d'Adrian Frutiger (1957), caractère dessiné dans la lignée du *Futura* de Paul Renner (1927) et de l'*Architype* d'Herbert Bayer (1925) directement hérité du Bauhaus [7, p. 181].

Cette position sera clairement remise en cause par les tenants d'une inscription culturelle de la production typographie. Ainsi Ladislas Mandel note que «*l'écriture la plus fonctionnelle comporte toujours une dimension que nous pourrions qualifier de géographique*». Soucieux de l'histoire et des «*signes identitaires*» de l'écriture et de la typographie, Mandel précise que «*l'on ne pourrait composer correctement un texte allemand et un texte français avec le même caractère*», allant jusqu'à formuler l'idée selon laquelle «*on ne saurait dissocier la formulation visuelle de la pensée elle-même*» [8, p. 4–13] et [9], idée proche de la notion de «*sens formel*» [3] sur laquelle il nous faudra revenir.

Mais si dans sa classification Vox exclut apparemment l'apport historique de son propos théorique, il ne cesse toutefois d'y revenir pour justifier ses choix.

On appréciera ainsi, sur cet exemple dédié aux «*Mécanes*», le mélange de formalisme, d'historicisme et de présupposés idéologiques fondés sur une érudition forgée au cœur des savoirs professionnels, mélange qui ne parvient toutefois pas à expliquer l'heuristique typographique.

Dans son *Anatomie de la lettre* [16, p. 10], sans élaborer de nouvelle classification, Jacno fait le pari d'une description «*concrète*» débarrassée «*de toute abstraction*». Il privilégie alors «*les notions de forme, d'alternance (succession des noirs et des blancs d'une ligne de texte), de valeur expressive (qualité qui donne au texte sa physionomie)*». Son propos est tourné vers la définition d'un «*nombre minimum de lois générales*» résumées à trois ordres différents de considérations :

Jacno œuvre en anatomiste ; son objet, disséqué avec art, est soustrait de son contexte (le texte mis en situation). Non qu'il fasse abstraction de l'histoire, la chose ayant, selon lui, déjà été traitée par ailleurs «*avec la*

1. «*Toute lettre — avons-nous établi — tout caractère d'imprimerie (sauf les types d'écriture)* trouve sa caractéristique dans son empattement, *c'est-à-dire dans la forme de la terminaison de ses jambages.*» [5, p. 416].

2. Association TYpographique Internationale, 1962.

3. Sur l'adaptation sémiotique de cette notion littéraire empruntée à Jacques Roubaud [10], voir [11, 12]. Du point de vue de l'histoire de l'écriture et des médias, voir [13, p. 52–56], [14, p. 258], [15, p. 35].

L'ordre de classement qui a été suivi dans cet ouvrage est celui de la CLASSIFICATION NOUVELLE DES CARACTÈRES D'IMPRIMERIE établie en 1952 par l'École de Lure et publiée par l'école Estienne à Paris. Exposée au séminaire 1953 de l'Institut Graphique de Stockholm, au Salon TPG de Paris 1954, à Monotype House de Londres 1955, à Barcelone, Lausanne, Anvers, Florence par des conférences ; étudiée et discutée dans la presse internationale, la 'classification Vox' a été adoptée en 1962 par l'Association Typographique Internationale, conjointement avec la classification D.I.N. rédigée en Allemagne par Hermann Zapf, et qui n'en diffère que par les termes du vocabulaire. Elle est utilisée par de nombreuses imprimeries, fonderies et ateliers de composition en France et dans le monde.

Nullement révolutionnaire, revenant même sur ce que le principe français dit 'de Thibaudeau' avait de trop abstrait, la Classification A.TYP.1 1962 consacre les catégories empiriquement acceptées en langue anglaise : mais en leur donnant une valeur biologique et une définition précise, d'usage universel, dont on pourra trouver ici l'exposé.

Quant aux noms nouveaux créés en français pour les familles de base, il est apparu qu'à des concepts révisés correspondait une terminologie rénovée. Elle fait désormais partie du langage professionnel.

Le graphisme des 52 signes (26 capitales et 26 minuscules) qui composent l'alphabet offre trois ordres différents de considérations :

I. Les *éléments constitutifs* de la lettre.

II. Les *modulations* sur l'alphabet élémentaire, qui sont des enrichissements multipliant les possibilités d'expression de la lettre.

III. Enfin le *blanc*, ce milieu intérieur, cet élément négatif qui alterne avec le noir des lettres.

I. — *Éléments constitutifs*, se classant en trois groupes distincts :
1° 5 éléments de *structure ;*
2° 3 éléments d'*alignement ;*
3° 6 éléments *décoratifs.*

II. — *Modulations principales* (au nombre de 7) :
1° Graisse;
2° Stature;
3° Italique;
4° Style;
5° Contraste;
6° Rondeurs aplaties;
7° Nuance anglaise.

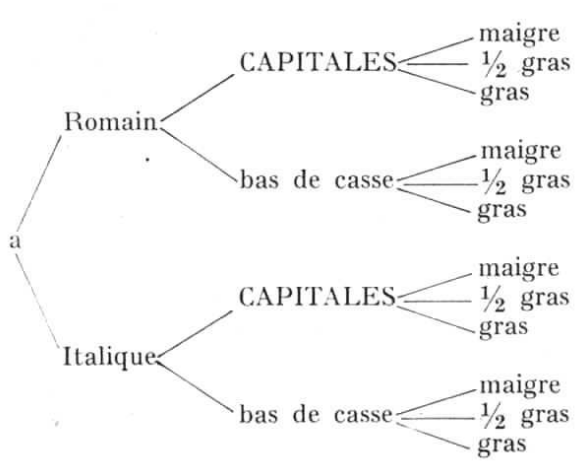
III. — *Les blancs :*
1° Blancs des lettres;
2° Espaces.

clarté souhaitable», mais bien qu'il soit nécessaire «*d'étudier dans la typographie, là où ses normes apparaissent avec le plus de pureté*» [16, p. 9]. Une esthétique de la forme en somme qui, d'un point de vue théorique, ne se préoccupe pas de la fonction du texte et de sa dimension communicationnelle ou, plus exactement, qui fait de la dimension communicationnelle un objet théorique unifié absorbant toutes les distinctions situationnelles. Et c'est précisément sur cette fonction du texte qu'Emil Ruder mettra l'accent.

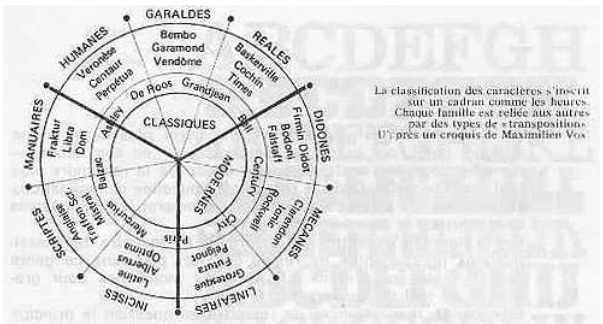
«*Les chefs-d'œuvre typographiques attestent une parfaite unité entre texte et forme*», écrit Ruder, ce qui prouve «*au typographe que la forme doit rester liée au but de l'œuvre, mais aussi qu'un pur fonctionnalisme ne suffit pas à la bonne forme*» [17, p. 34].

La notion de fonction sera reprise par Pierre Duplan qui, après avoir redéfini «*les invariants typographiques*» (romain / italique ; capitale / bas de casse ; variation de graisse), propose trois registres pour une «*classification fonctionnelle*» : «*jaillissement*, *objectivité*, *intégration*» correspondant respectivement à la «*typographie expressive*», la «*typographie fonctionnelle*» et la «*typographie invisible*» [18, p. 295–347]. Registres sur lesquels nous reviendrons lorsque nous aborderons la question du discours d'escorte.

Jean Allessandrini revisitera à son tour le théâtre typographique et proposera sa propre classification, le *Codex* 80 [19, p. 35–56]. En toile de fond, la critique adressée à Vox et Thibaudeau. Néanmoins, son propos consiste avant tout à renommer les anciennes familles et à intégrer les productions récentes marquées par l'afflux des «*lettres-transferts*» qui ont bouleversé l'esthétique des an-



nées 1970–1980. Ses «*Exotypes*» témoignent des «*rêves de voyage que l'Occident n'a cessé de nourrir*» [7, p. 74] et de la circulation des objets visuels dans un monde dont la culture s'internationalise par le truchement du «marché».

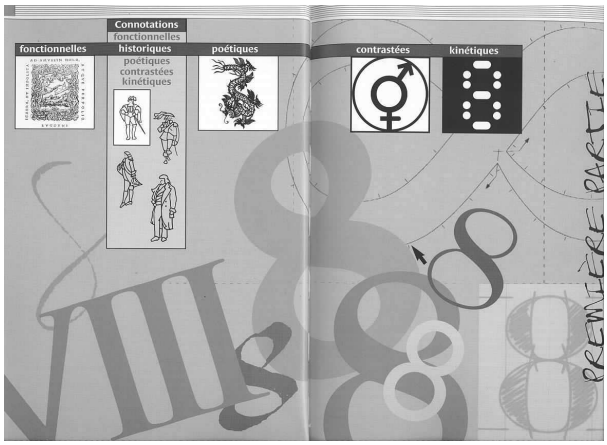


Système Thibaudeau (1921)	Classification Vox (1954)	Codex 1980	
		Désignations préliminaires	Eventualités
Antiques	Linéales	Simplices	DIAGONES / STENCILIENNES
Egyptiennes Egyptiennes anglaises	Mécanes	Emparectes Emparectes à congés	
Didots	Didones	Filaxtres Filaxtres à congés	
Elzevirs	Humanes Garaldes Réales	Claviennes	
Empattements triangulaires		Deltapodes Deltapodes à congés	
Helléniques	Incises	Romaines	
Cursives	Scriptes	Gestuelles calligraphiques Gestuelles brossées	
	Manuaires	Onciales	
Gothiques	Fractures	Germanes	
	Formes étrangères	Aliennes	
		Exotypes	
		Machinales	
		Ludiques	
		Hybrides	
		Transfuges	

Tableau comparatif des classifications Thibaudeau, Vox, Alessandrini

Cette révolution esthétique liée à une innovation technique — le décalcomanie — précède l'arrivée de la microinformatique qui va à nouveau bouleverser la donne. Toutefois, si la lettre-transfert se cantonnait essentiellement aux sphères professionnelles, le «texte mise en forme» grâce à la microinformatique sera pratiqué par les utilisateurs eux-mêmes. Les données relatives à la lecture et l'écriture sont dès lors à réévaluer en termes de prescription et d'usage.

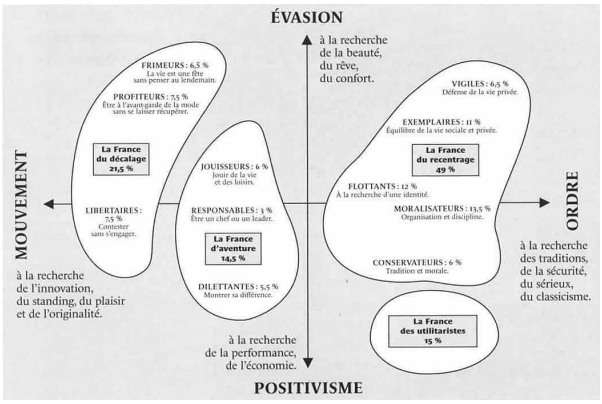
Gérard Blanchard est sans doute le premier en France à allier à la démarche du praticien, celle du théoricien et de l'historien. Au début des années 1980, il introduit la typographie à l'université [20] en systématisant le principe formel dans une classification d'inspiration structuraliste pour laquelle les discriminants tiennent compte des huit variables visuelles héritées de la *Sémiologie graphique* de Jacques Bertin (les *deux dimensions du plan*, la *forme*, l'*orientation*, la *valeur* (gris), la *taille*, le *grain* (trames), la *couleur*) [21], démarche qu'il assouplira par la suite pour privilégier une approche érudite fondée sur le concept sémiologique de «*connotation*» inscrit dans l'histoire et la culture.



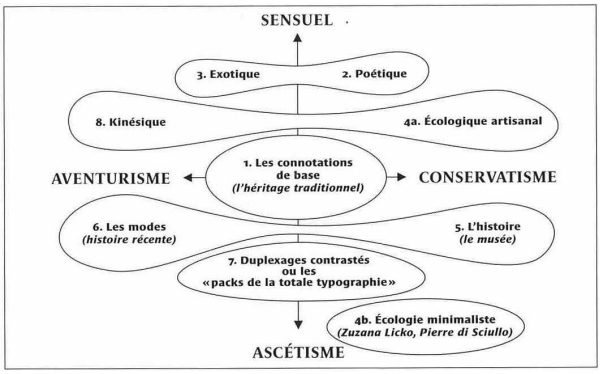
Blanchard s'inspire alors du *Musée imaginaire* d'André Malraux [22] et, d'un point de vue plus strictement méthodologique, se réfère explicitement à la démarche sémiologie du *Système de la mode* de Roland Barthes [23]. Lorsqu'il réinvestit la notion de connotation, Blanchard replace le contexte et l'interprétant au cœur du processus de signification [7, p. 36–37]. Aussi, face à l'hétérogénéité et à la complexité que ses prédécesseurs avaient tendance à évacuer, il est mené à convoquer ce qu'Abraham Moles nommera *Les Sciences de l'imprécis* [24].

Afin de rendre compte de l'évolution technique et esthétique de son temps, tout en conservant l'acquis de ses aînés, Gérard Blanchard propose une démarche théorique articulée en «*Cinq orientations pour huit connotations différentes*» [7, p. 40–41] cartographiés à la croisée de la tradition mnémotechnique de la *Carte du Tendre*

(1654) et de la méthode des socio-styles de Bernard Cathelat [25].



[Ci-dessus] Exemple d'une carte des socio-styles, faite en 1980 (dans *Le Publicitor*, 1989). [Ci-dessous] Les huit points de vue sur la connotation, répartis sur une carte des socio-styles établie sur le modèle proposé par Bernard Cathelat, en 1988.



Ce que Gérard Blanchard apporte est sans doute le trait d'union, jusque-là inexistant, entre une corporation de métiers jalouse de son histoire et de ses anciens secrets — alors même qu'elle est en passe de disparaître — et un monde académique qui condescend enfin à s'approprier «*l'image du texte*» [26, p. 137–145], domaine qu'il a par trop longtemps ignoré. Il conviendrait sans doute d'éclairer cette remarque à la lumière du contexte historique dans lequel a pu se déployer cette parole singulière (déclin de la typographie, informatisation des procédures de production du texte industriel, apparition de la micro informatique grand public, formalisation logicielle des pratiques de métiers, appropriation de la culture visuelle par le grand public écrivant, désenclavement des savoirs spécifiques et déclin de leur application ...). Au-delà de cette remarque, que convient-il de retenir du bref tour d'horizon consacré au fait typographique que nous venons de faire ?

Typographie & communication

Six critères essentiels peuvent être momentanément dégagés. L'*histoire*, tout d'abord, sur laquelle chacun s'accorde à fonder la raison d'être d'une création visuelle. Le *discriminant formel* ensuite, même s'il est de nature distincte selon les théoriciens. Puis la *fonction du texte* qui doit en déterminer la forme et par là-même l'esthétique. Enfin, le sens inscrit dans *la culture et l'usage des professionnels*. Reste la posture de Gérard Blanchard qui réinvestit en théorie *la place du lecteur* et *le rôle du contexte*. Autrement dit, qui sort la lecture typographique du cercle des gens de métier ou des formalistes pour lui apporter une réflexion sémiologique et communicationnelle nourrie des études littéraires. Il déplace ainsi les enjeux de lecture et d'écriture de «*l'image du texte*» du plan de la conception à celui de sa réception en pariant sur une interaction entre ces deux niveaux. Nous ne nous situons donc plus uniquement dans un débat professionnel, lié à la lisibilité de la lettre et du texte comme objets physiques destinés à une fonction prédéfinie, mais bien dans une réflexion d'ordre communicationnel où le contexte, la fonction du texte, la valeur d'usage et le rôle du lecteur vont être constitutifs du processus de communication, partant du processus de signification.

Une fois affirmée sa fonction communicationnelle, on ne peut définir la typographie qu'en termes situationnels et, plus précisément encore, en termes d'intégration. À l'appui de cette remarque, deux notions complémentaires doivent être convoquées, celles de *contexte* et de *cadre*.

En analysant l'interrelation entretenue, dans un logotype, entre le sens linguistique du nom d'un quotidien et sa dimension visuelle (iconique), j'avais souligné le fait que les éléments graphiques constitutifs de la forme visuelle (de la typo-graphie) ne signifiaient qu'en fonction de leur contexte [27, p. 87–100]. Cette remarque nous invite à remettre en évidence deux points essentiels à la compréhension du fait typographique. D'une part les signes graphiques (et par extension, les traits pertinents qui permettent de les distinguer formellement) n'ont pas de sens en soi. Le sens étant une coproduction élaborée par le lecteur à partir de l'objet-texte réalisé par les auteurs. Le lecteur a accès sensoriellement et cet objet produit dans un contexte à l'élaboration duquel il participe également, ainsi que l'a fort justement souligné Christian Vandendorpe [28, p. 76].

En d'autres termes, nous remettons en cause de ce fait l'invariance de la relation «signifiant-signifié» postulée par Saussure. D'autre part, il ne saurait y avoir de «degré zéro» de la signification en vertu d'un principe sémiologique formulé par Barthes pour qui «*tout signifie ou rien ne signifie*». En effet, un élément graphique intervient nécessairement dans un contexte ; partant, il est défini dans

les cadres de représentation du lecteur. Le registre de «*l'objectivité liée au fonctionnel*» proposé par Pierre Duplan demeure donc problématique de ce point de vue. La typographie n'est pas plus «*objective*» qu'elle n'est «*scientifique*»[4], elle se donne à voir dans un contexte textuel qui peut être scientifique ou chercher à tendre vers une certain objectivité. En revanche, aucun caractère typographique ne peut se prévaloir en soi de telles «valeurs».

Roy Harris a systématisé la remarque en proposant une «*théorie intégrationnelle* (...) *qui reconnaît un principe fondamental, celui de l'intégration contextuelle du signe*» ; théorie selon laquelle «*il n'existe pas de signe sans contexte*». Le travail du lecteur se résumant alors à un travail «*d'intégration sémiologique*» [29, p. 9–25]. Mais Christian Vandendorpe avait déjà clairement expliqué en quoi, d'un point de vue littéraire, la lecture est «*un acte auto-énonciatif au moyen duquel le lecteur met un énoncé en résonance avec les schèmes contextuels susceptibles d'en rendre compte*». Autrement dit, que «*le contexte doit être pensé comme une instance indispensable à la compréhension du langage et qui lui est étroitement associée*» [30].

S'agissant de l'écriture et plus encore de la typographie, le contexte est un jeu d'emboîtements (de poupées gigognes), qui sont autant de cadres physiques déterminant la pratique [31]. La typographie est en effet inscrite dans la dimension visuelle du texte qu'elle donne à lire à travers une mise en page. Or le texte relève également d'une «*forme texte*» générique, élément de représentation d'un niveau supérieur qui permet une première appréhension de l'objet dans l'univers informationnel. Deux dimensions essentielles le caractérisent, sa dimension textuelle (au sens linguistique du terme) et sa dimension visuelle (la mise en page). Dualité entre *forme* et *substance* [28, p. 87–91] qui nous invite à reformuler l'idée du «*sens formel*» prédéterminant la lecture de façon intégrative. Les modalités d'interprétation du texte (de la mise en page et de la typographie) vont en effet tenir compte de cette information supra segmentale qui classifie formellement les genres textuels. En outre, le texte est lui-même inscrit dans le média (contexte physique du média en tant que support, tel que la page par exemple [32]), lequel est redevable d'une pratique, d'un usage spécifiques (la lecture du journal, du livre, de l'affiche ...). Le contexte de lecture de la «lettre typographique» est donc déterminé par un ensemble de cadres de natures sémiotiques et communicationnelles distinctes. Son interprétation est le fruit d'une interdépendance entre les divers niveaux qui le constituent et de l'activité propre du lecteur. La signification de la typographie tire ainsi paradoxalement parti du contexte et des pratiques cognitives et sociales dans lesquelles elle s'intègre.

Néanmoins l'importance qu'il convient d'accorder au contexte dans la lecture d'un texte est une dimension très souvent impensée qui a pour corollaire l'effacement de «l'image du texte». À cet égard, la pratique et la culture des professionnels — les *médiateurs* du texte — ne sont en rien comparables à celles des lecteurs.

L'invisible image du texte — *apprentissage*

Lorsque les praticiens évoquent la valeur d'usage, à l'instar de Thibaudeau, celle-ci n'est pas considérée prioritairement du point de vue du lecteur, mais de celui du professionnel, du typographe, c'est-à-dire du médiateur. La remarque est d'importance car elle soulève la question complexe de la culture visuelle et de l'apprentissage de la lecture. Si l'apprentissage de l'acte de lecture-écriture — activité conjointe que l'on dénommait la «*lettrure*» à l'époque médiévale[5] —, si cette activité est considérée comme un des actes d'apprentissage fondamentaux dans nos sociétés, il en va tout autrement pour ce qui est de l'apprentissage de l'écriture industrielle, autrement dit de «l'image du texte» qui façonne notre univers textuel quotidien. Les codes visuels spécifiques à la typographie et à l'art du livre notamment, s'ils font partie de l'apprentissage des gens de métiers, sont pour bonne part ignorés du reste de la population qui, au pire, en ignore l'existence, ou au mieux, les considère comme un être-là, une donnée inhérente à l'écriture, naturalisée par son évidence même. L'écriture typographique et «l'image du texte» ne font pas partie de l'apprentissage de la lecture et de l'écriture. Nous sommes donc confrontés à un déséquilibre référentiel, car lorsque les médiateurs de l'écriture industrielle élaborent un savoir réflexif, les lecteurs s'accommodent d'un apprentissage autodidacte qui n'accède que très rarement au stade d'une lecture consciente.

D'un point de vue réflexif, l'écriture typographique, enfermée dans sa gangue «*infra-ordinaire*», pour reprendre l'expression de Perec [34], ne se donne à lire à l'usager que lorsqu'elle est extraite de l'habitude par une manifestation d'ordre publicitaire, d'où l'expression de «*typographie jaillissante*» de Pierre Duplan [18, p. 332]. Elle est alors classée dans le registre de ce qu'André Derval a appelé la «*typographie expressive*» [35, p. 89–97], à la suite de la «*lettre expressive*» de Massin [36].

Or le phénomène a pris toute son ampleur avec l'apparition de la micro informatique. Les usagers ont en ef-

4. Lorsqu'il affirme que «*cette typographie est visible, scientifique et sert à transmettre le plus facilement possible au lecteur un message textuel*», il se situe dans un cadre idéologique comparable à celui de la typographie classique qui affirme se mettre au service du texte à travers son effacement même [18, p. 335].

5. Dans la traduction qu'elle propose de l'ouvrage d'Illich et Sanders, Maud Sissung explique que «*la langue française ne possède pas d'équivalent du mot anglais* literacy, *qui désigne la capacité de lire et d'écrire*». Elle précise toutefois que le français a possédé un mot pour désigner cette capacité, «*la* lettrure, *terme que l'on rencontre dans des textes du XII^e et du XIII^e siècles et sous toutes les orthographes possibles*» [33, p. 9].

fet été directement confrontés à la conception visuelle de leurs textes. Mais s'ils ont découvert l'image du texte, ils n'en ont pas pour autant maîtrisé le fonctionnement, les règles, l'histoire non plus que l'esthétique. Le déplacement hypothétique des pratiques de métier vers les logiciels en général et les "traitements de texte" en particulier, ne cèle pas l'ignorance des usagers qui sont en outre confrontés à la dualité constitutive des écrits informatisés scindés en deux médias dissemblables. «*L'écrit d'écran*» et «*l'écrit d'imprimante*» jouent une partition distincte [37, p. 105–119] et ne relèvent pas des mêmes critères techniques, sémiotiques et esthétiques.

La typographie, et plus généralement les traces de ce que j'ai appelé «*l'énonciation éditoriale*» [26], relèvent bien du projet «*d'anthropologie endotique*» esquissé par Perec, une anthropologie qui parle des «*choses communes*» que nous ne voyons plus, aveuglés que nous sommes par leur omniprésence quotidienne [34]. Projet qui s'apparente au dessein sémiopolitique de Barthes lorsqu'il ambitionnait de «*rendre compte en détail de la mystification qui transforme*» les faits de culture «*en nature universelle*»[6].

Au fond, il pourrait ne s'agir que de re-donner à lire au lecteur le texte dans sa complexité et son intelligence à un moment où la technique l'enjoint à prendre conscience de sa culture visuelle. Les intermédiaires du texte se raréfient et les métiers dédiés à la mise en page sont relayés par des instruments logiciels ; l'usager de la micro-informatique est donc investi de nouvelles tâches dont il ne maîtrise pas nécessairement les attendus. Le travail qu'il produit n'a plus aucune commune mesure avec ce qu'était sa pratique jusqu'alors. En contrepartie, il acquiert par la force des choses une conscience de la dimension visuelle de ses écrits et de sa matérialité. L'effet est paradoxal en ce que la technique permet une prise de conscience de l'image du texte et libère un certain nombre de tâches alors que, dans le même mouvement, elle prend en charge une partie de l'élaboration textuelle par sa structuration «*architextuelle*» et aliène une part non négligeable de l'espace d'écriture [39, p. 97–107].

L'homme contemporain est entré dans une civilisation du texte où la majeure partie de ses activités sociales (de création, production, réception ...) est désormais médiatée par des outils d'ordre textuel [40, p. 100–105]. Mais il n'a pas encore pris la mesure de l'ampleur du phénomène, non plus qu'envisagé ses répercussions dans les pratiques socioprofessionnelles. L'emprise de la machine textuelle dans le tissu social est en passe de transformer en profondeur les pratiques culturelles et c'est à une véritable mutation d'ordre anthropologique que nous assistons. Mais nous entrons là dans le domaine des questions générales soulevées par les «*médias informatisés*» [41], à leur structuration et leurs pratiques [42] qui nous éloigne quelque peu de notre objet typographique.

Quoi qu'il en soit, au cours de l'activité de lecture, l'image du texte infra-ordinaire est prise dans une série d'automatismes qui relèvent de la mémoire à la long terme [43, p. 237–253]. Elle dépend donc de structures de fond signifiantes et s'inscrit par là-même dans le cadre d'une théorie du sens formel. Le distinguo opposé entre la «typographie expressive» et la «typographie infra-ordinaire» par exemple [44, p. 62–64] ne prendra sens qu'en fonction du contexte et du genre textuel qui sera donné à voir ou à lire. La publicité ou l'expression artistique pour l'une, le texte courant pour l'autre.

Toutefois, la question de la conscience, partant de l'apprentissage, demeure et relève du politique ; Brecht la posait déjà en termes méthodologiques dans *L'Achat du cuivre* : «*Pour établir des lois, il faut accueillir les processus naturels avec étonnement, pour ainsi dire, c'est-à-dire qu'il faut abolir leur "évidence" pour accéder à leur compréhension*» [45, p. 477–625].

Reste que le problème de la naturalisation de la signification de la «part typographique», ancrée dans son invisibilité infra-ordinaire, a été précédée, affirmée, validée par le discours d'escorte qui n'a cessé d'accompagner la pratique professionnelle des médiateurs de l'écrit industriel. Et c'est sur le discours de ces praticiens que j'aimerais m'arrêter à présent.

Qu'en est-il donc de l'imaginaire et des discours qui ont généralement accompagné la typographie en Occident depuis son avènement ?

Typographie servante & silence du texte

L'un des axes de compréhension de la signification du fait typographique repose sur la perception que l'on se fait de cet art selon les objectifs qui lui sont assignés. Perception physique tout d'abord, inscrite dans les routines et habitudes de lecture qui nous engage au-delà de la visualité pour accéder à «un sens». Du point de vue imaginaire, il y a, dans la conception archétypale du livre et de la lecture, une démarche d'ordre métaphysique qui consiste à vouloir accéder à un sens transcendant, fatalement extérieur au média [46]. L'écriture et le texte révèlent ainsi un savoir issu d'un au-delà de la lettre qui les constitue. Le discours de Fournier le Jeune est à cet égard exemplaire. Son célèbre catalogue de caractères est en effet placé sous un quatrain de sa façon qui a valeur de manifeste :

> *Des faits éloignés de nos yeux*
> *Ces Caractères nous instruisent,*
> *Et par cet Art ingénieux,*
> *Tous les talens s'immortalisent*

Les caractères typographiques renvoient à des «*faits éloignés de nos yeux*», c'est-à-dire à un au-delà d'eux-mêmes. Le typographe joue du paradoxe, ses caractères

6. [38, p. 7–8]. L'auteur parle d'une «*sémio-clastie*», c'est-à-dire d'une sémiologie de classe, au sens social du terme.

ont cette qualité particulière qui consiste à pouvoir être transparents, à s'abstraire de la matérialité, de l'encre et du papier, tout en immortalisant les «*talens*», c'est-à-dire en matérialisant les «*faits*» évoqués à l'instant. La conception que nos contemporains se font de «l'écrit d'écran» à travers la thématique de «*l'immatérialité*» repose sur le même paradoxe ou le même déni de la matérialité de l'écrit et des dispositifs techniques et médiatiques qui le donnent à lire.

Le principe de la «*typographie servante*» a été clairement exprimé par Maximilien Vox qui affirmait que «*l'art du typographe est de rester inaperçu*», ajoutant aussitôt «*si le lecteur s'exclame …oh ! la belle typographie ! c'est que le texte passe après*». Rester inaperçu, passer après le texte …voici les commandements de cet art. Mais qu'on ne s'y trompe pas, de telles vertus reposent sur une conscience aiguë de l'influence de l'art typographique et de sa signification. Elle l'inscrit dans un projet idéologique déterminé qui puise son pouvoir de l'infra-ordinaire. Maximilien Vox, en bon professionnel, sait la part signifiante de son art. Il défend donc sa pratique d'un point de vue politique. Il s'agit pour lui d'influencer, de prendre un pouvoir symbolique sur les esprits. Militant, il écrit en 1943 : «*dans le monde de demain, grande sera la place de la nation qui aura su créer un style*». Une vingtaine d'années plus tard, toujours dans le même registre, à la question «*y a-t-il une typographie française ?*» il répond «*oui, mais elle ne se voit pas*» contrairement à celle d'autres nations. En d'autres termes, elle remplit sa fonction : être au service du texte et disparaître derrière lui. Nous sommes loin des années 1930 où selon Vox débutait avec la «linéale» «*l'invasion fonctionnaliste de la Neue Typographie propagée par le Bauhaus de Dessau*» [47, p. 186], [48, p. 117] et [50, p. 14], mais il est vrai que le Bauhaus n'était pas alors en odeur de sainteté dans certains milieux …

Les praticiens plongent la typographie dans les silences de l'infra-ordinaire. La culture typographique naît de ce silence. Aussi, tout écart pour le texte courant est-il rigoureusement proscrit, car il suffit à révéler la présence condamnable du «*corps*» de la lettre au sein de l'espace de la page. Le vocabulaire corporel lié à la lettre est du reste révélateur des champs sémantiques mis en cause. La lettre porte en elle-même un pouvoir signifiant qu'il convient de cerner. Nous n'avons pas abandonné la pensée magico-religieuse du *Cratyle*, loin s'en faut. Les sémiologues lui ont simplement substitué la plus raisonnable «connotation». Conscient du pouvoir signifiant des caractères d'imprimerie, Henri Fournier bannit ainsi les «*caractères de fantaisie*» pour les textes classiques ; deux cultures, l'une visuelle l'autre textuelle, qui ne pouvaient que s'affronter : «*Nous recommandons en tout cas de ne les employer dans le cours des volumes qu'avec réserve et discernement, et de les exclure avec rigueur de tout ouvrage classique, et de tous les textes où ils formuaient un contraste choquant avec la sévérité des matières.*» [51, p. 50]

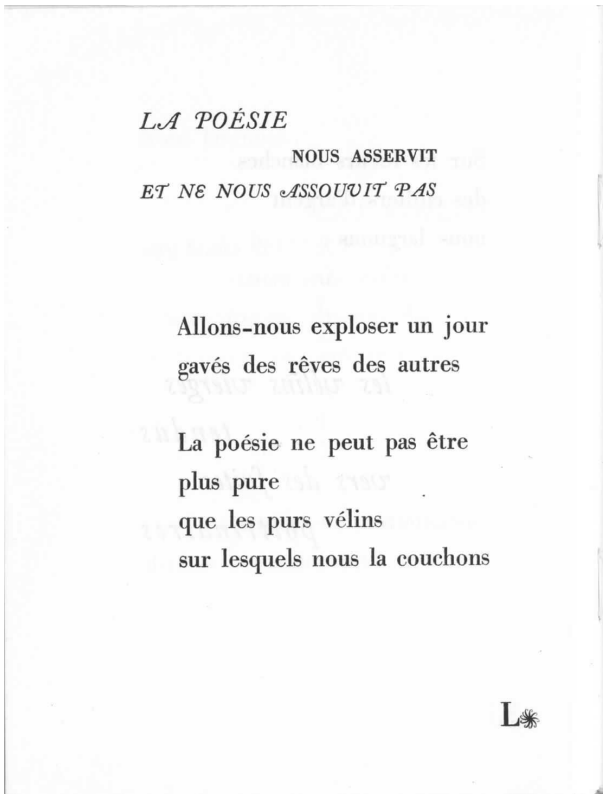
Lorsqu'il propose les règles d'une «*typographie logique*» qui «*repose sur l'analyse logique du texte*», François Richaudeau ne déroge pas à la règle. Cette «*mise en page foisonnante*» se donne en effet pour but «*de hiérarchiser les diverses ressources de composition (cops, graisse, justification, interligne) afin de mettre en valeur le message et lui seul*» [50, p. 15]. D'une phrase, il exclut la «*typographie expressive*» qui avec le phénomène des «*Clubs de livres*» accompagna la renaissance du livre après guerre. Toutefois le travail des Clubs et de ses «*maîtres typographistes*» se démarquait de la typographie servante sans pour autant s'y opposer. Pour Bernard Gheerbrant, il fallait que la «*typographie du livre-club exprime le texte*» ; toutes les ressources matérielles et visuelles furent donc

7. Ouvrage collectif dédié au maréchal Pétain et introduit par Charles Maurras. En note 1, p. 182, l'auteur renvoie à un album illustré, *Nouveaux destins de l'Intelligence Française*, en précisant : «*publié par le ministère de l'Information, sous la direction d'Henri Massin et de Maximilien Vox*».

mises en œuvre dans une optique expressive [52, p. 48]. Démarche analogue à la recherche expressive de l'émotion actuellement prônée des typographes tels que David Carson, pare exemple [53].

En France, rares furent les périodes d'explosion créatrice comme celles des Livres clubs où les éditeurs cherchèrent avec autant d'audace et de justesse l'harmonie des textes et de leur image. Plus généralement, dans l'ordre du texte, de génération en génération, la corporation des typographes transmet une vision logocentrique qui a ceci de particulier qu'elle interdit au média visuel de prendre sa pleine expression. Placé dans le carcan linguistique, lorsque la typographie s'affranchit de la ligne pour s'élancer à travers l'espace la page, c'est pour retrouver une articulation du discours calquée sur la rhétorique classique, c'est-à-dire sur l'oral. La *dispositio* typographique se coule alors dans la *dispositio* rhétorique. François Richaudeau pensait qu'une fois érigée en système, la *dispositio* typographique devait imposer ses normes au texte lui-même. Une question demeurait toutefois, «*les auteurs accepteraient-ils de se plier, de plier leur talent aux servitudes rigoureuses d'une mise en page logique ?*» [49, p. 15]. Mais pouvait-il y avoir une réponse ? Les artistes, peintres et écrivains, n'ont pas attendu le dogme pour se jouer de cette maîtresse servante qu'est la typographie.

Qui se souvient des trois typographes — Guy Levis-Mano, Georges Duchêne & Roger Bonon — qui en décembre 1934 déclarent en avoir «*marre*» d'être au service d'une poésie qui ignorait que sans typographie, elle n'aurait su exister [54] ? Levis-Mano écrivait alors :

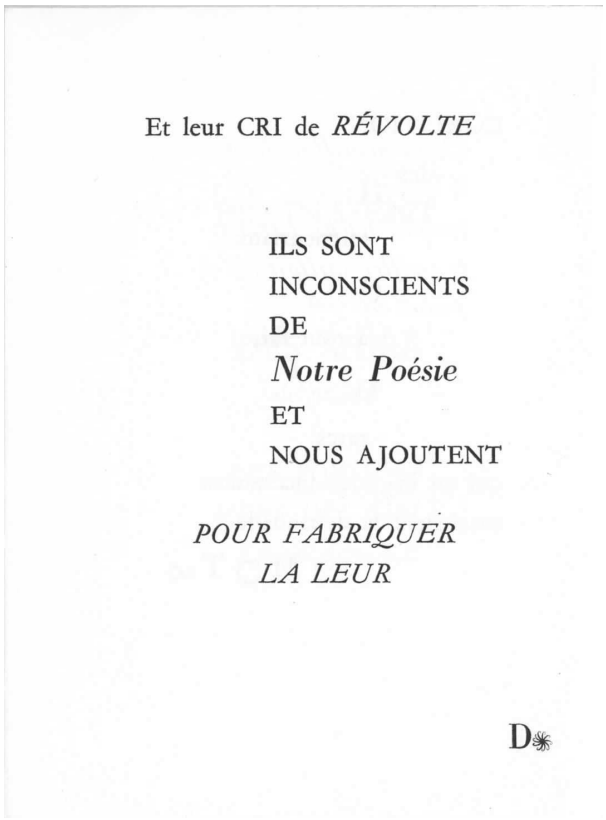

LA POÉSIE
NOUS ASSERVIT
ET NE NOUS ASSOUVIT PAS

Allons-nous exploser un jour
gavés des rêves des autres

La poésie ne peut pas être
plus pure
que les purs vélins
sur lesquels nous la couchons

L*

Et c'est aux poètes qu'il s'adressait lorsqu'il composait la page D* de ce recueil collectif :


Et leur CRI de *RÉVOLTE*

ILS SONT
INCONSCIENTS
DE
Notre Poésie
ET
NOUS AJOUTENT

POUR FABRIQUER
LA LEUR

D*

Kassak, le théoricien du constructivisme hongrois, semblait avoir répondu aux propos de Guy Levis-Mano, lorsqu'il écrivait qu'«*il n'y a pas de poème sans sa traduction typographique*». Sans entrer dans l'exégèse du mot «*traduction*» qui soulève à nouveau la question d'un au-delà du texte, notons que rares furent les théoriciens qui partagèrent ce point de vue associant aussi intimement le texte à sa typographie. D'autant plus rares que ceux qui s'en firent les plus fervents défenseurs retombèrent fréquemment dans un logocentrisme de bon aloi. Ainsi, après avoir exposé les théories de Kassak, Jérôme Peignot conclut qu'«*en matière de typographie le dernier mot revient à ceux qui écrivent, aux écrivains et, sans doute plus encore, aux poètes*» [55, p. 306].

De retour au texte, donc. Reste que, derrière ces affirmations, les rapports entre le texte et la typographie s'expriment toujours en termes de rapports de pouvoirs. Seule la façon d'exprimer la relation semble devoir évoluer. Ainsi des éditions Actes Sud qui proposent cette autre version de la typographie servante en glissant vers le «*livre rempart du texte*». Dans l'un de ses catalogues,

l'éditeur précise que «*le livre est en quelque sorte le rempart du texte, ce qui l'entoure, le protège et en même temps le propose au lecteur, tel un objet digne de ce qu'il contient*» [56, p. x].

La métaphore de l'objet défensif et le rapport de contenant à contenu interdisent toute relation de signification, toute existence propre. Le livre et par là-même tous les éléments qui le constituent sont des objets dont la «*dignité*» se mesure à l'aune du texte. Mais n'est-ce pas précisément oublier que le texte n'existe qu'à travers ces «*objets*» ?

L'idéal typographique est né d'un dilemme et d'un rêve : associer l'œil et la voix. Mais il fallait pour cela réunir le corps et l'esprit, le sens et la matière. Or notre culture a fait du verbe le maître de l'écrit, aussi était-il logique de voir répercutée cette dualité dans la conception du couple texte-typographie. Toutefois, théoriciens et praticiens semblent avoir longtemps ignoré l'une des caractéristiques essentielles de la lettre typographique. Non pas celle liée à sa pratique rhétorique, mais bien celle qui a trait à son statut sémiologique.

Sortie de son rôle servant, la lettre change de statut selon les usages, le contexte d'utilisation et le point de vue du lecteur. Ainsi, l'image publicitaire use à loisir des différents statuts sémiologiques accordés à ses éléments constitutifs [57, p. 36–51].

Le fait que la typographie ait dégagé «*la lettre de sa fluidité cursive pour lui donner un statut d'objet indépendant*» [58, p. 168] lui a permis d'acquérir un statut d'illustration ou d'idéogramme. À l'instar des hiéroglyphes égyptiens, elle peut donc, au sein de la même page, selon l'usage qui en est fait, relever de l'alphabétique, de l'idéographique ou de l'illustration [59], [60, p. 45–69], [61, p. 60–65]. En ce sens, on peut légitimement parler d'une «écriture typographique». Ainsi pourra-t-on redonner à cet art qui révèle le texte industriel et le fait exister, toute sa dimension expressive et signifiante.

Voix du pouvoir

Pierre Faucheux écrivait que «*la beauté d'une page provient d'abord de sa lisibilité ; de ses proportions, du caractère choisi*», moyens nécessaires pour arriver à un «*résultat esthétique*» qu'il définissait en ces termes : «*l'accord parfait entre la signification du texte et la forme typographique adoptée*». Raymond Gid n'assignait précisément pas d'autre «*vocation*» à la typographie que cet «*accord parfait avec l'expression de l'écrit*». Marcel Jacno précisant quant à lui qu'«*à chaque message correspond un choix limité de modes d'expression*» et qu'il revient «*au typographe de découvrir ces affinités*» [62, p. 167–168], [4, p. 8], [16, p. 78]. Loin de faire l'unanimité, cet «*accord*» est souvent l'objet d'enjeux de pouvoirs exacerbés dans l'espace des textes.

Reste que le discours qui a accompagné la typographie tout au long de son existence a réduit ses capacités aux préoccupations idéologiques de la parole. Les lettres ne pouvaient donc compromettre «*le sort de la langue au service de laquelle elles se sont mis*» [63, p. 135]. Paradoxe étonnant que cette «image de texte» qui annonce sa servitude, sa fidélité à la parole en revendiquant sa force du silence. La typographie semble avoir passé un pacte secret avec le texte écrit qui lui doit existence. Mais si dans ce mouvement la part iconique de la lettre se nie elle-même, c'est pour mieux régner sur le silence de la page. Ainsi les serviteurs deviennent-ils parfois des maîtres …

Références

[1] Isabelle Garron, «La part typographique. Pour une anthropologie de la page imprimée», *Communication & langages*, n° 134, 2002.

[2] André Leroi-Gourhan, *Le geste et la parole*, Paris, Albin Michel, 1965–1966.

[3] Groupe μ, *Traité du signe visuel. Pour une rhétorique de l'image*, Paris, Éd. du Seuil, 1992.

[4] Raymond Gid, «À l'heure où le plomb devient lumière», *De plomb d'encre et de lumière. Essai sur la typographie & la communication écrite*, Paris, Imprimerie nationale, 1982.

[5] François Thibaudeau, *La lettre d'imprimerie et 12 notices sur les Arts du Livre*, Paris, Au Bureau de l'Édition, 1921, Tome second.

[6] Maximilien Vox, *Faisons le Point. Cent alphabets 'Monotype'*, Paris, Librairie Larousse, 1963.

[7] Gérard Blanchard, *Aide aux choix de la typographie*, Reillane, Atelier Perrousseaux Éd., 1998.

[8] Ladislas Mandel, «Propos sur l'ordre typographique», *Communication & langages*, n° 130, 2001.

[9] Ladislas Mandel, *Écritures Miroir des hommes et des sociétés*, Reillane, Atelier Perrousseaux Éd., 2000.

[10] Jacques Roubaud, *La Fleur inverse, essai sur l'art formel des troubadours*, Paris, Ramsay, 1986.

[11] Emmanuël Souchier, «Introduction» au *Traité des vertus démocratiques* de Raymond Queneau, Gallimard, 1993.

[12] Elizabeth Lavault, *Forme et mémoire d'une contrainte. Poïé6 de la sextine dans les romans d'Hortense de Jacques Roubaud*, Thèse de doctorat de l'Université Paris 7 Denis Diderot, 2002.

[13] Jean-Marie Durand, «Espace et écritures en cunéiforme», *Écritures. Systèmes idéographiques et pratiques expressives*, Le Sycomore, 1982.

[14] Samuel Noah Kramer, *L'histoire commence à Sumer*, Artaud, 1975.

[15] Roger Chartier, *Culture écrite et société, L'ordre des livres (XIV^e-XVIII^e siècle)*, Albin Michel, 1996.

[16] Marcel Jacno, *Anatomie de la lettre*, Paris, Compagnie Française d'Édition, 1978.

[17] Emil Ruder, *Typographie. Un manuel de création*, Bâle, Verlag Arthur Niggli Teufen AR, 1967.

[18] Pierre Duplan, «Pour une sémiologie de la lettre», *L'espace de la lettre*, Cahiers Jussieu n° 3 — Université Paris 7, Paris, UGE, 10/18, 1977.

[19] Jean Allessandrini, «Une nouvelle classification typographique : *Codex* 80», *Communication & langages*, n° 43, 1979.

[20] Gérard Blanchard, *Pour une sémiologie de la typographie*, Belgique, Éd. Magermans, 1979.

[21] Jacques Bertin, *La Sémiologie graphique*, Paris, Gauthier-Vilars, 1967.

[22] André Malraux, *Le Musée imaginaire*, Paris, Gallimard, 1965.

[23] Roland Barthes, *Le système de la mode*, Paris, Éd. du Seuil, 1967.

[24] Abraham Moles et Elisabeth Rohmer, *Les Sciences de l'imprécis*, Paris, Éd. du Seuil, 1990.

[25] Bernard Cathelat, *Socio-Styles système : les styles de vie, théorie, méthodes, applications*, Éditions d'Organisation, 1990.

[26] Emmanuël Souchier, «L'image du texte. Pour une théorie de l'énonciation éditoriale», *Cahiers de médiologie*, n° 6, 1998.

[27] Emmanuël Souchier, «La manchette de L'Équipe», *Communication & langages*, n° 51, 1982.

[28] Christian Vandendorpe, *Du papyrus à l'hypertexte. Essai sur les mutations du texte et de la lecture*, Paris, Éd. de la Découverte, 1999.

[29] Roy Harris, *La sémiologie de l'écriture*, Paris, CNRS éd., 1993, p. 137, 151.

[30] Christian Vandendorpe, «Contexte, compréhension et littérarité» *RSSI*, vol. 11, non 1, 1991.

[31] Annette Béguin-Verbrugge, *La discipline du lisible. Rôle des dispositifs spatiaux dans l'acte de lecture*, Habilitation à diriger des recherches, Université Charles de Gaule — Lille 3, 2002.

[32] Emmanuël Souchier, «Histoire de pages & pages d'histoire», *L'Aventure des écritures, La Page*, Anne Zali (sous la dir. de), Paris, Bibliothèque nationale de France, 1999.

[33] Ivan Illich et Barry Sanders, *ABC : l'alphabétisation de l'esprit populaire*, trad. Maud Sissung, Paris, Éd. La Découverte, 1990.

[34] Georges Perec, *L'infra-ordinaire*, Paris, Éd. du Seuil, 1989.

[35] André Derval, «Typographies expressives», *Massin*, Paris, IMEC Éditions, 1990.

[36] Massin, *La lettre et l'image*, Paris, Gallimard, 1973.

[37] Emmanuël Souchier, «L'écrit d'écran. Pratiques d'écriture et informatique», *Communication & langages*, n° 107, 1996.

[38] Roland Barthes, *Mythologies*, Paris, Éd. du Seuil, [1957] 1970, p. 7–8.

[39] Yves Jeanneret et Emmanuël Souchier, «Pour une poétique de l'écrit d'écran», *Xoana*, n° 6/7, 1999.

[40] Emmanuël Souchier et Yves Jeanneret, «Écriture numérique ou média informatisé ?», *Pour la Science, Du signe à l'écriture*, Dossier hors série, n° 33, 2001–2002.

[41] Yves Jeanneret, *Y a-t-il (vraiment) des technologies de l'information ?* Presses universitaires du Septentrion, 2000.

[42] Emmanuël Souchier, Yves Jeanneret et Joëlle Le Marec (sous la dir. de), *Lire, écrire, réécrire : objets signes et pratiques des médias informatisés*, à paraître, BPI - Beaubourg, «Études et recherches», 2003.

[43] Christian Vandendorpe, «La lecture entre déchiffrement et automatisation», Denis Saint-Jacques (sous la dir. de), *L'Acte de lecture*, Québec, Éd. Nota Bene, 1994 et 1998.

[44] Emmanueël Souchier, «Raymond Queneau. *Exercices de style*. Prélude. Voix du silence», *Massin*, Paris, IMEC Éditions, 1990.

[45] Bertold Brecht, *L'Achat du cuivre*, Écrits su le théâtre, Paris, L'Arche éd., 1967, vol. 1.

[46] Yvonne Joahannot, *Tourner la page. Livre, rites et symboles*, Grenoble, Jérôme Millon éd., 1988.

[47] Maximilien Vox, «Vers un style français», *La France de l'esprit* 1940–1943, Sequana éd., 1943.

[48] Maximilien Vox, *Faisons le point. Cent alphabets 'Monotype' pour servir à l'étude de la typographie du demi-siècle*. Union bibliophile de France, Larousse, 1963.

[49] François Richaudeau, *La lettre et l'esprit, Vers une typo-graphie logique*, «Présence», éd. Planète, 1965.

[50] Maximilien Vox, «Préface» à [49].

[51] Henri Fournier, *Traité de la typographie*, Garnier Frères éd., 1904.

[52] Bernard Gheerbrant, *Le Club des Libraires de France* 1953–1966, IMEC Éditions, 1997.

[53] Christian Vandendorpe, «Rhétorique de la typographie. David Carson et la visibilité du signe», *Communication & langages*, n° 137, 2003, à paraître.

[54] Guy Lévis Mano, Georges Duchêne et Roger Bonon, *Trois typographes en avaient marre*, GLM éd., [1935] 1967, paginé de A à Z.

[55] Jérôme Pelta, «L'esprit de la lettre», *De Plomb d'encre & de lumière. Essai sur la typographie & la communication écrite*, Centre d'Étude et de Recherche Typographiques, Éd. de l'Imprimerie Nationale, 1962.

[56] Actes Sud, «Mode d'emploi» *Catalogue* 1996, Éd. Actes Sud, Arles, 1996.

[57] Emmanuël Souchier, «La publicité comme détournement du politique», *Communication & langages*, n° 93, 1992.

[58] Anne-Marie Christin, *L'image écrite ou la déraison graphique*,Paris, Flammarion, 1995.

[59] Pascal Vernus, «L'écriture de l'Égypte ancienne», *L'espace de la lettre*, Cahiers Jussieu n° 3 — Université Paris 7, Paris, UGE, 10/18, 1977.

[60] «Des relations entre texte et représentation dans les textes de l'Egypte pharaonique», *Écritures II*, Le Sycomore éd., 1985.

[61] «L'ambivalence du signe graphique dans l'écriture hiéroglyphique», *Écritures III. Espaces de la lecture*, Paris, Éd. Retz, 1988.

[62] Pierre Faucheux, *Écrire l'espace*, Robert Laffont éd., 1978.

[63] Jérôme Peignot, *De l'écriture à la typographie*, Paris, Gallimard, 1967.

Système automatisé de co-rédaction de livres

Yves Maniette
UNESP, Araraquara, São Paulo, Brésil
yves@iq.unesp.br
http://sibee.iq.unesp.br/~yves

Abstract

Producing a book that requires the collaboration of dozens or hundreds of people, as for a conference, is known to be a major challenge. Every organizer should be able to check every article, or every author be able to follow the relevant typographic rules or corresponding coding. On the other hand, the ubiquitous availability of ever-improving desktop publishing (DTP) programs has convinced everyone that their text is already "camera ready", whereas most of the time, achieving good typography means that the typesetting has to be re-done, from scratch, thereby casting doubt on the validity of such DTP programs. In this paper we present a system that will perform the appropriate typesetting without any special typographic knowledge required of the writers. Thus, every author is able to use their everyday word-processing program to produce their text, because most of the detailed typographic work will be done silently, eventually resulting in a master file for the entire book, including any relevant indices.

Résumé

La rédaction d'un ouvrage produit en collaboration par plusieurs dizaines ou centaines de personnes, par exemple à l'occasion d'un congrès, n'est pas aisée. Elle exige des organisateurs les corrections requises dans chaque article, ou bien de chacun des rédacteurs le respect de règles précises de typographie ou de la codification correspondante. Par ailleurs, l'emploi généralisé de logiciels de mise en page chaque jour plus sophistiqués fait que nombreux sont ceux qui pensent produire un texte «bon à tirer», alors que bien souvent une typographie soignée exige que la mise en page soit entièrement refaite, ce qui met en cause le bien fondé de ces logiciels de mise en page. Nous présentons un système qui tient compte des nécessités d'une mise en page de qualité sans demander de grandes compétences aux rédacteurs, qui peuvent ainsi rédiger le texte de leur communication avec leur logiciel habituel. La plupart des traitements typographiques seront réalisés de manière transparente, jusqu'à produire le fichier maître du livre, y compris les annexes.

Introduction

De nombreuses sociétés savantes organisent leur congrès annuel par roulement : à tour de rôle, un groupe de membres se charge des multiples tâches administratives et techniques d'organisation du congrès. La plupart du temps, ce groupe a des connaissances et une disponibilité réduites. Une des tâches les plus difficiles à mettre en œuvre est justement la préparation du livre des actes. C'est sans doute la raison pour laquelle bien souvent les organisateurs de congrès optent pour la préparation d'un disque contenant les textes sous divers formats. Il n'est pas rare, toutefois, que lors de la première consultation du disque après la réunion on constate que certains fichiers ou illustrations ne s'ouvrent pas ou demeurent invisibles. Ce genre de disque contient aussi, bien souvent, un index automatisé qui permet par exemple de retrouver tous les textes contenant le mot «the» ou le caractère «;» ou «#» ! En revanche, il est parfois impossible d'y localiser un article avec plusieurs mots-clé, comme par exemple «cuivre», «protection» et «oxydation».

Pour de telles raisons, il est extrêmement important de pouvoir offrir à chaque participant un livre des actes du congrès dès le premier jour de la réunion. Idéalement, ce livre doit être typographiquement soigné et comprendre des index permettant de retrouver rapidement un texte quelconque ou l'adresse d'un correspondant.

Je me suis trouvé il y a un peu plus d'un an membre du groupe de préparation du Symposium brésilien d'électrochimie et électroanalytique (SIBEE), et bien que je ne sois ni chimiste ni Brésilien, je me suis spontanément proposé pour participer à la préparation du congrès et aider à l'élaboration des annales. C'est pourquoi je participe à la présente réunion, afin de faire part de mes réflexions.

Aspect logiciel et financier

Le budget initial du congrès étant modeste, il devait être possible d'en commencer l'organisation sans que cela n'entraîne de frais particuliers, ni en logiciels ni en matériel. Cette décision a aussi été motivée par la volonté

de pouvoir mettre à disposition d'autres groupes de recherche brésiliens un outil typographique et de gestion de documents qui soit librement utilisable. De plus, sachant que, hormis dans certaines grandes entreprises ou administrations, la plupart des ordinateurs fonctionnant au Brésil le font sans license particulière, que les budgets alloués à l'achat de logiciels sont souvent dérisoires et que les logiciels libres y sont paradoxalement peu connus et inspirent la méfiance, il est apparu que l'emploi de logiciels libres s'imposait d'emblée.

Typographie et détails du livre

Le livre des actes devait contenir environ 300 documents de 2 à 3 pages contenant des équations mathématiques et chimiques, des références bibliographiques, des illustrations de tailles variables et parfois formées de plusieurs parties, ainsi que les index par auteurs, mots clés et titres. L'outil typographique devait permettre d'inclure aisément ces divers éléments, de classer les divers textes en temps réel jusqu'au dernier moment, quelques heures avant de transmettre le fichier final à l'imprimerie.

Les solutions existantes

Une recherche des produits existants a montré que n'existait guère que des logiciels fonctionnant sous license d'exclusivité. De plus, la plupart du temps originaires des EUA, ceux-ci n'offraient que peu de libertés quant à la typographie ou le traitement des langues. Par exemple, les responsables de l'Initiative de Budapest pour le libre accès à la recherche ont publié un recensement des logiciels destinés à publier des journaux scientifiques[1]. Toutefois, la plupart de ces programmes ne sont pas de code ouvert. Par ailleurs, un logiciel français employé dans de nombreux magazines en ligne, `SPIP`[2], a des atouts très intéressants mais n'offrait pas la composition des mathématiques ou de la chimie au moment où nous devions décider quel logiciel employer. `SPIP` a toutefois subi de notables améliorations très prometteuses.

En tout état de cause, j'ai créé un ensemble capable d'assurer au moins les fonctions suivantes :

- inscription des participants ;
- inscription de chaque article dans un chapitre déterminé ;
- réception des textes ;
- réception des illustrations ;
- traitement des illustrations en temps réel ;
- production de chaque article en temps réel ;
- aide à la typographie de la chimie ;
- production de transparents ou panneaux à partir du même document que l'article ;
- production du livre à tout moment.

1. Voir notamment `http://www.soros.org/openaccess/fr`
2. `http://www.spip.org`

Logiciels employés

Le serveur : logiciels employés en aval Le centre du système est un serveur programmé en script `php` qui fait appel à plusieurs logiciels, dont le principal est TEX. `Ghostscript` est employé pour créer les fichiers au format `pdf`, et `jpeg2ps`[3] pour transformer automatiquement en fichiers `.ps` les illustrations reçues au format `jpeg`.

Rédaction : logiciels employé en amont La grande majorité des participants au congrès, chimistes brésiliens, tout comme de nombreux scientifiques du monde entier, produisent leurs textes en employant un logiciel commercial fort connu dont le nom tient en un mot, et ne connaissent même pas d'autre logiciel. C'est dire que pour nombre d'entre eux, la question que pose Hans Hagen dans sa communication de ce congrès n'a même pas lieu d'être : au Brésil, de nombreuses sociétés savantes n'acceptent que les documents produits par «Word for Windows».

Recherche d'un compromis Il y a donc une apparente contradiction entre ma volonté d'employer des programmes qui soient en règle et accessibles à tous pour faire fonctionner le serveur, et la réalité des chercheurs attachés à un outil avec lequel ils produisent la majorité de leurs travaux mais dont les capacités typographiques ou de programmation sont limitées ; de plus, il est pratiquement inutilisable pour traiter de gros documents en toute sécurité.

Pour cette raison, le serveur devait pouvoir accepter des textes rédigés avec Word, mais pouvoir les faire traiter par TEX. Pour ce faire, il a simplement été demandé à chaque participant de placer quelques codes simples dans le texte et de le sauvegarder au format de texte simple (`.txt`) avant de l'envoyer au serveur pour la compilation en direct.

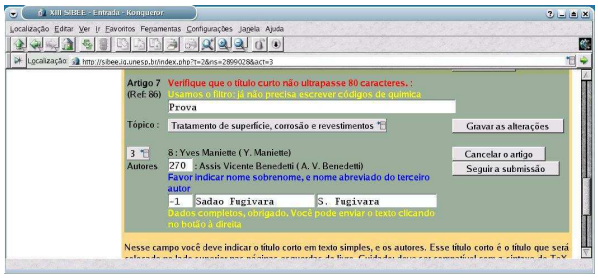
Commandes simplifiées Compte tenu de ce qui précède, des commandes extrêmement simples ont été créées, pour ne pas compliquer la tâche des rédacteurs. Elles comprenaient pour la plupart deux lettres majuscules et le symbole d'échappement. Par exemple, le titre de l'article est défini par :

```
\TR Titre du travail
```

suivi d'une ligne vierge. La macro utilise aussi d'autres variables créées automatiquement lors de l'enregistrement de l'article sur le serveur, qui définissent les noms des auteurs et de leurs institutions, ainsi que le titre court destiné à apparaitre en haut des pages de gauche. Cette option est a priori dangereuse car les noms des auteurs ne sont pas enregistrés sur le même document que l'article. Toutefois, comme toutes les manipulations sont faites automatiquement, cette crainte n'est pas justifée, et surtout,

3. Logiciel permettant d'enrober un fichier `jpeg` pour en faire un document PostScript, par Thomas Merz.

cette disposition permet d'éviter des erreurs sur les noms et d'aider à préparer l'index des auteurs. Présent ou non au congrès, chaque auteur doit s'inscrire sur le serveur, qui l'identifie de façon unique. On évite ainsi tout problème d'homonymie, et pour le responsable de l'article, il suffit d'indiquer les numéros d'inscription de chacun des co-auteurs pour qu'apparaissent automatiquement leurs noms et adresses sur le document. Voici par exemple la zone d'enregistrement d'un article :



Le nom du premier auteur ne peut pas être modifié, et les autres auteurs sont indiqués par leur numéro d'inscription, bien qu'il soit aussi possible de faire apparaître un auteur avec le numéro `-1` si, par exemple, il ne juge pas utile d'apparaître dans l'index, et donc ne s'inscrit pas. En pareil cas l'auteur principal doit remplir les champs du nom et du nom abrégé.

Le résumé de l'article vient après le titre. Il est composé sur la largeur totale de la page, à la suite de quoi le système bascule automatiquement en mode de deux colonnes. Tout cela est réalisé par la commande

```
\SA Voici le résumé...
```

placée au début du texte et suivie d'une ligne vide. La macro copie aussi verbatim dans un fichier annexe le résumé («Sinopse do Artigo», en portugais). TEX est alors prêt à recevoir le corps de l'article. De façon semblable :

```
\ABA Here is the Abstract...
```

placé n'importe où dans le texte et lui aussi suivi d'une ligne vierge copie le résumé en anglais dans un second fichier auxiliaire. Ce résumé sera composé sur une page unique placée à la fin de l'article, pour que le rédacteur puisse le relire. Les résumés en anglais devaient ensuite être exploités pour catalogage dans les *Chemical Abstracts*.

Une série de commandes comparables permet de créer des titres de paragraphe (`\TI`), sous-paragraphes (`\TSI`), listes numérotées ou alphabétiques, etc.

Un habitué de LATEX travaillant avec un éditeur adapté à ce logiciel pourra sourire en voyant de tels codes et pourra se demander pourquoi «réinventer la roue», car il lui est extrêmement facile d'écrire des commandes plus longues. En revanche, pour un usager de Word, de telles commandes pourtant simples sont déjà difficiles à lire, et l'option de correction automatique peut même perturber le texte, car les noms de certaines macros ne signifiant rien en portugais — ils sont «corrigés» automatiquement.

Langues La langue de base du congrès est le portugais mais souvent les références sont rédigées en anglais, ou en latin dans certains cas. Par ailleurs, certains participants venus d'Argentine ont rédigé leur communication en espagnol.

Le système a donc été compilé pour pouvoir traiter une dizaine de langues répondant aux cas de figures les plus probables, et une série de macros permet de basculer d'une langue à l'autre afin d'employer le dictionnaire de coupures adéquat. Enfin, certains textes composés automatiquement, comme par exemple les mots «figure» ou «tableau» ou «Références» sont définis en fonction de la langue de base de l'article, sans que l'auteur n'ait à s'en soucier.

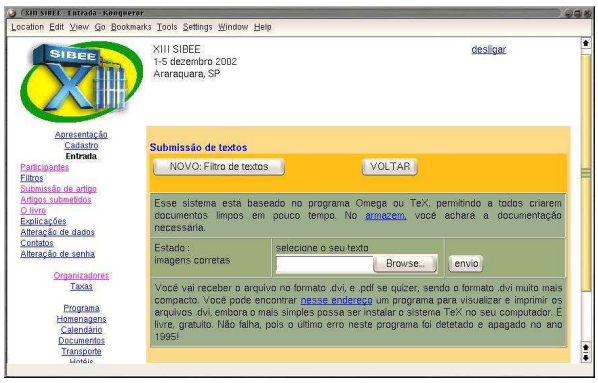
Illustrations Chaque illustration est indiquée par la commande suivante :

```
\FIG légende de la figure.[référence]
```

où la `référence` est un mot-clé employé par le système pour nommer le fichier de l'illustration correspondante, et qui permet aussi de définir la commande `\figure{référence}` comme étant égale au numéro correspondant de figure.

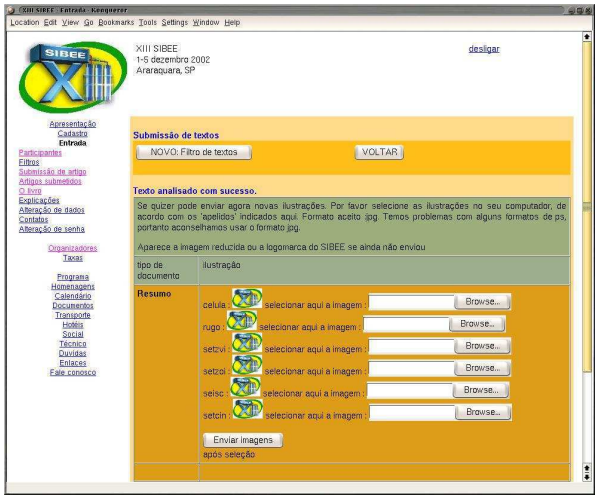
Double compilation

Afin de pouvoir recevoir correctement les figures et de pouvoir y faire référence y compris avant leur apparition, chaque fichier est traité deux fois par le système, de manière transparente pour l'usager. Avant la première passe, le visiteur doit simplement fournir au système le fichier du texte de son document :

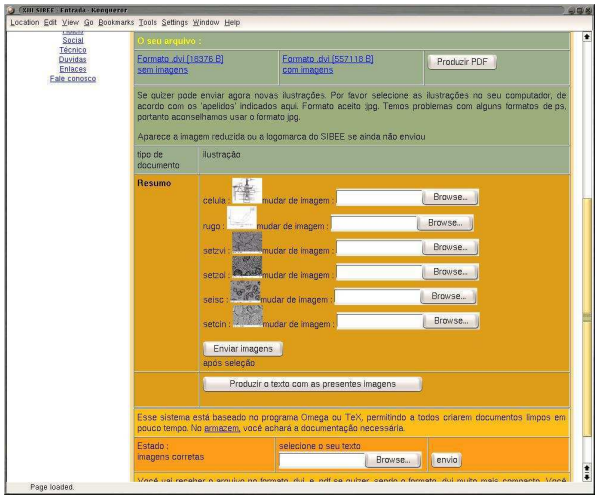


À ce moment, le script commande la compilation du fichier par TEX, en utilisant une première série de commandes. Dans cette série, la plupart des commandes de mise en page sont réduites au strict minimum, mais les commandes de figures sont exploitées afin que TEX prépare deux fichiers auxiliaires. Le premier contient les définitions lui permettant de connaître le numéro de chaque

figure, et le second est destiné à être de nouveau exploité par le script `php`. Ce second fichier comprend, pour chaque illustration, une ligne du type :
`$figr[1]="référence" ; ++$totfigr ;`
signifiant qu'à la variable `$figr[1]` on assigne la valeur `"référence"` et que l'on doit incrémenter le total du nombre de figures. Si les figures requises sont déjà présentes sur le serveur, la seconde compilation, menant à la production du fichier `.dvi`, est conduite automatiquement. En cas contraire le serveur présente une page telle que la suivante, montrant les `références` des figures nécessaires, et un logotype «par défaut» quand elles sont absentes.



Chaque figure arrivant au serveur est transformée instantanément en fichier PostScript avec le programme `jpeg2ps` de Thomas Merz, et ce fichier PostScript est ensuite exploité immédiatement par Ghostscript pour former un second fichier `jpg` de basse résolution employé pour montrer l'onglet correspondant à l'auteur.



En tout état de cause, on peut à tout moment envoyer une nouvelle version du texte et de nouvelles versions des figures (au format `jpeg`). Les versions nouvelles du texte ou des figures viendront écraser les anciennes et les numéros de figures seront mis à jour instantanément. Aucune manipulation humaine n'est nécessaire et la présence de l'onglet sur le serveur prouve que le fichier pourra être compilé sans problème, au moins pour ce qui concerne les illustrations.

Chimie

Le paquet le mieux adapté qui permette d'écrire des symboles chimiques assez aisément est probablement `ppchTeX`, groupe de macros de ConTEXt qui est compatible avec TEX et que l'on installe en chargeant les fichiers `pictex.tex` et `m-ch-en.tex`.

«Lavoisier» : le filtre Sachant que les textes sont envoyés au serveur sous la forme de texte selon le codage ISO 8859-1, on perd une quantité considérable d'information quant aux termes placés en italiques ou en exposants. Les commandes permettant de placer des italiques ont été indiquées aux auteurs et, dans un premier temps, une série de commandes correspondant à divers composés chimiques ou unités a été préparée. Cependant, les auteurs étant peu habités aux textes à compiler, il est apparu qu'une solution plus simple et efficace était absolument nécessaire.

Un des aspects les plus ergonomiques du logiciel `SPIP` évoqué plus haut se trouve dans les «raccourcis SPIP» qui codifient simplement l'usage de l'italique, du gras ou la notation des liens hypertexte. Ces «raccourcis SPIP» sont destinés à être traités par un script `php` qui utilise des fonctions de remplacement automatique. Dans le présent système, un filtre semblable a été installé, qui permet de nombreuses transformations. Il doit encore voir de nombreuses améliorations mais la fonction a toutefois déjà reçu le nom pompeux de «`lavoisier`».

`CuSO4` est ainsi transformé en `\q{CuSO_4}`, qui donne $CuSO_4$. Le code `\q` est un synonyme du code `\chemical` du paquet ConTEXt.

Un grand nombre de composés de chimie sont parfaitement descriptibles en format de texte ASCII car pour la plupart il existe une relation bijective entre le format de texte brut et le format composé selon les règles typographiques des chimistes.

En conséquence l'emploi d'un logiciel compliqué de mise en page est ainsi même remis en question puisqu'un tel filtre permet de s'affranchir des détails de typographie. En tout état de cause, il est toujours possible de bloquer le travail du filtre, par exemple en ajoutant des accolades que TEX n'imprimera pas.

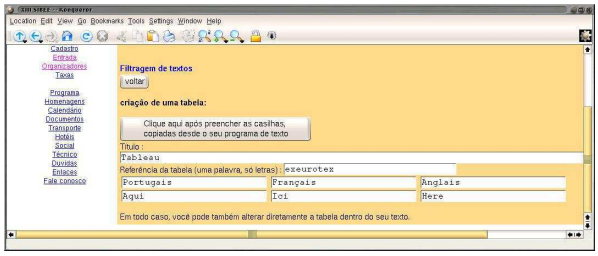
Le fichier de filtrage contient quelques centaines de lignes, et son exécution par le script `php` dure quelques dixièmes de seconde sur un texte de deux ou trois pages. Une seule passe permet de traiter tous les composés du

type $C_xH_yO_z$, par exemple, et quelques autres passes permettent aussi de traiter raisonnablement un grand nombre d'unités : `mV.s-1` ou bien `mV s-1` sont alors transformés de suite en `mV$\cdot$s$^{-1}$` qui donne donc $mV{\cdot}s^{-1}$ sur le papier. On notera au passage que ce dispositif permet d'assurer une bonne homogénéité des conventions typographiques, et qu'il peut être adapté, par exemple en forçant l'emploi du L majuscule pour le symbole du «litre» ou au contraire pour lui substituer le «dm^3».

Le même travail est mené avec toutes les unités nécessaires, et peut être étendu à certains des codes typiques de TEX que de nombreux auteurs oublient ou ne connaissent pas. Par exemple, le filtre ajoute automatiquement le caractère d'échappement avant `%`, `$`, `#` ou `&`, mais sans en ajouter un second s'il y en a déjà un, et peut aussi éliminer des espaces parasites avant les virgules. Le traitement des espaces fines avant les symboles hauts de ponctuation sont en revanche traités directement par TEX, car dans certains cas on veut pouvoir modifier le comportement du programme.

Programmé en script `php`, le filtre est modifiable à tout moment et peut aussi être mis à profit pour engendrer les codes `html` correspondants. Cela signifie que à partir d'un même document lisible par un être humain, nous pouvons créer un document TEX ou un document destiné à être publié sur l'Internet.

Tableaux Une page spéciale de filtre a été préparée pour que chacun puisse rédiger sans trop de peine les codes permettant de créer un tableau. En premier lieu on indique le nombre de lignes et de colonnes. Alors le système présente un ensemble de cases qu'il suffit de remplir pour afficher ensuite le code complet correspondant au tableau. Ainsi la page suivante :



permet de préparer sur le champ les codes suivants, qui permettent de rendre le tableau :

```
\TV 3 colunas Tableau [exeurotex]
\PT | Portugais | Français | Anglais |
\NL | Portugais | Français | Anglais |
\NL | Aqui | Ici | Here | \FIMTAB
```

Mots-clés

Chaque article est susceptible de recevoir trois mots-clés, et dans l'index apparaîtront donc trois entrées par article, dans lesquelles les deux autres mots-clés seront indiqués entre parenthèse. On pourrait ainsi trouver rapidement dans le livre, parmi quinze articles traitant du carbone, ceux qui auraient rapport à sa protection contre l'oxydation. Cette façon de créer l'index des mots-clés peut paraître traditionnelle mais son simplicité apparente n'empêche qu'elle soit efficace et donne un document très compact.

Création du livre

À tout moment chaque participant au congrès peut afficher, en temps réel, les titres des articles en préparation, classés par chapitre. Les données sont stockées sur une base consultée à chaque demande du serveur, et cela ne surcharge pas excessivement la machine car les clients potentiels sont peu nombreux.

À tout moment également le responsable du site peu préparer un fichier du livre entier ou bien d'un ensemble d'articles, par exemple pour l'envoyer à l'un des membres du comité scientifique. Une fois de plus, c'est le script `php` qui prépare le fichier à faire compiler par TEX. Les quelque 700 pages du livre des actes de ce congrès, contenant 265 travaux, sont compilées en moins d'une minute.

Améliorations nécessaires

Ce système a subit de nombreuses modifications au cours de son rodage, dont la plupart n'étaient pas imaginées au début de sa construction, car la différence énorme de vision entre un usager de programme de PAO grand public et celle d'un usager de TEX est telle que la communication avec les participants au congrès fut parfois difficile.

Toutefois, chaque participant au congrès s'est accordé à reconnaître que le résultat obtenu est d'excellente qualité, et le livre des actes, finalement imprimé dans les 5 jours qui ont précéde le congrès, a fait l'unanimité des participants, et un système semblable sera probablement employé lors de la prochaine édition du SIBEE, en août 2004. Il reste néanmoins de nombreux détails à améliorer, tant du côté du serveur que du côté des programmes de rédaction.

En amont : rédaction Ma méconnaissance de certains outils du logiciel Word ou OpenOffice fait que je n'ai pas imaginé que l'on puisse créer certaines macros qui permettraient de sauvegarder le texte en un format qui soit acceptable par le système. Il existe, notamment disponible avec le logiciel Abiword, la possibilité de sauvegarder en format LATEX, mais ce format ne fait de reprendre trait pour trait la disposition du texte original tel qu'il apparaît lorsqu'il est mis en page par Abiword. Il faut au contraire extraire les données pertinentes, sans autre détail inutile.

En aval En aval, le serveur doit offrir des outils plus performants permettant de filtrer les textes de façon entièrement transparente ou de créer des tableaux complexes plus aisément. En particulier, une option mérite d'être sérieusement reconsidérée, qui consiste à employer LaTeX au lieu de TeX comme outil de mise en page, bien que la simplicité des commandes actuelles a rendu possible la création d'un ouvrage aussi volumineux, rédigé par une majorité de non spécialistes de TeX. Le site du congrès est accessible à l'adresse `http://sibee.iq.unesp.br` où nous placerons une zone d'accès libre, et une fois qu'une version fonctionnant bien sera en service, elle sera mise à la disposition du public.

Remerciements

Erwin Bergamo a passé de nombreuses heures à corriger de nombreux textes ayant des problèmes de compilation. Durant trois mois il a fourni une aide précieuse, et pourtant il lui fallait simultanément terminer la rédaction de son mémoire de doctorat ; et son mémoire est un des premiers de notre institut à être mis en page avec LaTeX.

Éthique et édition scientifique d'ouvrages anciens sur support électronique : choix de traitements numériques des ornements et marques typographiques dans le programme de numérisation des collections d'ouvrages anciens du laboratoire ATILF

Isabelle Turcan
Université Jean Moulin, Lyon III ; Laboratoire ATILF, UMR 7118, Nancy II, France
Isabelle.Turcan@sunlyon3.univ-lyon3.fr

Viviane Berthelier
Centre de documentation, Laboratoire ATILF, UMR 7118, Nancy II, France
viviane.berthelier@inalf.fr

Abstract

This paper presents a small part of the work being done at our university to coordinate and digitize the collections of rare, old and fragile books currently in the library of the former *Institut National de la Langue française* (INALF).

Our paper focuses on the library's very old dictionaries, documents often treated with indifference, during the first stages of the digitization project, as being of only passing interest. Old printing specificities are quite often the vehicle for very interesting details regarding the history of such documents; electronic data must be able to transmit those details to their modern users. Our work lies in better defining and integrating these specificities into our modern electronic procedures. With the help of some original transparents, we will show how difficult — but how interesting and useful as well — the task of bringing such typographic ornaments to a digitized display can be.

Résumé

Notre propos s'inscrit dans le cadre du programme de numérisation des collections d'ouvrages anciens du laboratoire ATILF, «Analyse et Traitement Informatique de la Langue Française» visant à préserver les volumes plus fragiles ou les plus précieux pour en valoriser le contenu et en faciliter le partage avec la communauté scientifique des chercheurs, avec l'ensemble des fonds anciens de bibliothèques, et plus largement, avec le grand public cultivé.

Notre réflexion porte sur les dictionnaires anciens, documents souvent «maltraités», ou médiocrement, qu'il s'agisse des premières étapes de la genèse de création des documents numériques, des modalités de conception des corpus d'images numériques respectant les caractéristiques typographiques de l'imprimerie ancienne pour être en mesure d'en transmettre les enseignements afférents, ou de données textuelles électroniques : à partir d'un échantillon représentatif des collections du Laboratoire, nous présenterons sous forme de transparents les documents relatifs aux différentes phases de notre travail, en particulier pour le traitement des ornements typographiques.

Préambules

Numérisation des collections d'ouvrages anciens du laboratoire ATILF Le Laboratoire ATILF, Analyse et Traitement Informatique de la Langue Française (UMR 7118, Nancy II) a mis en chantier depuis l'an dernier, en septembre 2002, un programme de numérisation de ses collections d'ouvrages anciens, dans trois buts :

- préserver les exemplaires les plus fragiles ou les plus précieux ;
- transmettre l'ensemble du catalogue des collections et les contenus des ouvrages à la communauté scientifique des chercheurs français et étrangers ;
- valoriser le contenu des collections en en facilitant le partage, non seulement au-delà de la communauté scientifique, avec la communauté des bibliophiles consultant les fonds anciens de bibliothèques, et plus largement encore, avec le public des enseignants et des lycéens, et le grand public cultivé, de Lorraine ou d'ailleurs.

Dans le cadre de ma première année de délégation auprès du laboratoire dont l'objectif défini fut celui de

travailler à la valorisation du fonds d'ouvrages anciens, je me trouve avoir l'honneur de piloter ce vaste programme de numérisation et j'ai le bonheur de travailler avec Viviane Berthelier, Responsable du Centre de Documentation, et ses collègues, dont je salue ici la qualité du travail réalisé en équipe. Je m'exprimerai bien entendu à la fois en mon nom propre et au nom du Centre de Documentation du laboratoire.

Un projet inscrit dans une démarche scientifique Mon propos s'inscrit dans le cadre d'une réflexion que j'ai menée depuis déjà plusieurs années (cf. *infra*, la bibliographie indicative) sur une catégorie de documents numériques souvent, hélas, «maltraitée», à tout le moins trop médiocrement traitée, celle des ouvrages anciens, et plus particulièrement des dictionnaires imprimés depuis le XVI^e^ siècle, qu'il s'agisse :

- des premières étapes de la genèse de création des documents qui impliquent une identification des exemplaires et une référenciation rigoureuses ;
- des modalités de conception des corpus d'images numériques fiables et de qualité ou de données textuelles électroniques répondant aux exigences philologiques de la recherche ;
- des choix d'exploration par un public diversifié et d'exploitation par des chercheurs ;
- des choix de transmission et de diffusion des corpus numériques constitués[1].

Les trois grandes orientations de notre démarche : rappel

Genèse de la création des documents numériques : impératif de référenciation Pour les premières étapes de la genèse de création des documents numériques, il faut signaler en priorité la délicate question d'identification la plus rigoureuse possible des exemplaires à numériser : édition originale ? Retirage ou contrefaçon ?

Principes minimaux d'identification bibliographique L'identification qui doit être fondée sur des critères scientifiques, effectuée de façon minutieuse quand il s'agit d'ouvrages dont on sait qu'ils ont fait l'objet de contrefaçons, entre autres en raison de leur succès lors de leur première parution[2]. On ne saurait en conséquence cautionner toute édition électronique de dictionnaires anciens sur l'aléatoire de considérations matérielles, auxquelles s'ajoutent dans certains cas d'affiliation, des considérations commerciales, réalisée sur des exemplaires non identifiés, non localisés, non référencés, comme c'est le cas du site *Gallica*, pourtant institutionnel (BNF), sur lequel on peut encore trouver, hélas trop souvent et malgré les nombreuses critiques émises par des spécialistes de l'histoire du livre, des images d'ouvrages anciens dont les pages sont coupées et ne laissent apparaître ni l'adresse de l'imprimeur, ni la date d'impression, ni la mention de privilège ou d'approbation de l'autorité royale à laquelle était soumise toute publication sous l'Ancien Régime. Le décalage référentiel est alors plus que gênant.

Un exemple tristement révélateur des impasses d'une mauvaise numérisation On retiendra l'exemple caractéristique du *Dictionnaire Universel de Furetière*, imprimé pour la première fois en Hollande en 1690, et dont le succès a suscité ensuite de nombreuses impressions, notamment en 1691, en 1694, puis en 1701 et en 1702 : simples retirages ? Nous avons réalisé des transparents avec des images diffusées sur *Gallica* (document A) pour en proposer une comparaison avec des images prises sur les exemplaires du fonds ATILF (documents B, b1, b2, b 3) qui sont en eux-mêmes éloquents : quelle est la valeur référentielle d'une page tronquée ne comportant ni l'adresse du libraire imprimeur, ni la date d'édition ? quelle est la valeur pour le bibliophile ou le chercheur soucieux de mener une étude de bibliographie matérielle d'une image de marque typographique de libraire floue, voire sale, sans qu'il soit possible de l'extraire du fichier pour réaliser un gros plan permettant de l'apprécier à sa juste valeur ?

Les corpus d'images numériques et de données textuelles électroniques

Conception des corpus numériques Pour les modalités de conception des corpus d'images numériques ou de données textuelles électroniques, de leur gestion en vue d'une bonne transmission puis d'une diffusion en réseau, on distinguera plusieurs catégories de choix réparties en deux grands ensembles, avec d'abord :

- pour les techniques de numérisation :
  - en mode image simple — en choisissant la meilleure qualité possible de définition des images ;
  - en mode image plus ou moins amélioré par une indexation minimale automatique susceptible d'être enrichie progressivement par différentes séries d'indexation de plus en plus fines et rigoureusement mises en œuvre ;
  - en mode image équivalent à un mode texte fondé sur une connaissance fine des textes reproduits et sur des critères rigoureux de balisage[3] ;
- pour le choix de la méthodologie mise en œuvre en vue d'une première exploration des bases de

1. Je reviendrai ultérieurement sur quelques-uns des critères les plus importants à mes yeux.

2. Cf. notre intervention au colloque consacré à l'«Éthique numérique» [15]

3. Cf. pour mémoire nos différents travaux sur le balisage, en particulier [7].

données ainsi produites en tenant compte de l'opposition entre la conception de simples bases d'images et celle de bases de données textuelles fondées sur des images converties selon différentes techniques, non pas limitées à de la simple océrisation[4].

La diversité des options On doit réfléchir à la diversité des options d'exploration, d'exploitation et de diffusion des corpus numériques ainsi constitués en fonction de la variété des publics auprès desquels les versions numérisées des textes sont susceptibles d'être diffusées[5], sans se leurrer sur la facilité d'une vulgarisation de bas niveau comme cela a malheureusement été parfois fait dans des perspectives purement commerciales, et sans respect ni des ouvrages susceptibles d'être consultés, ni a fortiori des consultants[6] : la conception de documents numériques implique à la fois une réflexion préalable sur le principe des utilisations potentielles des produits ainsi réalisés, à la fois la prise en compte de l'attente de publics diversifiés.

Commentaire des documents C :

- en C1, la reproduction de la page de titre de la *Bibliothèque Orientale* de d'Herbelot (1697), de l'exemplaire conservé à l'ATILF, qui seule peut légitimer par sa présence les clichés de l'ensemble de l'exemplaire référencé ;
- en C2, la reproduction d'une page extraite de la *Bibliothèque Orientale* avec des exemples conjoints de typographie ancienne et de notes manuscrites qu'il faut transmettre de façon correcte aux chercheurs intéressés, par l'association de procédures de traitement des images des différentes catégories de texte et des contenus textuels au sein de l'image globale.

Objectif du programme de numérisation de l'ATILF

La constitution de corpus associés De fait, l'objectif du programme de numérisation du laboratoire ATILF est de constituer plusieurs corpus de bases de données complémentaires et hiérarchisées, bases que nous enrichirons au gré des possibilités matérielles dont nous disposerons, sachant que notre corpus électronique constituera une base de travail pour trois grands axes de recherche, en linguistique (histoire de la langue, histoire du lexique, histoire des textes théoriques et didactiques), en sciences humaines et sociales et en traitement automatique de la langue.

Nous distinguons pour le moment, outre l'accès sur le web au catalogage fin du fonds comprenant des indications relatives aux ouvrages présentant des particularités telles que des *ex libris* ou des annotations manuscrites, deux ensembles de corpus constitués en bases de données, un corpus d'images et un corpus de textes, avec :

- pour le premier, un corpus réunissant l'ensemble des images de pages de titres et des paratextes (préfaces, épîtres dédicatoires, avertissement), — sachant qu'est défini à l'intérieur de ce premier ensemble un sous-corpus consacré aux ornements typographiques, lui-même organisé selon les différentes catégories d'ornements (lettrines, bandeaux, culs de lampe) —, et un corpus comprenant l'ensemble des images des pages ou colonnes des ouvrages du fonds (la pagination de certains volumes en format *in folio* correspond aux numérotations des colonnes : cf. le *Dictionnaire Universel François et Latin, de Trévoux* (1704–1771)), avec indexation minimale automatique ;
- pour le second, un corpus textuel minimal correspondant à l'ensemble des mots-vedettes traités dans les différents dictionnaires, ce corpus donnant accès, grâce à des liens hypertextuels aux images correspondantes des pages concernées, sachant que sera élaboré un second corpus élargi progressivement aux textes des articles des dictionnaires, avec d'abord ceux qui auront pu produire des résultats satisfaisants à l'océrisation, puis ceux qui auront pu bénéficier des progrès techniques de traitement de ce que j'ai choisi de désigner par le syntagme d' «image textuelle»[7] ou qui auront fait l'objet d'une saisie manuelle traditionnelle.

NOTA BENE En aucun cas, nous n'admettons la notion de corpus textuel à propos d'images numériques de textes de dictionnaires dès lors que ces dernières n'auront pas été traitées de façon à en permettre une exploitation automatisée en plein texte.

À la faveur des premières expérimentations de réalisation de documents numériques à partir d'un échantillon d'ouvrages lexicographiques anciens du fonds, échantillon représentatif de la collection conservée au Laboratoire ATILF, il est désormais temps de présenter, sous forme de transparents (reproduits en annexe) les documents relatifs à une des phases essentielles de notre travail, tel que nous l'envisageons pour les prochains mois, en particulier pour ce qui concerne le traitement informatique de l'ensemble des composantes de la typographie ancienne qu'il s'agisse de la diversité des ornements typographiques et des différentes sortes de caractères typographiques.

4. Cf. sur ce sujet, nos précisions dans l'article à paraître dans le prochain numéro de la revue *Document numérique* (septembre 2003).

5. Cf. nos deux textes d'introduction aux cédéroms réalisés aux éditions Redon [19, 20].

6. Cf. [17].

7. Cf. ma proposition de définition dans l'article à paraître dans le prochain numéro de la revue *Document numérique* (sept. 2003).

Les ornements typographiques dans les dictionnaires anciens

Rappel concernant l'importance des ornements typographiques

Les acquis des études de bibliographie matérielle

Toutes les études de bibliographie matérielle ont largement montré l'importance que revêtait une étude rigoureuse des ornements typographiques — gravures, bandeaux, marques de libraires, lettrines, culs de lampe — présents dans les ouvrages imprimés sous l'Ancien Régime : ils permettent d'identifier, de la façon la plus précise possible, les éditions originales légitimées par l'autorité royale qui a délivré, à une date précise, un privilège et une approbation ; outre les détails textuels de ces documents officiels reproduits sur tout exemplaire imprimé sur une presse autorisée, l'analyse fine des ornements typographiques offre un critère complémentaire important de différenciation des retirages non autorisés et contrefaçons, qui, parfois, sous couvert d'une utilisation abusive sur les pages de titre de la mention des privilèges et approbations, souvent d'ailleurs pas même reproduits, ont bénéficié de la récupération des matrices, bois et cuivres, de marques typographiques plus ou moins similaires à celles des éditions de référence[8].

Le principe d'identification de la nature exacte des exemplaires reconnus comme appartenant à une série de tirages originaux ou au contraire à un ensemble d'impressions suspectes s'impose non seulement dans la perspective d'une contribution à l'histoire du livre ancien, mais aussi par rapport aux conditions de transmission des contenus qui peuvent varier selon l'identité des exemplaires identifiés. Or, le cloisonnement des disciplines que l'on connaît malheureusement en France, n'a pas toujours jusqu'à présent permis aux chercheurs en sciences humaines, en particulier aux linguistes, philologues et historiens de la langue, comme aux littéraires, pourtant attentifs aux moindres détails formels, telles les graphies, l'accentuation, orthographe ou la ponctuation, ou textuels, comme les tournures de phrases, les emplois lexicaux, la mise en paragraphes ou encore la mise en page, d'être au moins sensibilisés aux variantes observables d'un exemplaire à l'autre d'une même édition ni de se former à une connaissance minimale du livre ancien pour être à même d'exercer leur esprit critique dans bon nombre de leurs références bibliographiques.

De la matérialité, l'objet-livre, au virtuel : la fiabilité numérique À l'ère du presque tout numérique et de la transmission documentaire massive à l'échelle mondiale, il devient capital de prêter une attention particulière aux techniques susceptibles d'être mises en œuvre pour diffuser, au public des chercheurs en sciences humaines et sociales, aux spécialistes du livre ancien, tout autant qu'à un public plus diversifié d'amateurs ou de curieux, des images d'exemplaires d'ouvrages anciens qui soient fiables, au moins à trois égards :

- par une référenciation détaillée qui permette à tout chercheur de se reporter sans difficulté à l'exemplaire d'origine (indication rigoureuse de l'ensemble des détails figurant sur la page de titre de l'exemplaire, localisation dans un fonds particulier, cote de l'exemplaire dans ce fonds, format) ;
- par une qualité de définition des images qui soit la plus à même d'analyser les moindres signes distinctifs de récupération des ornements et des casses de caractères ;
- dans le cas de corpus importants, par une indexation des fichiers images qui tienne compte autant de la matérialité des mises en page que des contenus combinatoires avec les composantes textuelles.

Comme nous avons déjà traité le premier point *supra* en partant d'exemples négatifs (2. 1.) qu'il ne faut plus prendre le risque de reproduire, dès lors que l'on est conscient de l'indispensable éthique numérique (cf. *supra*, n° 3) à laquelle devrait adhérer tout chercheur sérieux, me reste à commenter désormais les exempliers consacrés au traitement de l'ensemble des informations présentes sur une page de dictionnaire sous forme numérisée en un certain mode image …

D'une image à l'autre : images et textes Une page de dictionnaire sous forme numérisée n'offre qu'une image globale dont il convient de définir les différents champs informationnels de façon à les indexer en vue d'une transmission fiable et efficace. Ainsi une page de titre, dont on connaît aussi la fonction publicitaire, comprendra plusieurs champs susceptibles de faire l'objet d'indexations différenciées, qui elles-mêmes peuvent nourrir des bases de données spécifiques ; de même pour une page comprenant à la fois un bandeau et une lettrine, une gravure et

8. Pour la langue française, deux de mes travaux sur les dictionnaires anciens : «Présentation de quelques exemplaires des ouvrages de Gilles Ménage au fonds ancien de la Bibliothèque Municipale de Lyon : originaux et contrefaçons», in *Gilles Ménage (1613–1692), Grammairien et lexicographe : le rayonnement de son œuvre linguistique*. Actes du Colloque International tenu à l'occasion du Tricentenaire du Dictionnaire Etymologique ou Origines de la Langue Françoise (1694), édition proposée par I. Leroy-Turcan et T. R. Wooldridge, Siehlda, Université Jean Moulin, 1995, p. 363–389 + p. 411–413 ; «Présentation des différentes éditions du *Dictionnaire de Trévoux* recensées à Nancy (INALF), Lyon (BM), Paris (BNF), Bourg-en-Bresse (BM) et Trévoux : comparaison des éditions et principes méthodologiques pour une nouvelle approche de l'ordre canonique des éditions trévoltiennes, parisiennes et nancéiennes», Colloque international sur «Connaissance et rayonnement du *Dictionnaire Universel … de Trévoux* (1704–1771)», publication des Actes, sur le site universitaire Trévoux : `http://www.univ-lyon3.fr/siehldaweb/Trévoux-accueil.htm`.

un lettrine, un cul de lampe puis une lettrine, etc.

L'image d'une image avec texte : le repérage des champs Nous avons choisi deux exemples différents d'images extraites de dictionnaires anciens[9] spécialisés dans le domaine de la marine, celui d'une page de titre où la partie textuelle domine, la page de titre du *Dictionnaire des termes propres de marine*, de Desroches, Paris, 1687[10] : exemplier 1a, p. 365), celui d'une gravure où l'on ne saurait négliger la composante textuelle, gravure présente au début du tome III de l'ouvrage intitulé *Arts de l'Homme d'épée ou Dictionnaire du gentilhomme*, le tome III, l'*Art de la navigation*, étant un dictionnaire technique de termes de marine, ouvrage signé de G. Guillet, édité en 1678[11] (exemplier 1b, p. 366).

Sur l'image de la page de titre reproduite sur l'exemplier 1a, la composante textuelle domine et nous déterminons, indépendamment de la découverte traditionnelle de toute page de titre selon la progression d'une lecture horizontale (intitulé, auteur, pièces complémentaires, marque du libraire imprimeur, adresse d'édition, date et indication des privilège et approbation), des ensembles verticaux répartis dans la page qui correspondent aux différentes strates d'informations liées, en situation centrale à la confection du livre, et en ajouts latéraux à sa conservation : ainsi pour la cote de la BML, pour l'*ex libris* manuscrit réparti de part et d'autre du titre, l'*ex libris* ancien tamponné dans la marge de droite, le tampon plus récent de la bibliothèque municipale dans la marge de gauche. Sur la gravure de navire figurant sur l'exemplier 1b, nous distinguerons, indépendamment de l'opposition manifeste entre la partie image pure, la plus importante, elle-même analysable en plusieurs champs (la référence intratextuelle, la rose des vents, la banderole servant de légende à la gravure), la liste de termes correspondant aux numéros présents sur la gravure, liste étendue en double colonne que nous devrons traiter à la fois comme composante de l'image et comme ressource textuelle devant faire l'objet de liens hypertextuels avec les mots traités dans le corps du dictionnaire, c'est-à-dire avec les vedettes et le texte des articles leur correspondant. Ce premier travail de repérage des champs analysables dans une image soumise à numérisation est forcément indissociable des choix afférents de traitement en matière d'indexation minimale en vue de la constitution de différentes catégories de bases de données complémentaires.

Traitement des champs dans une image : les différentes catégories d'indexation

Les champs informationnels : images et textes dans l'image J'ai choisi dans un premier temps de matérialiser par des couleurs, sans autre portée sémiotique que la fonction que je leur ai attribuée, les principales modalités de traitement des différents champs, ce qu'il est possible de vérifier déjà sur l'exemplier 1b, p. 366, avant de revenir sur l'exemplier 2, p. 367 consacré aux modalités de traitement des champs dans l'image présentée en 1a.

Le violet correspond aux données textuelles qui doivent être passée à l'OCR ou ressaisies quand l'OCR risque de ne pas offrir de résultats satisfaisants (cf. la liste en deux colonnes de l'exemplier 1b, ainsi que les deux autres composantes textuelles en légende, du fait conjoint de la police en italique et de la petitesse des caractères). Toutes les données textuelles doivent faire l'objet d'une indexation absolue pour qu'en soit envisageable une exploration automatisée en plein texte. Le corps de l'image représentant le trois mâts fait l'objet à la fois d'une indexation banale de gravure liminaire associée au sein d'une base restreinte aux ornements typographiques, une base «OrTy» matérialisée ici par le bleu, et d'une indexation exemplificatrice associée aux termes techniques définis dans le dictionnaire par des liens hypertextuels. Le vert correspond aux composantes textuelles qu'il est difficile de passer à l'OCR avec obtention de résultats satisfaisants et qu'il faut donc envisager de saisir manuellement, qu'il s'agisse, comme on le vérifie sur l'exemplier 2 des *ex libris* manuscrits ou des tampons anciens dont certaines abréviations nécessitent une explicitation, qu'il s'agisse des passages en police italique de petite taille. Il nous paraît important en effet, au sein de notre programme de numérisation du fonds ATILF de pouvoir constituer des bases associées consacrées aux principales caractéristiques des collections du fonds, en particulier de tous les détails relevant de la traçabilité des itinéraires de ouvrages d'un fonds à l'autre. De même est-il essentiel de pouvoir organiser la base consacrée aux ornements typographiques, en plusieurs composantes correspondant chacune aux principales catégories de marques : ainsi, comme on le voit dans l'exemplier 2, la marque du libraire imprimeur, en bleu. Les deux exempliers suivants me conduit à préciser l'intérêt d'une extraction de motifs typographiques déterminés pour documenter les chercheurs soucieux de mener des analyses de bibliographie matérielles complexes.

L'ornement typographique : un champ informationnel particulier J'ai associé sur les transparents 3a et 3b, les reproductions de pages liminaires d'exemplaires conservés à l'ATILF, apparemment sans rapport bibliographique, la page de titre du tome 1 du *Nouveau dictionnaire universel des arts et des sciences, françois, latin et anglois, traduit de l'anglois de Th. Dyche*, Avignon, 1753 face à celle d'une contrefaçon du *Dictionnaire de l'Académie françoise*, nouvelle édition, Nismes, 1778 : mais si l'on s'intéresse aux adresses des libraires imprimeurs, Avignon et

9. Conservés au fonds ancien de la bibliothèque municipale de Lyon qui a eu l'amabilité de nous en préparer d'excellents clichés.

10. Fonds ancien de la bibliothèque municipale de Lyon, cote 306 226.

11. Fonds ancien de la bibliothèque municipale de Lyon, cote 343 197.

Nîmes, villes connues dans l'histoire de l'imprimerie pour leur rivalité, et aux ornements typographiques présents sur chaque exemplaire, avec en premier lieu la marque du libraire, un motif de panier de fleurs ornementé de feuillages, similaires à l'exception d'une bouclette (exemplier 3a, p. ??), en second lieu sur la première des épîtres dédicatoires (exemplier 3b, p. ??), les deux bandeaux et les deux lettrines à motif paysager qui signent en quelque sorte chacun des ouvrages si l'on pense au rôle des ornements typographiques et aux enjeux afférents pour une meilleure maîtrise de l'histoire de l'impression et de la diffusion du livre ancien.

La constitution de corpus spécifiques de ces ornements typographiques associés aux textes numérisés dans leur intégralité offre un outil de travail tout à fait complémentaire de bases déjà consacrées aux ornements typographiques, tel l'exemple réalisé pour les Pays-Bas, pour identifier des exemplaires dont les contenus textuels peuvent varier, comme c'est en particulier le cas pour contrefaçons du *Dictionnaire de l'Académie française*, d'où notre souci de ne pas dissocier les ornements typographiques des textes sur lesquels ils figurent, même en isolant des sous-corpus.

Textes en images et images de textes : de l' «image textuelle»

Ambiguïtés sémantiques : des données textuelles en images ne restent que des images ... On prendra garde aux ambiguïtés terminologiques concernant la numérisation de documents textuels en mode image, abusivement considérés comme des *données textuelles* et désignés sous la mention «Type de ressource électronique», comme on le constate avec l'exemple flagrant de la notice du *Dictionnaire universel* de Furetière sur *Gallica*[12], quand seul l'accès aux images des textes est fourni, sans que les composants textuels soit explorables en plein texte, qu'offre une saisie absolue des textes, manuellement[13] ou via des OCR suivis de corrections manuelles[14].

En revanche, quand un fichier image jouit d'un traitement en fichier texte, minimal (traitement des vedettes : cf. l'exemplier 4, p. 368) ou étendu (traitement en plein texte), on parlera légitimement de «données textuelles». L'exemplier 4 montre que les médiocres résultats de reconnaissance des caractères d'une typographie ancienne, elle-même de faible qualité, conduisent à limiter, dans un premier temps, l'usage des OCR pour les vedettes et sous-vedettes, balisées sur les fichiers images, listées et enregistrées dans une base textuelle associée aux fichiers images.

Un mode image amélioré par une mauvaise indexation ? Des bases de données textuelles fiables ? Le flou choisi de certaines désignations Quand d'aucuns consultants ont pu être attirés par une forme de publicité, hélas, trompeuse, par le syntagme de «mode image amélioré», laissant entendre / imaginer que le simple mode image d'un texte de dictionnaire pourrait bénéficier d'une amélioration de qualité le rapprochant de façon avantageuse d'une numérisation fondée sur une saisie du texte offrant alors un accès aux documents en plein texte ... mais, par la seule indexation des vedettes. Encore faudrait-il être assuré que toutes les vedettes et sous-vedettes aient été alors recensées correctement, encore faudrait-il d'ailleurs que les images traitées correspondent à un ensemble d'exemplaires homogènes quand il s'agit d'ouvrages imposants en plusieurs volumes comme ce fut le cas en 1999, sur le site d'un grand organisme public de conservation du patrimoine, pour une première expérience insatisfaisante de numérisation en mode image prétendument amélioré du *Grand Dictionnaire Universel du XIX^e siècle* de P. Larousse à partir des reproductions photographiques d'exemplaires dépareillés. On s'amusera d'ailleurs à noter que l'objet a été enlevé de la vitrine des dictionnaires de Gallica, mais que l'on peut néanmoins retourner à ces images en cherchant dans la boite de requête le nom de Larousse. Cette décourageante expérience reste ainsi accessible en sous-couche de l'entreprise de publication électronique, mais avec tous les défauts qui avaient pu lui être adressés dès l'origine par des experts, dont le moindre n'est pas que — faute d'indexation minimale des vedettes — la consultation électronique de ce dictionnaire monumental ne puisse pas être effectuée à partir de la seule indication des pages de chaque tome. Il est effectivement ridicule que les nouvelles technologies d'information — mal maîtrisées — rendent obligatoire d'avoir sous la main les 17 tomes de l'édition papier pour repérer l'entrée cherchée dans la suite des pages de chacun d'entre eux ! Depuis, la définition de l' «image textuelle» telle qu'elle a été conçue et mise en œuvre en 2002 par les éditions Redon pour proposer un traitement textuel des images numérisées du *Grand Dictionnaire Universel du XIX^e siècle* de Pierre Larousse, est en mesure de proposer un accès au plein texte[15].

Conclusions

Du texte en image à l'image textuelle, les principes de traitements novateurs d'une image de texte prennent en compte l'ensemble des composants de la page image et ne

12. Consultation effectuée le 6/5/03, sur `http://gallica.bnf.fr/Catalogue/Notices/IMP/n050614.htm`

13. Cf. l'exemple des textes intégraux des dictionnaires saisis manuellement pour la réalisation de cédéroms, *infra*, bibliographie.

14. Cf. l'expérimentation réalisée au sein du laboratoire pour la huitième édition du *Dictionnaire de l'Académie française*, 1932–1935.

15. Une première édition proposée par les éditions Champion en 2000 a été depuis compensée par la belle réalisation des éditions Redon, sous la responsabilité scientifique de J.-Ph. Saint-Gérand, diffusion [21].

peuvent alors que respecter le texte, à la fois du point de vue typographique, du point de vue philologique et donc du point de vue des contenus, ce qui permet d'aboutir à des résultats très satisfaisants dans des délais plus avantageux qu'avec de multiples passages d'OCR comparés ou combinés, tout en autorisant l'extraction des ornements typographiques comme un composant de l'image isolable comme tel.

Telle est donc notre choix de combinaison de bases de données d'images d'images, d'images associées à des textes et d'images de textes ouvertes sur des bases textuelles.

Certes, de l'image au texte, reste la question parfois délicate du traitement des signes propres à la typographie ancienne, tels le f long, les graphies i/j ou u/v, les perluettes et les ligatures. Notre principe consiste à choisir le maintien des signes quand ils ont une fonction linguistique distinctive, sinon, un traitement relevant de la translittération.

Notre programme de numérisation des collections patrimoniales du fonds d'ouvrages anciens de notre laboratoire en étant encore à la première phase expérimentale, il nous est à ce jour difficile de détailler les résultats auxquels nous sommes déjà parvenus sur échantillons modestes dans le cadre de ce vaste chantier, mais la numérisation d'un premier corpus de 10 000 images d'un grand dictionnaire du XVIII^e siècle est en cours et nous en envisageons pour l'été avenir les premiers traitements selon les principes que nous avons énoncés, ce qui devrait nous autoriser à présenter un ensemble de réalisations concrètes lors d'un prochain colloque.

Références

[1] I. Leroy-Turcan et T. R. Wooldridge, «Une autre identité de la première édition du *Dictionnaire de l'Académie Française* : quelques exemples des acquis de la base informatisée de la première édition du *Dictionnaire de l'académie française* (1694)», conférence donnée à Québec et Montréal (édition électronique sur le site Académie à Toronto).

[2] —— «L'informatisation des premiers dictionnaires de langue française : les difficultés propres à la première édition du *Dictionnaire de l'Académie française*», *Les Dictionnaires de Langue française et l'informatique*, Actes du Colloque international sur les «Dictionnaires de langue française et informatique », *La Journée des Dictionnaires* 1995, revue *Centre de recherche texte/Histoire*, Actes publiés par J. Pruvost, Université de Cergy-Pontoise, Juin 1997, p. 69–86 (Rédaction I. Leroy-Turcan à partir de données informatisées fournies par T. R. Wooldridge ; éd. parallèle sur le site Académie à Toronto).

[3] I. Turcan, «Modalités de mise en œuvre de l'informatisation de la première édition du *Dictionnaire de l'Académie françoise* (1694)», in Actes du colloque international de Clermont-Ferrand sur *Les dictionnaires électroniques du français des XVI^e et XVII^e siècles*, 14–15 juin 1996 : Actes édités par T. R. Wooldridge (SIEHLDA) sur Internet (sites de Toronto, Lyon et Clermont-Ferrand).

[4] —— «L'informatisation de la pré-édition du *Dictionnaire de l'Académie françoise* (1687) est-elle à l'ordre du jour ?», in Actes du colloque international de Clermont-Ferrand sur *Les dictionnaires électroniques du français des XVI^e et XVII^e siècles*, 14–15 juin 1996 (texte publié sur Internet).

[5] —— «Intérêt d'une base informatisée pour le *DEOLF* de Gilles Ménage : les modalités de mise en œuvre.», in Actes du colloque international de Clermont-Ferrand sur *Les dictionnaires électroniques du français des XVI^e et XVII^e siècles*, 14–15 juin 1996 (texte publié sur Internet).

[6] —— «Modalités de création d'une base informatisée "vocabulaire de la marine au XVII^e siècle" : problèmes relatifs aux corpus de référence et aux documents permettant de vérifier la vitalité des termes enregistrés par les dictionnaires», in Actes du colloque international de Terminologie Maritime : *traduire et communiquer*, organisé par l'Institut supérieur de traducteurs et interprètes de Bruxelles, Editions du Hasard, Bruxelles, décembre 1999, p. 70–92.

[7] —— «Balisage formel ou balisage fin pour les dictionnaires anciens informatisés : objectifs et implications méthodologiques. L'exemple du *Dictionnaire de l'Académie Française* (1694) et des bases échantillons des dictionnaires de Gilles Ménage (1694) et de Thomas Corneille (1694)», in Actes du colloque international sur les problèmes de balisage de dictionnaires électroniques, Limoges, 19–20 novembre 1998, édités sous forme électronique en janvier 1999 par T. R. Wooldridge.

[8] —— «Conflits de génération et usages littéraires concurrents dans la première édition du Dictionnaire de l'Académie française, Paris, Coignard, 1694.», Conférence publique donnée à l'Université de Toronto, Trinity College, le 29 février 1996, édition électronique sur le site Académie de Toronto.

[9] —— «Variantes graphiques et norme orthographique dans la première édition du *Dictionnaire de l'Académie Françoise* (1694)», Journée internationale des Dictionnaires de Cergy le 17 mars 1999 (article paru sur Internet avant la publication papier des *Actes*).

[10] —— «Modalités de repérage des informations paradoxales présentes dans le *Dictionnaire de l'Académie française* (1694) tiraillé entre synchronie et diachronie», édition électronique sur le site Académie de Toronto en 1997.

[11] —— «L'informatisation de la pré-édition du *Dictionnaire de l'Académie française* (1687) est-elle à l'ordre du jour ?», édition électronique sur le site Académie de Toronto en 1998.

[12] —— «La construction de la base informatisée des huit éditions du *Dictionnaire de l'Académie française*, 1694–1935», in Actes du colloque international sur *Construction et utilisation de grands corpus. Les grands corpus diachroniques*, organisé par H. Huot, à Paris VII, 24–27 sept. 1997, *Revue française de linguistique appliquée*, vol. IV, 1 *Grands corpus : diversité des objectifs, variété des approches*, juin 1999, p. 47–55.

[13] —— «Les acquis des premiers travaux effectués concernant les différentes formes d'édition électronique des dictionnaires anciens et les projets en cours.», in Actes du colloque NTIC-SHS organisé à Lyon, le 27 mars 2001 sur *Les nouvelles technologies de l'information et de la communication et les sciences humaines*, Lyon, Service commun de la Recherche, 2002, p. 25–36.

[14] —— «Lexicographie et métalexicographie, lexicologie et métalexicologie : la métalexicologie, une discipline née des réflexions métalexicographiques sur corpus électroniques», colloque international de Potsdam «Histoire de la linguistique en textes et concepts», 15–17 novembre 2001 (texte à paraître dans les Actes).

[15] —— «Numérisation et édition électronique des dictionnaires anciens : éthique et pratiques pour le respect des livres et de leurs consultants», actes du Colloque organisé sur le thème *Éthique Numérique*, par le Centre de Recherche en Information et Communication de l'université de Toulon, Saint-Cyr sur Mer, 12–14 mai, publication électronique sur `http://www.colloque-ethique-numerique.com`.

[16] I. Turcan et J. Ph. Saint-Gérand, «L'électronique et Internet au service de la diffusion des savoirs et du rayonnement de la langue française.» in Actes du colloque international *Foire des Études françaises*, Foire 2000, édition électronique double sur le site de T. R. Wooldridge à Toronto et sur le cédérom *Foire des Études françaises*, Toronto, mai 2000.

[17] —— «Une bonne version électronique de dictionnaires anciens ? Définitions, objectifs et risques : manifeste pour LE respect des consultants», texte publié sur «Le Net des Études françaises : ACRE» (Toronto) et en annexe au texte donné au colloque NTIC-SHS organisé à Lyon, en mars 2001 sur *Les nouvelles technologies de l'information et de la communication ...*

[18] I. Turcan et J.-M. Pierrel, «Analyse d'expériences de numérisation et projet d'avenir en réédition électronique de textes et de dictionnaires», à paraître en septembre 2003, dans le numéro thématique de la revue *Document Numérique* consacré au thème «Valorisation du patrimoine littéraire et linguistique».

[19] 2000 : Publication du cédérom *Le Dictionnaire de l'Académie française*. Les huit éditions complètes depuis 1694, consultables sur un même écran et réunies dans un cédérom unique, avec possibilités multiples d'affichages, de requêtes et de récupération des résultats. réalisé sous la direction scientifique d'I. Turcan, avec une présentation historique et critique. Éditions Redon (le texte de présentation du cédérom est également publié sur le site `http://www.dictionnaires-france.com`).

[20] 2001 : Publication du cédérom *Le grand atelier historique de la langue française*, cédérom réunissant les éditions électroniques des grands dictionnaires des XVII^e^, XVIII^e^ et XIX^e^ siècles, avec en particulier les ouvrages suivants : le *Dictionnaire françois* de P. Richelet (1679–80), le *Dictionnaire étymologique ou Origines de la langue françoise* de G. Ménage (1694), le *Dictionnaire des arts et des sciences* de Th. Corneille (1694), le *Dictionnaire critique* et le *Dictionnaire grammatical* de J. F. Féraud (1787–88) et le *Dictionnaire Universel François & Latin* de Trévoux dans l'édition de 1743 avec le *Supplément* de 1752, Editions Redon, Marsanne : réalisé sous la direction scientifique d'I. Turcan, avec un texte de présentation (introduction générale, méthodologique, historique et critique), documents annexes et listes des auteurs sources pour les dictionnaires de Furetière, Ménage, Corneille et Trévoux (le texte de présentation du cédérom est également publié sur le site `http://www.dictionnaires-france.com`).

[21] 2002 : Le *Grand Dictionnaire Universel* de Pierre Larousse, cédérom associant le mode image et le plein texte, réalisé sous la direction scientifique de J. P. Saint-Gérand, Redon-Robert (deux versions DVD / Cédérom) : texte de présentation et d'analyse de J. Ph. Saint-Gérand.

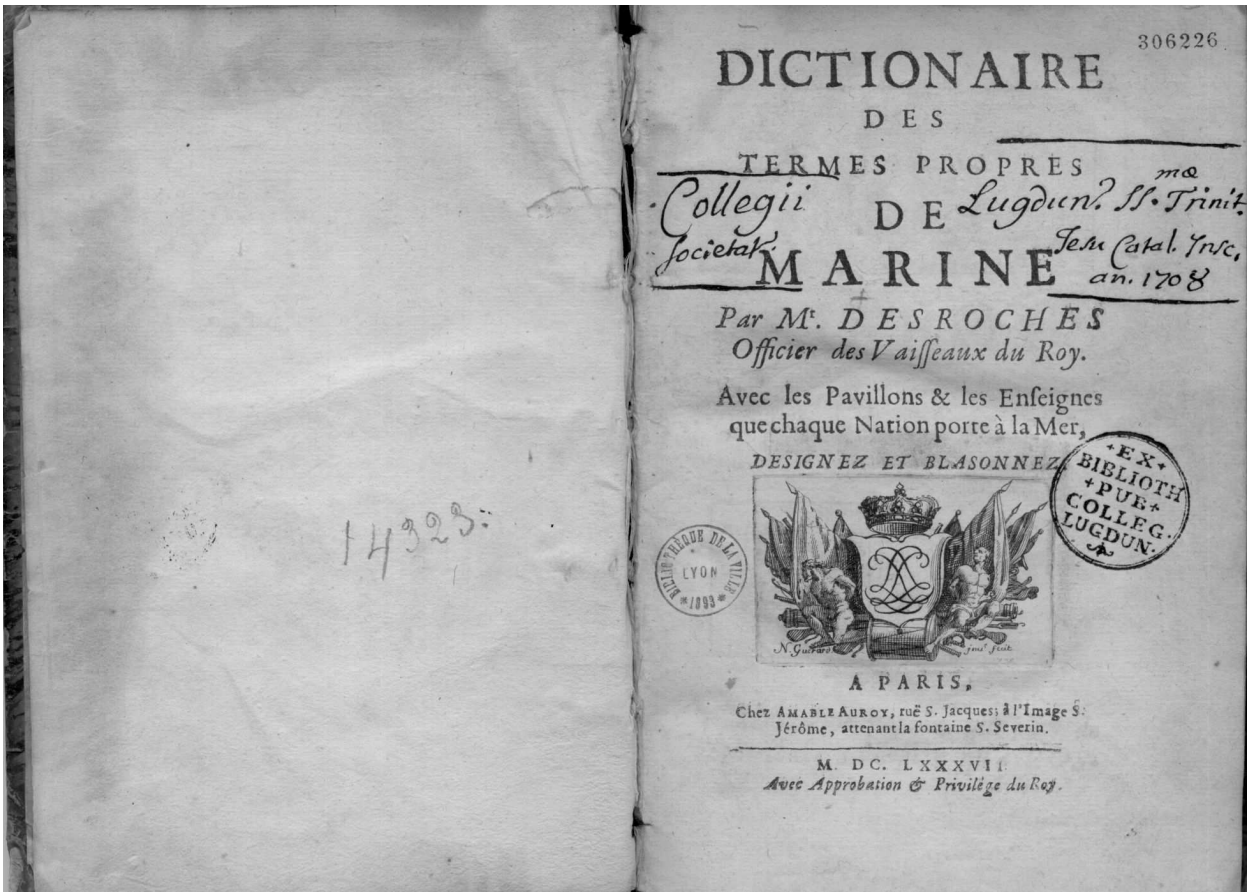


FIG. 1 : Exemplier 1a : repérage des champs dans une image.

Présentation des champs sous forme de tableau

		Cote du fonds ancien de la bibliothèque de Lyon
	Début de l'intitulé	
Première partie de l'*ex libris*	Suite de l'intitulé	Seconde partie de l'*ex libris*
	Nom et fonction de l'auteur	
	Complément de l'intitulé	
Tampon moderne de bibliothèque	Gravure indexée	Tampon ancien (= *ex libris*)
	Adresse de l'édition	
	Date et privilège	

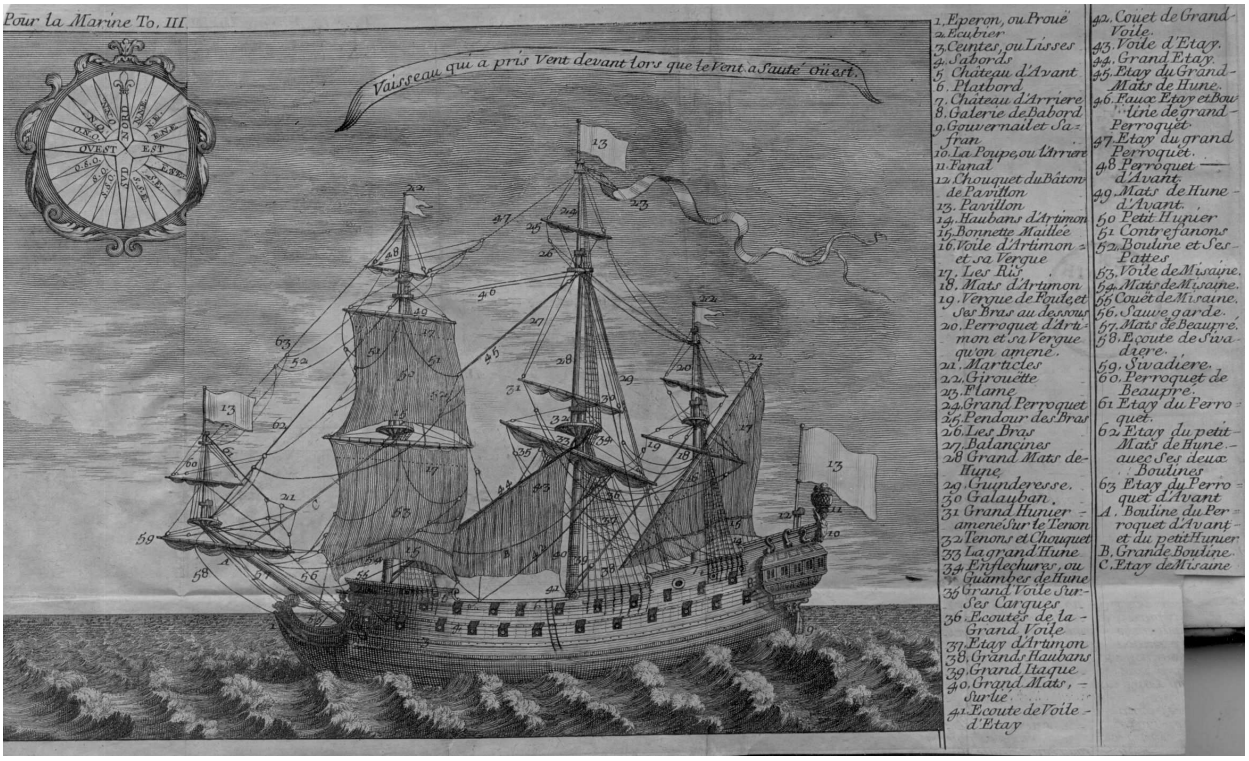
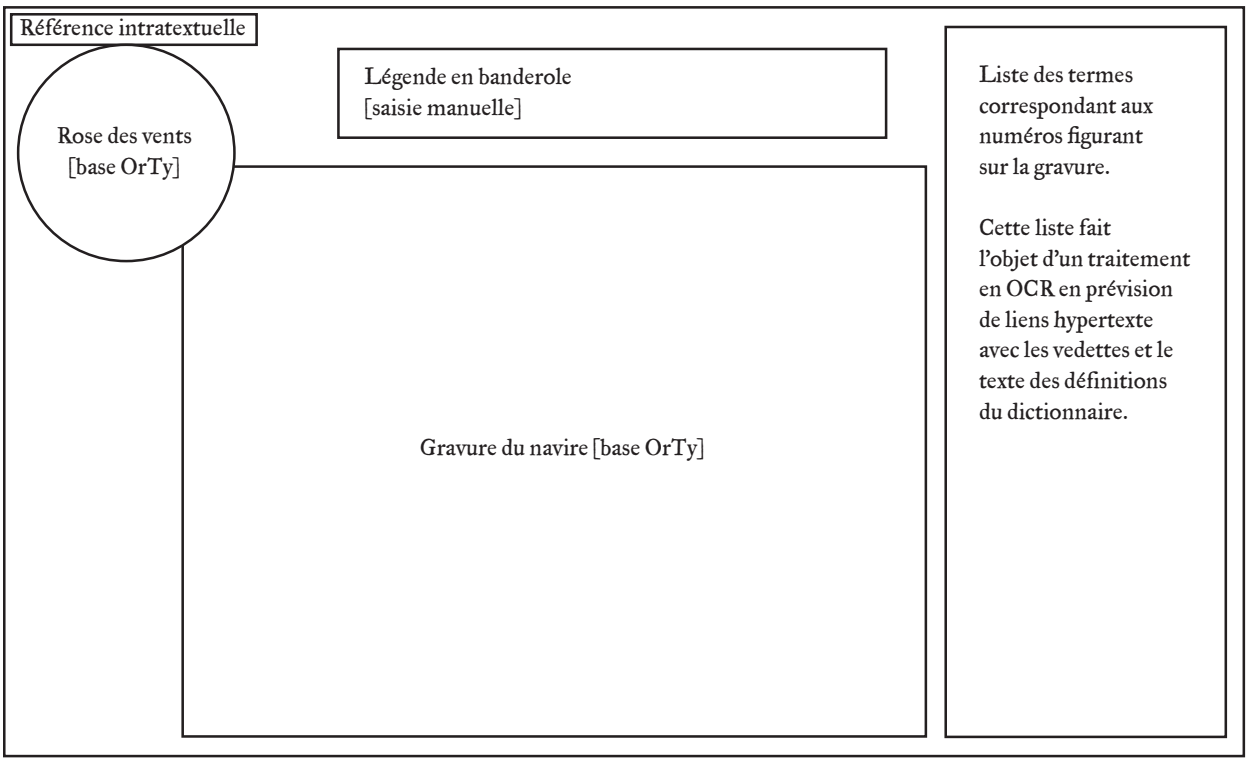


Fig. 2 : Exemplier 1b : repérage des champs dans une image

Présentation des champs de l'ensemble de l'illustration et choix de traitement



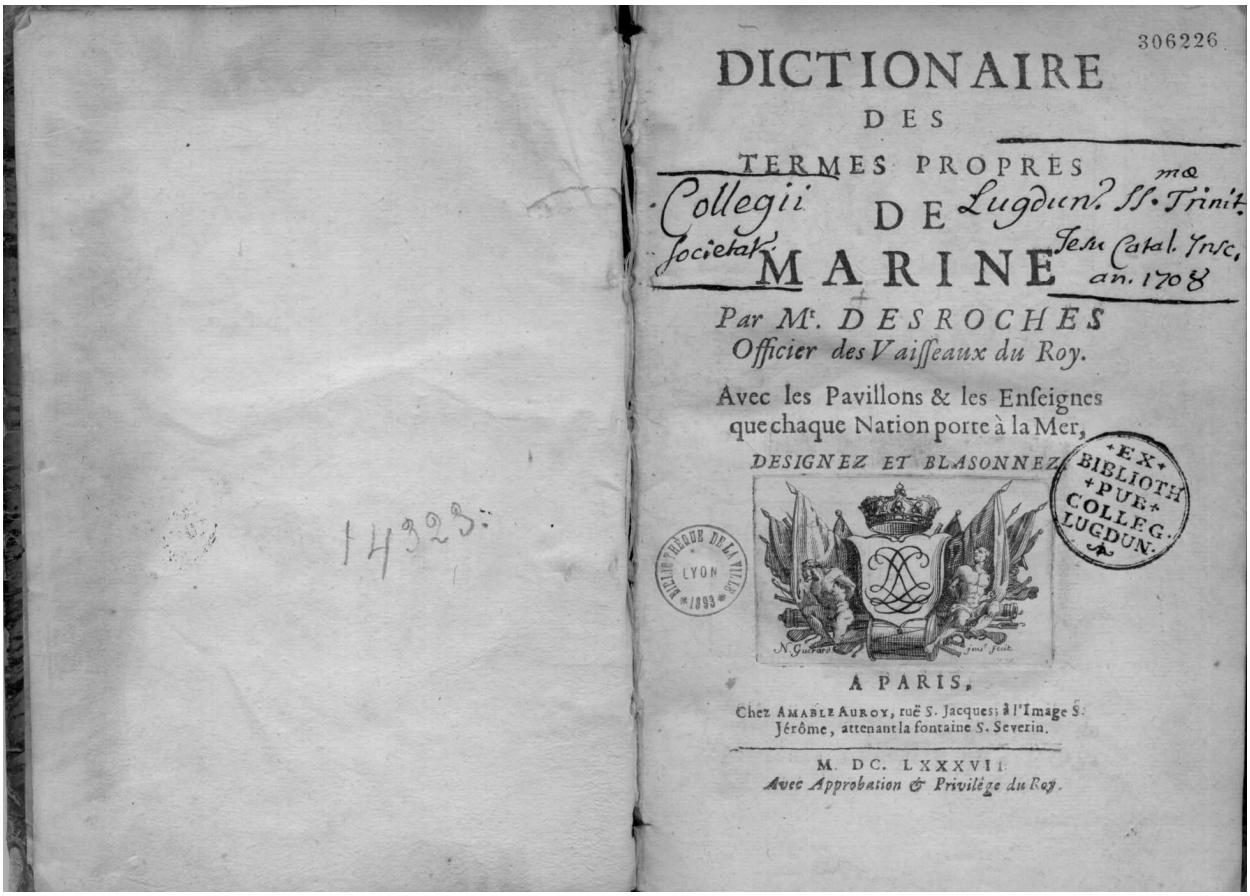
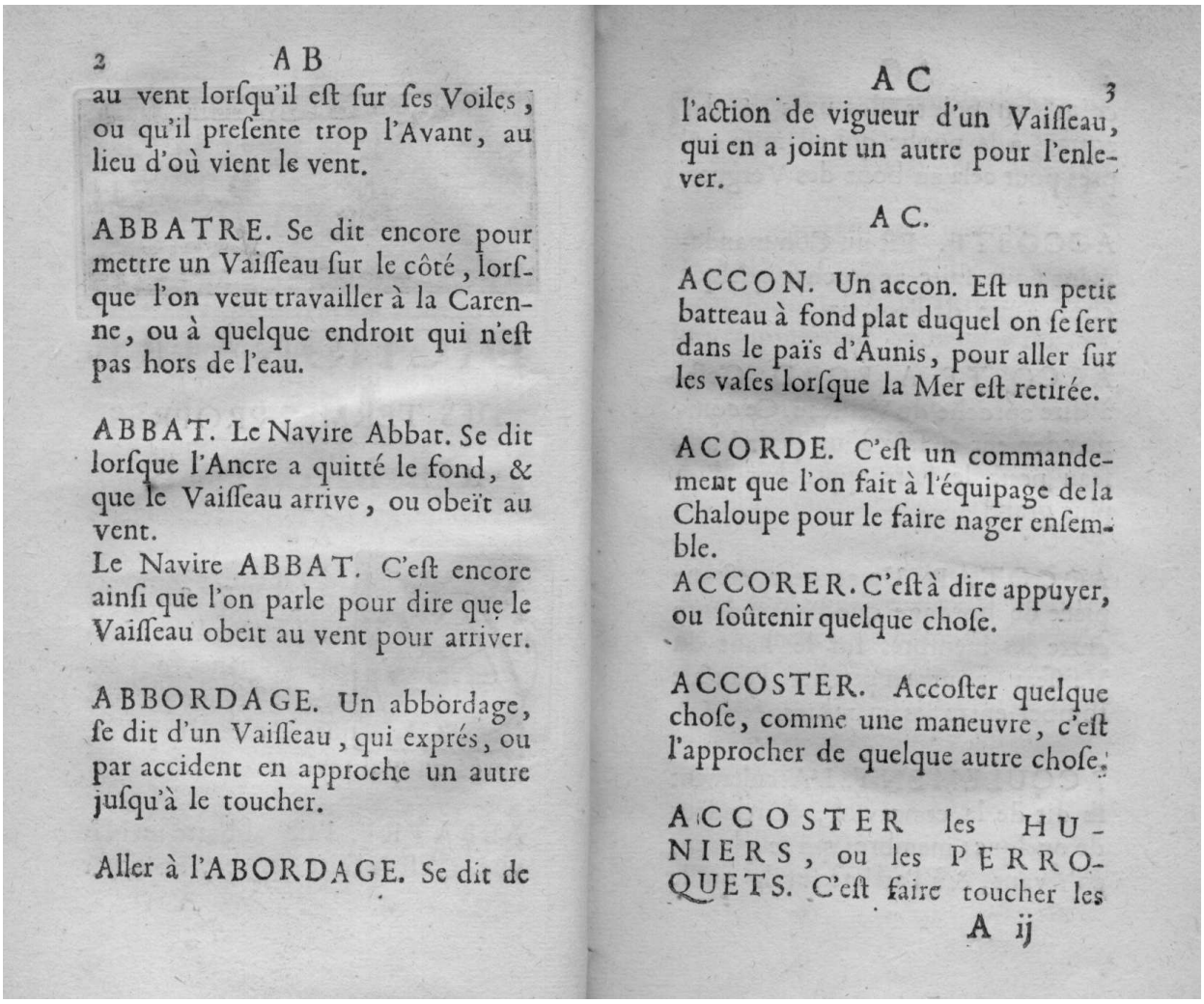


Fig. 3 : Exemplier 2 : traitement des champs dans une image

Présentation du traitement différencié selon les champs

		Cote du fonds ancien de la bibliothèque de Lyon
	Début de l'intitulé (OCR)	
Première partie de l'*ex libris* (saisie manuelle)	Suite de l'intitulé (OCR)	Seconde partie de l'*ex libris* (saisie manuelle)
	Nom et fonction de l'auteur (OCR)	
	Complément de l'intitulé (OCR)	
Tampon moderne de bibliothèque	Gravure indexée (base OrTy)	Tampon ancien (= *ex libris*) (saisie manuelle)
	Adresse de l'édition (OCR)	
	Date et privilège (saisie manuelle)	



2 AB

au vent lorſqu'il eſt ſur ſes Voiles, ou qu'il preſente trop l'Avant, au lieu d'où vient le vent.

ABBATRE. Se dit encore pour mettre un Vaiſſeau ſur le côté, lorſque l'on veut travailler à la Carenne, ou à quelque endroit qui n'eſt pas hors de l'eau.

ABBAT. Le Navire Abbat. Se dit lorſque l'Ancre a quitté le fond, & que le Vaiſſeau arrive, ou obeït au vent.

Le Navire ABBAT. C'eſt encore ainſi que l'on parle pour dire que le Vaiſſeau obeit au vent pour arriver.

ABBORDAGE. Un abbordage, ſe dit d'un Vaiſſeau, qui exprés, ou par accident en approche un autre juſqu'à le toucher.

Aller à l'ABORDAGE. Se dit de

AC 3

l'action de vigueur d'un Vaiſſeau, qui en a joint un autre pour l'enlever.

A C.

ACCON. Un accon. Eſt un petit batteau à fond plat duquel on ſe ſert dans le païs d'Aunis, pour aller ſur les vaſes lorſque la Mer eſt retirée.

ACORDE. C'eſt un commandement que l'on fait à l'équipage de la Chaloupe pour le faire nager enſemble.

ACCORER. C'eſt à dire appuyer, ou ſoûtenir quelque choſe.

ACCOSTER. Accoſter quelque choſe, comme une maneuvre, c'eſt l'approcher de quelque autre choſe.

ACCOSTER les HUNIERS, ou les PERROQUETS. C'eſt faire toucher les

A ij

Fig. 4 : Exemplier 4 : le traitement d'un fichier image en fichier texte

1. Premier passage d'OCR, sans apprentissage

2 AB
au vent lorfqu'il efr fur fes Voiles, ou qu'il prefente trop l'Avant, au lieu d'où vient le vent.
A B B A T R E. Se dit encore pour mettre un Vaiffeau fur le côté , lori; que l'on veut travailler à la Carenne, ou à quelque endroit qui n'eil pas hors de l'eau.
AB BAT. Le Navire Abbar. Se dit lorf ue l'Antre a quitté le fond, & que Te Vaiffeau arrive , ou obeït au vent.
Le Navire A B B A T. C'efl encore aine que l'on parle pour dire que le Vaiffeau obeit au vent pour arriver.
A BBORDA GE. Un abbardage, fe dit d'un Vaiflèau , qui exprès , ou par accidcnt en approche un autre jufqu'à le toucher.

2. Fichier de balisage des entrées d'articles avec les vedettes reconnues par OCR

```
<VED>ABBATRE</VED>.
<VED>ABBAT</VED>.
<SVED>Le Navire ABBAT</SVED>
<VED>ABBORDAGE</VED>.
<SVED>
Aller à l'ABORDAGE</SVED>
<VED>ACCON</VED>.
<VED>ACORDE</VED>.
<VED>ACCORER</VED>
<VED>ACCOSTER</VED>.
<SVED>ACCOSTER les HUNIERS , ou
les PERROQUETS</SVED>
```

XLaTeX, a DTD/Schema Which is Very Close to LaTeX

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
Yannis.Haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW Sydney NSW 2052, Australia
plaice@cse.unsw.edu.au
http://www.cse.unsw.edu.au/~plaice

Abstract

Our main idea is to change the LaTeX document preparation process by migrating to XML input (with eventual LaTeX code insertions via XML processing instructions). To do this, we need a DTD or an XML Schema which is very close to LaTeX syntax, so that users do not need to learn keywords anew. The keywords which used to be LaTeX commands and environments now become elements, attributes, namespaces (and eventually entities, as long as we still deal with DTDs). The advantage of this method is that one can use XML tools to validate and process documents before typesetting.

Résumé

L'idée centrale de XLaTeX est de changer le processus de préparation de document à la LaTeX, en migrant vers XML (avec toujours la possibilité d'inclure du code LaTeX dans le code XML en passant par des instructions de traitement). Pour ce faire, nous allons utiliser une DTD ou un schéma XML, très proches de la syntaxe de LaTeX, de manière à ce que les utilisateurs de XLaTeX n'aient pas à apprendre de nouveau les mots-clés. Ainsi, les mots-clés utilisés, jusqu'à maintenant, dans les commandes et environnements LaTeX seront dorénavant des éléments, attributs et espaces de nommage XML (ainsi qu'éventuellement des entités, tant que l'on utilise encore des DTD). L'avantage de cette méthode est que l'on puisse utiliser toute la panoplie d'outils XML pour valider et traiter de documents avant leur composition.

SGML and XML

The author (YH) first heard about SGML at the 1991 DANTE meeting in Vienna when a group of SGML evangelists presented it to the TeX community. Klaus Thull, sitting next to him, reacted by saying "what do we need another markup system for? we already have LaTeX..."

LaTeX is indeed a markup system, and one can even say it is easy to parse: when writing `\begin{center}` ... `\end{center}` it is quite clear where the "centered" block of text starts and where it ends — also it is quite easy to extract the "tag name:" `center`, since this name is made out of letters, delimited by braces.

But LaTeX has a major drawback: it is based upon TeX, and the latter is one of the hardest programming languages to parse, especially when one is playing with weird macro definitions, or with changing catcodes.

As with most "drawbacks", there are always people for whom they are advantages rather than drawbacks. In some cases it is a quite interesting feature of LaTeX to be able to go back to TeX for obtaining special effects, or for doing things that occur only once "manually". One can hardly prevent people from using TeX code in a LaTeX document. In fact there is no way to tell if a LaTeX document is a "good one" (in SGML jargon, a "valid one"), other than compiling it and seeing if it produces errors or warnings.

SGML has such a mechanism: there are programs called "validators" which can assert weither a given document is "valid" or not, for a given set of rules, called a "document type definition" (DTD).

But SGML has other drawbacks: it uses escape characters which can be different in each document (a phenomenon similar to changing catcodes in TeX), and,

worse, it uses "optional" tags: the DTD can declare that the existence of a given tag can be deduced from its context, and hence it does need to be explicitly written. Both of these features make the presence of a DTD absolutely necessary to be able to *parse* SGML documents correctly (and not only for validating purposes). These features make SGML documents hard to parse and hard to process.

A solution to these problems came with XML (see [7, 1]). It is easy to underestimate the difference between the two systems. Indeed, when comparing SGML and XML documents, they seem quite alike. In fact the philosophy is entirely different: in XML, no part of the markup is ever hidden or ambiguous. Escape characters are uniquely defined and no tags can ever be optional. DTDs (and their successors: XML schemas) are not necessary to parse an XML document, they are only needed to validate it.

The principles of XLaTeX

When using LaTeX for more than 15 years, like the authors, command and environment names, as well as the general syntax, become as automatic as driving a car, putting clothes on, eating and drinking, etc. Braces and backslashes are part of everyday life, and we can hardly imagine it without them.

For the first author, living in France and using a Macintosh, it is quite funny to note that none of these symbols is available on his keyboard (or any of the Macintosh keyboards he used in the last 20 years). The braces are obtained by 2-keys combinations (alt + parentheses), and the backslash by a 3-key combination (alt + shift + slash).

The backslash is quite a strange symbol: in most languages it doesn't even have a real name (in French it is usually called "barre oblique inverse", lately a name has been invented for it: "contre-oblique"). Legend [3, p. 29] says it was introduced into ASCII only as a graphic complement of "/" in order to obtain symbols $\wedge$ (= "/" + "\") and $\vee$ (= "\" + "/"). As all legends, *se non è vero, è ben trovato*, since the author could not find any rational use of the backslash symbol before the arrival of programming language syntax (besides its use in set theory, which is quite limited).

Is the TeX community a community of backslash worshippers? Not necessarily, although a document full of backslashes certainly feels "like home" for many of us.

Nevertheless TeXists are aware of the disadvantages of TeX syntax. While it is a general rule that commands start with a backslash, some commands (like ~) do not have a backslash, and sometimes (as in \\) a backslash is not used as an escape character. Command arguments are generally delimited by braces, but sometimes also by brackets — and sometimes commands have no arguments at all and must be included into groups (as in the case of `\small`). Some commands (like `\verb`) can even use *any* character as argument delimiter (and when we say *any*, we mean any; even in the case of paired delimiters, like parentheses or braces, one must use the left one on the left and the left one again on the right: `\verb=bla=` is right, `\verb{bla}` is wrong, `\verb{bla{` is right).

This is the only the tip of the iceberg of TeX syntax problems. On a more philosophical level, a big disadvantage of LaTeX (considered as a markup language), is the fact that there is no clear distinction between data and markup. When writing `\begin{center}` is is clear that `center` is part of the markup (the "tag name"), while in `\emph{hello}`, `hello` is data. But what about:

```
\textcolor{red}{green}
```

Will this produce the word "green" in the color red, or the word "red" in the color green? The author of the `color` package has decided that the first argument is markup and the second data, but there is no way for parsing software to guess it. Not to mention the fact that there are commands producing data (like `\today`) and others changing the status of data (like the % character, or the `comment` environment, or — in a more TeX-like fashion — the `\bdef` and `\edef` commands which convert a string into a command, or the `\token` command which converts a command into a string...).

Using software like *latex2html* one realizes that parsing TeX code is a perilous daredevil project. In fact only TeX can parse TeX code well. This is quite fair in a world where TeX files are written for the sole purpose of being compiled, but in the current era of electronic documents this can hardly be the case anymore. Nowadays documents are used in many different ways: they can be parsed, transformed, translated, re-assembled, etc.

For this to happen, a stable and simple markup system like XML is much more suitable than LaTeX.

But shall LaTeXists learn an entirely new syntax? Of course not. Only the basic syntactic rules should change: "less than" and "greater than" instead of backslash and braces, "elements" and "attributes" instead of "commands" and "environments".

The XLaTeX proposal is the following: *a set of XML elements and attributes (= a DTD or an XML schema), with tag names as close as possible to LaTeX command and environment names, easily convertible to LaTeX syntax.*

Tag names like `document`, `maketitle`, `center`, `quotation`, `itemize`, `enumerate`, `emph`, `footnote`, `chapter`, `section`, are used on a daily basis by all LaTeX users. They remain unchanged for XLaTeX. For example, the LaTeX code:

```
\begin{quotation}
Life shall go on\footnote{Said
```

```
\emph{he}.}.
\end{quotation}
```

becomes in XLaTeX:

```
<quotation>
Life shall go on<footnote>Said
<emph>he</emph>.</footnote>.
</quotation>
```

Elements or attributes? An attribute has a tag name and contents. It belongs to an element (and is actually written inside the opening tag of the element). The order of attributes is not relevant, but there cannot be two attributes with the same name in the same element. The contents of an attribute cannot include the character <, and hence cannot include element tags.

Attributes are used for metadata, that is data which can be considered as being either markup or contents. For example, if `<section>` is the opening tag of a section title which will be automatically numbered, then one could imagine `<section number="3">` as the opening tag of a section numbered "3". Is this number "3" part of the contents of the document? The answer is not clear.

Attributes are very useful when we have variable markup. The typical example is the format of a `tabular` environment. This format is different for each table, but it is nevertheless pure markup (nobody would like `|c|c|p{2cm}|` to appear in her document). Here is what an XLaTeX `tabular` environment/element looks:

```
<tabular format="|c|c|p{2cm}|">
A<tab/>B<tab/>C<br/>
D<tab/>E<tab/>F<br/>
</tabular>
```

We notice two things: first of all, attributes have names, so every LaTeX command argument becoming an attribute needs a name. "Format" seems to be the natural name for the format of a tabular environment. Secondly, the special character & and the command \\ have been replaced by elements: `<tab/>` and `<br/>`.

Sometimes the choice between element and attribute is not clear. Let us take for example the optional argument of the `\section` command (the version of a title used in the table of contents). It seems natural to write:

```
<section toc="Short version">Long version
</section>
```

But what happens when we need further markup inside such a title? If the long title contained, for example, an `<emph>` element, then this element could not be used in the short one, since an attribute may not contain tags. There are two solutions, neither entirely satisfactory:

1. use an element instead of an attribute, for example:

   ```
   <section>
   <toc>Short <emph>version</emph></toc>
   Long <emph>version</emph>
   </section>
   ```

2. use LaTeX commands instead of XML markup in the attribute:

   ```
   <section toc="Short \emph{version}">
   Long <emph>version</emph>
   </section>
   ```

How about TeX code? One may argue that by "writing LaTeX in XML", one can only use pre-defined elements, and hence one loses all the flexibility of TeX code. XML provides a very simple and natural mechanism to switch between syntaxes: *processing instructions*. In XLaTeX one can switch to TeX code at any moment, via the `tex` processing instruction:

```
<footnote>This symbol was
very <emph>scary</emph>
and looked like an
<?tex {\xx\char'124}?>.
</footnote>
```

Elegant XLaTeX code would, of course, rather try to avoid such processing instructions. As always in LaTeX, TeX code should be used only when unavoidable. But in XLaTeX such code is clearly marked and will be avoided by XML parsers. There is only one hitch: the string "?>" should never appear inside the TeX code, since it is the processing instruction escape sequence.

Other processing instructions used are `math` (for math formulas), `displaymath` (for display mathematics), `verb` (for short verbatim, similar to the `\verb` command), `verbatim` (for long verbatim code), `special` (similar to `\special` command).

Using processing instructions has the advantage that one doesn't need to care about protecting the characters <, >, & (only the sequence ?> must be avoided). But it also has a serious disadvantage: the data included in the processing instruction is not considered as contents of the document. In some cases this seems the right approach: in LaTeX one would hardly put textual contents into a `\special` command, although this is theoretically possible — thus, using processing instructions for specials will most probably not "hide" any contents of the document.

This is less clear with, for example, verbatim code or math formulas (although in the latter case one could as well also use MathML as the proper way of writing mathematics with XML). For that reason XLaTeX also provides XML elements for math formulas, verbatim code and specials. When using these elements, one must always protect characters <, >, &, by using the appropriate entities (`<`, `>`, `&`). As an example, to obtain the output:

```
<textbf>this is cute</textbf>
```

one can write either:

```
<?verbatim
<textbf>this is cute</textbf>
?>
```

or:

```
<verbatim>
&lt;textbf&gt;this is cute&lt;textbf&gt;
</verbatim>
```

The latter solution is "cleaner", but the former is more readable.

Other similar projects and history of XLaTeX In November 1998, Doug Lovell from IBM AlphaWorks released a package called *texml* to translate XML into TeX [5]. This package is described as *a three-part solution that provides a path from XML into the TeX formatting language*. This project is described in a *TUGboat* article [6]. It has been retired from IBM and appeared on Sourceforge in 2004 [8] (developer: Oleg Paraschenko). In 1999, Stefan Krauß, from Stuttgart University, starts a similar project called *SESAMDoc* [4].

Both TeXML and *SESAMDoc* use an approach quite different from ours. Instead of defining elements with names similar to those of LaTeX commands and environments, they define elements from commands and environments, where the names appear in an attribute. For instance, instead of writing

```
<emph>bingo!</emph>
```

as we are, *SESAMDoc* would write:

```
<cmd name="emph">
   <param>bingo!</param>
</cmd>
```

(the TeXML code would be similar except for the spelling `parm` instead of `param`).

TeXML and *SESAMDoc* are less suitable for manual keyboarding and editing than XLaTeX. By systematically using `param`/`parm` elements, one loses the distinction between data and markup/metadata. In XLaTeX it is possible to write arbitrary commands and environments that way, but one can also use attributes for LaTeX arguments, and mix the two approaches so that data gets into element contents and metadata/markup into attributes.

Compare the XLaTeX approach:

```
<com name="textcolor" arg1="red">
<arg2>This text is typeset in red.</arg2>
</com>
```

where argument 1 (whose value is metadata) is an attribute while argument 2 (whose value is textual content) is a sub-element, to the TeXML approach:

```
<cmd name="textcolor">
<arg1>red</arg1>
<arg2>This text is typeset in red.</arg2>
</cmd>
```

where there is no qualitative distinction between "red" and "This text is typeset in red".

The XLaTeX project began in late 2002 as a Diploma Project for Paweł Grams, at that time student of ENST Bretagne. He presented his work at the 2003 GUST meeting in Bachotek [2].

In the following section we describe the XLaTeX (version 1) syntax.

XLaTeX v. 0.9 syntax

Namespace The namespace of XLaTeX v. 0.9 is:

```
http://omega.enstb.org/2003/XLaTeX
```

Conventions Ages before the arrival of Unicode, Knuth introduced some easy ways to obtain characters not in ASCII: `` and '' for the double quotes, ` and ' for single quotes, -- and --- for en-dash and em-dash, `? and `! for Spanish inverted punctuation. The most frequently used of these is ' which produces an "apostrophe" (a raised comma) although the character used in the document is an "ASCII apostrophe" (a small straight line).

Packages like *babel* and Omega Translation Processes have introduced new conventions: for example, in French one leaves a blank space in front of double punctuation, this space is converted into a non-breakable space (in the case of colon) or into a thin space (in all other cases).

Such conventions were invented almost a century ago, when the typewriter began to be used. Going from a full blank space to a thin space is the same as going from the typewriter's world ("dactylography") to the printer's ("typography").

One may argue whether these conventions should be left in XLaTeX or not. They are part of TeX and our fingers are used to them, especially if we consider ourselves as being dactylographers and TeX as the typographer-in-the-box. On the other hand, XML and hence XLaTeX is based on Unicode, and this encoding contains all of these characters. The problem is not anymore to get the characters in the document, but to configure our keyboard to produce them easily. And the final argument is that even in TeX these conventions were deactivated in some contexts, for example in verbatim environment or when using a typewriter font.

To give the future (that is: Unicode) a chance we have chosen not to activate these conventions by default. They can be activated, though, using attribute `tex-conventions` which can take values `on` and `off`. The value of this attribute is inherited by children of a node, like XSL-FO properties.

Encodings The default encoding of XML (and hence of XLaTeX) is Unicode UTF-8. This encoding can be

changed through the `encoding` pseudo-attribute of the XML declaration, but we do not advise the user to do so.

Global document structure XML documents have a tree structure: they need a *single* top node. LaTeX documents have two parts: the preamble (which has no top node) and the document body (which has the top node `document`). To obtain a structure similar to LaTeX, we have to introduce an additional node above `document`. It is only natural for us to call this node `xlatex`.

On the other hand, every LaTeX document has one and only one `\documentclass` command. There are two ways to translate this into XML: either as an element under `xlatex`, or as an attribute of `document` (in the latter case, the value of this attribute is the name of the document class, and a second attribute `options` includes the eventual class options).

In a LaTeX preamble one finds a lot of code but mostly `\usepackage` commands. These can be included in an XLaTeX document as `usepackage` elements (with self-explanatory `name` and `options` attributes). These elements can be used either under `xlatex` and before `document` or directly under `document`.

Hence a typical LaTeX document like:

```
\documentclass[11pt]{article}
\usepackage[francais]{babel}
\usepackage[dvips]{graphics}
\begin{document}
...
\end{document}
```

can become (for LaTeX purists):

```
<?xml version="1.0"?>
<xlatex version="0.9">
<documentclass name="article" options="11pt"/>
<usepackage name="babel" options="francais"/>
<usepackage name="graphics" options="dvips"/>
<document>
...
</document>
```

or (more in XML style):

```
<?xml version="1.0"?>
<xlatex version="0.9">
<document class="article" options="11pt">
<usepackage name="babel" options="francais"/>
<usepackage name="graphics" options="dvips"/>
...
</document>
```

In the latter case, the `\usepackage` instructions will be placed at the beginning of the preamble before any code included under `xlatex` and before `document`.

It becomes obvious from the example above that XLaTeX to LaTeX translation is not trivial and requires more than one parsing run. In the next subsection this will be even more clear.

Languages Using the *babel* package, languages are first declared (as options of the `\usepackage` command) and then activated through the `\selectlanguage` command. This approach is made possible in XLaTeX:

```
<?xml version="1.0"?>
<xlatex version="0.9">
<usepackage name="babel"
   options="francais,english"/>
<document class="article"
   options="11pt">
Are we writing in Shakespeare's
language?
<selectlanguage name="french">
<emph>Ou est-ce dans la langue
de Molière ?</emph>
</selectlanguage>
</document>
```

But there is also a different approach, more XML-oriented. In XML, there is a standard way to specify the language used in an element: the `xml:lang` attribute. Values of this attribute are combinations of standard 2-letter language names (ISO-639) and 2-letter country names (ISO-3166), separated by a dash.

One can consider — although this is not stated in the XML specifications — that the value of this attribute is inherited by nodes underneath the element carrying it.

Every XLaTeX element can carry the `xml:lang` attribute and there is no need to declare the *babel* package with the appropriate language. The XLaTeX to LaTeX parser will find all occurences of the attribute and load the corresponding languages in the document preamble. Hence the example above could also be written as:

```
<?xml version="1.0"?>
<xlatex version="0.9">
<document class="article"
   options="11pt" xml:lang="en">
Are we writing in Shakespeare's
language?
<emph xml:lang="fr">Ou est-ce dans
la langue de Molière ?</emph>
</document>
```

Correspondence between values of the `xml:lang` attribute and *babel* language names is included in the XLaTeX configuration file `xlatex.conf`.

Sections, text styles, footnotes, lists, tables The LaTeX commands `\section` and the like become XLaTeX elements, containing section titles. Attribute `short` can contain a shorter version of the title for table of contents and/or headers (depending on the style file).

The LaTeX commands changing text style (`\emph`, `\textbf`, and the like) become XLaTeX elements. There is also a neutral element, used to carry attributes such as `xml:lang`; it is called `span` (as in HTML).

Footnotes are obtained with the `footnote` element. As in LaTeX there are also `footnotenumber` and `footnotetext` elements, but they should not be needed, as the XLaTeX to LaTeX translator should be able to replace a `<footnote>` element in an inappropriate LaTeX environment (such as a table) by paired `<footnotenumber>` and `<footnotetext>` elements. In other words, the XLaTeX to LaTeX translator should be able to rectify some of LaTeX's deficiencies (or at least to act as if these deficiencies were not there).

Lists are obtained through `itemize`, `enumerate` and `description` elements. Each list item is contained in an `item` element. This element can carry a `mark` attribute, containing the list item mark. If this mark contains a closing bracket, then it will be automatically converted into a `\char"5D` command (this is a well-known LaTeX problem coming from the fact that the mark is an optional argument of the `\item` command and hence is delimited by brackets instead of braces).

Tables are obtained through the `tabular` element, which takes two attributes: `format` and `pos`. When the `format` attribute contains values such as `m` or `b` then the `array` package is automatically loaded. When it contains the value `X` then the `tabularX` environment is automatically loaded. Cells are separated by the `tab` element, and ends of line are given by the `br` element. Horizontal lines are included by the `hline` element.

Multicolumn cells are obtained by the element `multicolumn`, which takes two attributes: `num` (the number of columns) and `format` (the format of the cell). The contents of the `multicolumn` element is the contents of the cell. Partial horizontal lines are included by the `cline` element which carries a `num` attribute.

Here is an example of a table with all the features described:

One	Two	Three
Four	Five	
Six	Seven	Eight

```
\begin{tabular}{|c|c|c|}\hline
One&Two&Three\\\hline
Four&\multicolumn{2}{c|}{Five}\\\cline{2-3}
Six&Seven&Eight\\\hline
\end{tabular}
```

```
<tabular format="|c|c|c|"><hline/>
One<tab/>Two<tab/>Three<br/><hline/>
Four<tab/><multicolumn num="2"
  format="c|">Five</multicolumn><br/>
  <cline num="2-3"/>
Six<tab/>Seven<tab/>Eight<br/><hline/>
</tabular>
```

Cross references There are two approaches to cross references: the LaTeX way and the XML way. The former is to use `label`, `ref` and `pageref` elements, carrying `id` attributes. The latter is to use `id` attributes instead of `label` elements. These attributes can be carried by any XLaTeX element.

Here is an example of these two approaches: a reference to a section title.

```
<section>Lyubov Bruk &amp; Mark Taimanov
<label id="bruk-taimanov"/></section>
```

is the "LaTeX way", and:

```
<section id="bruk-taimanov">Lyubov Bruk
&amp; Mark Taimanov</section>
```

the "XML way".

Mathematics, verbatim As already mentioned, mathematics and verbatim code can be included in two ways: either by XML processing instructions or by XLaTeX elements `math`, `displaymath`, `verb` and `verbatim`:

```
To calculate <math>\sqrt{2}</math>
use function <verb>sqrt(2)</verb>.
```

or:

```
To calculate <?math \sqrt{2}?>
use function <?verb sqrt(2)?>.
```

to obtain:

To calculate $\sqrt{2}$ use function `sqrt(2)`.

In the case of XML elements, the characters `<`, `>` and `&` must be entered as `<`, `>`, `&`. In the case of processing instructions the only constraint is that the string `?>` must not be included in the contents.

Let us emphasize that the XLaTeX to LaTeX translator does not produce `\verb` commands and `verbatim` environments from `verb` and `verbatim` elements or PIs. Instead it simply changes the font into a typewriter one and translates characters, in other words: we obtain the verbatim effect in a "manual way". This has the enormous advantage that verbatim code can be used everywhere, including in footnotes, tables, section titles and other places where it is prohibited in normal LaTeX.

Arbitrary commands and environments It may happen that a user wants to use a given LaTeX command which is not included in the XLaTeX DTD (or schema). In that case one can either use the `tex` processing instruction (which switches immediately into TeX mode) or elements `com`, `env`, `arg1`, ...`arg9` and `optarg`.

Here is an example: suppose that a user has defined a LaTeX command called `toto` with one optional argument and two mandatory ones. She wants to use it as follows:

```
\toto[red]{2}{Some text.}
```

There are three ways to obtain this code. By a processing instruction:

```
<?tex \toto[red]{2}{Some text.}?>
```

in which case an XML parser would not be able to parse the data properly, or by the `com` element and `arg*` attributes:

```
<com name="toto"
   arg1="2"
   arg2="Some text."
   optarg="red"/>
```

or by `com` and `arg*` elements:

```
<com name="toto"><arg1>2</arg1>
   <arg2>Some text.</arg2>
   <optarg>red</optarg>
</com>
```

The two approaches can be mixed so that the author of the document has full control of what is to be considered as markup/metadata, and what as data (textual content). In the case of our example, 2 and `red` are probably metadata while `Some text.` is obviously text. Hence, it would be more elegant to write:

```
<com name="toto" optarg="red"
   arg1="2"><arg2>Some text.</arg2>
</com>
```

To obtain an environment instead of a command, one uses the `env` element. The contents of the element is the contents of the environment.

The advantage of using elements instead of merely switching to TeX mode via a PI is that XML parsers or processors (like SAX/DOM or XSLT) can transform these elements into other XML elements, as desired — while this is hardly possible (or at least much more difficult) inside a processing instruction.

Graphics, figures, multiple columns, files The following elements produce *graphics* package commands:

```
<includegraphics src="toto.eps"
   bbox="50 70 327 655"/>
<scalebox amount="0.1"/>
<rotatebox amount="30"/>
<resizebox x="0.5" y="!"/>
```

Instead of using elements for scaling, rotating and resizing, one can also use attributes of the `includegraphics` element:

```
<includegraphics src="toto.eps"
   bbox="50 70 327 655"
   scale="0.1"
   rotate="30"
   resizex="0.5" resizey="!"/>
```

In that case, operations are done in the following order: first rotating, then resizing, and finally scaling.

Using either one of these elements automatically loads the *graphics* package.

Floating figures and tables are obtained by elements `figure` and `table` having a single argument `pos`. If an `H` is included in the value of `pos`, the *float* package is automatically loaded.

Captions are obtained by the `caption` element.

Multiple columns are obtained by the `multicols` element, which takes one attribute: `pos`. The *multicol* package is automatically loaded. One can also use elements `twocolumns` and `onecolumn` as in standard LaTeX.

To include files one can use `input`, `include` and `includeonly` elements (with `src` attribute, containing the file name).

Miscellanea The TeX, LaTeX, XLaTeX, METAFONT, etc. logos are obtained through the elements `<TeX/>`, `<LaTeX/>`, `<XLaTeX/>`, `<MF/>`, and so on. The `\today` command is obtained by the `today` element.

Bibliography, index Index entries are obtained by the `index` element, which can be used in three different ways:

1. empty, and with an `id` attribute:
   `<index id="horse"/>horses` is equivalent to:
   `\index{horse}horses`
2. non-empty without attribute:
   `<index>horse</index>` is equivalent to:
   `\index{horse}horse`
3. non-empty with an `id` attribute:
   `<index id="horse">horses</index>`
   is equivalent to:
   `\index{horse}horses`

The `printindex` command produces the index. The *makeidx* package is automatically loaded, and the `\makeindex` command automatically inserted. The `index` and `printindex` elements can also carry another attribute: `name`. In that case several indexes are built, identified by their "names". The *multind* package is automatically loaded.

Bibliographical references are obtained through the `cite` element which takes two attributes: `key` and `opt`. There is also a `nocite` element with `key` attribute. To obtain the list of bibliographical references one can use elements `bibliographystyle` (with attribute `src`) and `bibliography` (with attribute `src`). Instead of the `bibliographystyle` element, one can also use the attribute `style` carried by the element `bibliography`:

```
<bibliography
   style="plain"
   width="666"
   src="mybibliography"/>
```

The `bibliography` element can contain `bibitem` subelements. In that case it is converted into a `thebibliography` environment. `bibitem` elements contain `key` and `label` attributes.

Availability, further developments

XLaTeX is not yet stable, since we would like the TeX community to provide us with feedback and thorough testing before version 1.0 is released. Open questions remain, such as the additional packages that should be provided and automatically loaded as well as the exact features of the XLaTeX to LaTeX translator.

The first real-world document written in XLaTeX was the first author's book *Fontes et codages*, published by O'Reilly France in April 2004.

The up-to-date Web page of XLaTeX is `http://omega.enstb.org/xlatex`. A prototype XLaTeX to LaTeX translator (written in Perl) can be found on this page. All XLaTeX development is publically available (GNU copyleft license).

Once XLaTeX is stable we are planning to write an Omega input model (so that XLaTeX files can be read directly by Omega) as well as an XLaTeX mode for Emacs, XSLT code for converting XLaTeX into XHTML or XSL-FO, etc.

References

[1] Anelyse Boukhors, Alexandre Kaszycki, Jérôme Laplace, Sandrine Munerot, and Laurent Poublan. *XML, la synthèse*. Eyrolles, 2003.

[2] Paweł Grams. XML jako wejście Omegi. In *XI Ogólnopolska Konferencja Polskiej Grupy Użytkowników Systemu TeX*, 2003.

[3] Yannis Haralambous. *Fontes & codages*. O'Reilly France, 2004.

[4] Stefan Krauß. SESAMDoc, LaTeX-DTD für die Druckausgabe. `http://www.iste.uni-stuttgart.de/se/people/krauss/sesamdoc`, 1999.

[5] Douglas Lovell. IBM AlphaWorks TeXML. `http://www.alphaworks.ibm.com/aw.nsf/techreqs/texml`, 1998.

[6] Douglas Lovell. TeXML: Typesetting XML with TeX. *TUGboat*, 20(3):176–183, September 1999.

[7] W. Scott Means and Elliotte Rusty Harold. *XML in a Nutshell, manuel de référence*. O'Reilly France, 2e edition, December 2002. `http://www.oreilly.fr/catalogue/xml_nutshell_2.html`.

[8] Oleg Paraschenko. TeXML: an XML vocabulary for TeX. `http://sourceforge.net/projects/getfo/`, 2004.

Tralics, a LaTeX to XML Translator

José Grimm
INRIA
2002, Route des Lucioles, BP 93
06902 Sophia Antipolis CEDEX
Jose.Grimm@sophia.inria.fr
http://www-sop.inria.fr/miaou/Jose.Grimm

Abstract

In this paper we describe Tralics, a LaTeX to XML translator. A previous version of the software (written in Perl) was used to obtain the pdf version of Inria's "Rapport d'Activité" for year 2001. The current version of the software (written in C++) is used for both the HTML and pdf version for the year 2002. The XML generated by Tralics conforms to a local DTD, similar to the TEI; it was converted to pdf using pdfTeX and the `xmltex` package, and the HTML was obtained via an XSLT processor.

We explain here the philosophy of the software, its usage, limitations, and customization.

Résumé

Dans cet article nous décrivons le logiciel Tralics, un traducteur de LaTeX vers XML. Une version antérieure de ce logiciel, écrite en Perl, a été utilisée pour générer la version pdf du Rapport d'activité de l'Inria en 2001. La version actuelle du logiciel, écrite en C++, a été utilisée pour obtenir à la fois le HTML et le pdf de la version 2002 : nous avons utilisé une DTD locale, similaire à la TEI, et pdfTeX plus `xmltex` pour obtenir le pdf.

Nous expliquons ici la philosophie de Tralics, son usage, ses limitations, et comment paramétrer le logiciel.

Introduction

If you run Tralics on a document such as this article, you will get as output an XML document that starts and ends like this:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE eurotex SYSTEM 'tugboat.dtd'>
<!-- translated from latex by tralics 1.8a-->
<eurotex language='english'>
<titlepage>
<ti>Tralics, a <LaTeX/> to XML translator</ti>
<NetA>Jose.Grimm@sophia.inria.fr</NetA>
<U>http://www-sop.inria.fr/miaou/Jose.Grimm</U>
<resume><p>
In this paper we describe Tralics,
...
<byear>1999</byear>
</citation>
</biblio></eurotex>
```

All that is needed for this example to work is to put, in the configuration file read by Tralics, the following lines:

```
BeginType  eurotex
  DocType = eurotex tugboat.dtd
BeginTitlePage
  \maketitle <titlepage> "" ""
  \title <ti> "No title"
  \netaddress <NetA>  "No address"
  \personalURL <U> "No url given"
  \resume E<resume> "Pas de résumé"
  \abstract E<abstract> "no abstract"
  \author + <author> <auth> "No authors"
  \address p<address>  "no address"
End
End
```

Once you have an XML version of the document, you can use any tool you wish to process it (we used the Gnome library, `http://xmlsoft.org/`), but something has to be done. There are typically two kinds of applications. The first is that everybody publishing an Inria Technical Report has to give, together with the typeset PostScript document, the start of the LaTeX source, which is processed by an ad hoc tool that adds a new item to the publication database. One could use Tralics to convert the partial document, and an XSLT style sheet to extract from the `<titlepage>` element all relevant information: author, title, abstract, keywords, etc. In such a situation, the presence of an unknown element like `<LaTeX/>` or an unexpected math formula causes no problem; the element can be replaced by its name, and a math formula removed (the current process behaves this way).

A second type of application is the following: for

some reasons, the LaTeX document needs to exist in both PostScript and HTML form, and XML is used as an intermediate, common, format. In such a case there should be little difference between the PostScript as seen by the author (direct route) and the final version (the XML route). As a result, the translation of `\LaTeX` must match the LaTeX logo at best, and it is not possible to reject a math formula under the pretext that it does not fit in the MathML standard. As indicated on the figure on page 104 of [1], there are lots of possibilities, none of which is very simple. In practice, even more tools are needed if the objective is a complete web site.

This paper has two parts: we first explain the main application of Tralics (why a new program, how it is used, what is done with the resulting XML, etc.); after that, we shall explain what Tralics does in the same order as the *LaTeX Companion* [3].

Why another LaTeX to X translator?

Inria's annual activity report has a long history, as long as the history of Inria itself. In fact, each research team has to report, each year, on its activity; this set of documents, which we call the RA in what follows, is sometimes called the scientific annexes to the annual report (which is a bilingual document on glazed paper with lots of figures and photographs, produced by a specialized company).

Since 1987, the RA has been typeset by LaTeX, using a specific document style (or document class), see [7]. Since then, the number of pages has evolved from one thousand to three thousand, and the document is no longer printed on paper (a CD-ROM version has been available since 1999). On the other hand, an HTML version of the document has been available since 1994; it was produced by `latex2html` (using SGML as an intermediate language for the first year).

In the years 1997–98, some discussions showed that it would be nice to redesign the RA, using XML as the intermediate language. We contacted some companies, but got no good answer: nobody was able to translate the LaTeX input into an XML output, no guarantee was given about the quality of a printed version of the XML, and mathematical support was very poor. These three points will be discussed later on.

These discussions were not completely negative: a new concept emerged, the "raweb". Essentially, it is formed of "modules" that can be read independently (lots of nice properties of these modules are still to be implemented; for instance, depending on the user profile, these modules could be presented in different orders). Each module is equivalent to a web page, and can be put in a "classeur" (French equivalent of a loose-leaf file, it's like a caddy but elements are ordered; there is no official English name for it). The objects in the caddy can be re-ordered, and the caddy can be transformed into a single HTML document (in 1999, the caddy contained the LaTeX version of the modules, hence was able to produce a PostScript document; we hope that next year it will contain the XML version of the modules, but running pdfTeX on the web server is not an option).

Moreover, the notion of an RA-conforming document was formalized (and this served as the basis of the raweb DTD), and a syntax checker was written in Perl. This checker has some intrinsic limitations: since it was unaware of macro expansion and commands like `\csname`, it was very easy to fool it.

Our new translator

In fall 2001, we decided to fabricate a translator and to test it; the tool was based on `ltx2x` (a translator by Peter Wilson, available on CTAN) and Ω (a TeX extension that can produce MathML as a byproduct, [5]), together with a Perl script that was used as a syntax checker, a module splitter, and a pre- and post-processor. A second version of the tool, written entirely in Perl, was used for the pdf version of the RA2001.

The description of the software can be found in [4], together with lots of examples that could not be translated. For instance this one:

```
\catcode `$=\active %$emacs
\def$#1~{\catcode`$=3 zut}
$\left[1=2\right)$ is a formula~!
```

In May 2002, we decided to completely rewrite the software, and to call it `tralics`. The previous example is understood correctly by Tralics, meaning that Tralics groks catcodes, active characters, delimited arguments, etc. More complicated is

```
{\catcode`\_\active\global\let_X}
\begingroup\lccode `\~=`\_
\catcode`\~\active
\lowercase\relax{~\def~{u}\endgroup ~a}~
\MakeUppercase{abAB\ae\i}
```

This shows that `\lowercase` is fully implemented, including funny details, like the optional `\relax` before the open brace. One objective was to fully implement the TeX macro expansion mechanism, including conditionals (and to translate most LaTeX constructs). The following example provokes an error:

```
\font\foo=cmr10 scaled 1023
\setbox0=\hbox to 10pt{\foo p\hss}
\ifdim\dp0>0pt 1\else 2\fi
```

Here, the first line defines a command `\foo` that selects some font; since we do not want to parse font metric files, all these fonts are equivalent and `\foo` is a no-op. The second line constructs a horizontal box and puts its content into `\box0`; we shall explain later what Tralics does with the box. One thing is clear: the objective of Tralics

is not to split a paragraph into lines, so there is no packaging, no overfull hboxes, no underfull vboxes. In fact, the box specifications and the \hss command are ignored, the resulting box contains only the letter p. If you ask for the depth of the box, Tralics signals an error (and returns 0).

Note that \ifvmode and \ifinner are part of TeX's macro expansion mechanism, but these commands are not implemented, mostly because there is little relationship between the modes of TeX and the modes of Tralics. All you need to know is that the following lines work as expected:

```
\ensuremath{\Omega}
\leavevmode
\mbox{\par}
```

The last line is silly: inside a box, \par cannot start a new paragraph, and the resulting box is empty. For simplicity, the \par token is allowed anywhere (as if every command were prefixed with \long). Why \mbox is declared \long in LaTeX is beyond me.

The target language

The main job of a translator like Tralics is to read and analyze the input document, manage it (and simplify, complete, re-order, etc.) and produce a new document. The big question here is: what elements should be chosen? In the case of latex2html, the choice is limited to HTML version 2.0, 3.0, 3.2, and 4.0. In the case of Tralics, there is no predefined set of elements. In the opening example, we used <U> for the \personalURL command, purely to avoid long lines; usually, more suggestive names should be used.

The default behavior of Tralics is defined by its main application, namely the raweb, with the following constraints: the resulting XML has to be converted to a set of HTML pages, one page for each module; it has to be converted to PostScript or pdf, one document per research team, without loss in typographic quality; and finally, mathematics should not be forgotten. Two style sheets were used, one to produce HTML, and another for pdf. Formatting a document is not trivial, because some objects have to be evaluated more than once, sometimes depending on the context (this explains why some LaTeX commands are fragile). In our case, the formatting was done by TeX or pdfTeX, which was able to parse XML files thanks to the xmltex package, see [1]. There is still work to do, in particular concerning tables: the implementation of tables in fotex.sty is rather disturbing; 400 lines of TeX code were needed (for instance to patch the behavior of <mfenced> and <mover>, and to get the cover page right).

For the DTD, we first considered the Docbook DTD, [10], and discarded it, because it was too complicated. We then considered the TEI DTD, and simplified it drastically. Lots of interesting material can be found on that web site (http://www.tei-c.org/), including PassiveTeX [8], a companion tool to xmltex, and a model for the above-mentioned style sheets.

On the other hand, we added some features, in order to implement figures, tables, math, etc. For obvious reasons, we use the presentation elements of MathML [2]. The design of the bibliography is not yet done (more on this subject will follow).

There are nearly one hundred elements in this DTD, 30 elements being specific to the raweb, 20 for the bibliography; other elements are borrowed from TEI. Via the configuration file, you can change the default names (except those defined by MathML, and those specific to the raweb); using commands like \xmlelement, you can generate additional elements. If you do so, you can convince Tralics to use your own DTD.

The Tralics software, the documentation, and the raweb DTD can be found at:

http://www-sop.inria.fr/apics/tralics/

Application to the raweb

The main application of Tralics can be found on Inria's web site:

http://www.inria.fr/rapportsactivite

For year 2002, 125 teams have written their activity report (in LaTeX, with one exception) and the result is over 3000 pages of pdf (A4 format, 10 point font size), resulting in 3890 HTML pages, that occupy 598.386 MB of a CD-ROM. Translating these files took 100 seconds (30 seconds if Tralics is compiled with the -O2 switch, this shows that the C++ optimizer does a good job). The most time consuming operation was the conversion from XML to pdf (the xmltex parser is very slow), and production of images (conversion from PostScript to pdf via the epstopdf Perl script; and to png, using tools borrowed from latex2html).

Lots of people were involved in the process: first, the raweb team, which contributed to the design of the raweb (the web site, the paper version, the idea of using XML, the DTD), the authors of the texts, the people who collected the texts and who corrected some typos, and Marie-Pierre Durollet who deserves special thanks: she wrote some shell scripts, Perl scripts, cgi scripts, part of the style sheets, etc., in fact, all that is needed to make the web site function. She was also the first real user of Tralics, in that she translated all the files, on her Linux box (other people tried Tralics on SunOS, MacOS, Windows).

One non-trivial point is the question of mathematics: there are some browsers (Amaya, Mozilla, etc.), that understand MathML, or claim to do so, together with

plugins and other tools supposed to visualize MathML, but we decided nevertheless to convert all the math. In fact, Tralics comes with a Perl script (a bit over one thousand lines), that reads an XML file, and converts math and images.

Image post-processing Consider a simple, classical, example:

```
\begin{figure}
  \begin{center}
    \includegraphics{miaou_transf.ps}
  \end{center}
  \caption{Modèle de transducteur.}
  \label{trans}
\end{figure}
```

It is translated into

```
<figure file='miaou_transf' id='uid30'>
<head>Modèle de transducteur.</head>
</figure>
```

As can be seen, the argument of the `\label` command was normalized from 'trans' to 'uid30' and became an attribute of the figure element, the centering environment was ignored, the underscore treated as a normal character, the extension of the file name was removed, and the file became an attribute of the figure element (see below an example where more than one image appears in a figure environment). The post-processing script has the job of making sure that the image exists in PostScript, pdf and png formats. It also modifies the XML:

```
<figure aux='image_1.png'
  file='miaou_transf' id='uid30'>
<head>Modèle de transducteur.</head>
</figure>
```

and the HTML result will be:

```
<div align="center">
  <a name="uid30"></a>
  <table>
    <caption align="bottom">
      <strong>Figure 1. </strong>
      Mod&#xe8;le de transducteur.
    </caption>
    <tr><td><img alt="miaou_transf"
           src="image_1.png"></td></tr>
  </table>
</div>
```

A few remarks are needed: the initial LaTeX environment is a floating one, so that the result is outside a `<p>` element, and using `<div>` for grouping is justified. The figure and its caption are centered. Note that the image is a cell of a table, because this is the easiest way to associate a caption to the table.

The è character was transformed into `è` by an ad hoc program, because, for some unknown reasons, the style sheet refused to use latin1 encoding, forcing UTF-8 instead, an encoding not understood by the program that indexes Inria's web site.

The figure was the first one in the file, so that the style sheet numbered it as figure one (in the next example, there is no `\caption`, hence the figure has an empty `<caption>`, but nevertheless a unique number). The HTML translation of `\ref{trans}` is something like `<a...>1</a>`. The `<a>` element has a href attribute whose value depends on the name of the label, and the current web page, which was renamed from 'resonn', the name given by the author, to 'module7', which has no meaning (searching for 'module50' on Inria's web site reveals that three teams have written at least 50 modules). The important point however is the content of the `<a>` element: the value 1 is not computed by Tralics, but by the style sheet.

Another example is the following.

```
\begin{figure}[htbp]
\begin{center}
\begin{tabular}{ccc}
   \includegraphics[width=3.5cm]{imgl}&
   \includegraphics[width=3.5cm]{imgc}&
   \includegraphics[width=3.5cm]{imgr}\\
cap l&cap c&cap r
\end{tabular}
\end{center}
\end{figure}
```

The same effect could be achieved with the subfigure environment, but we wanted to give an example of a table. Note that, if more than one image is to be put in a figure, we suggest using a table, since there are still unresolved problems regarding spacing. The translation of the previous example is the following

```
<figure rend='array' id='uid9'><p>
  <table rend='inline'> <row>
     <cell><figure file='imgl'/></cell>
     <cell><figure file='imgc'/></cell>
     <cell><figure file='imgr'/></cell>
    </row><row>
      <cell>cap l</cell>
      <cell>cap c</cell>
      <cell>cap r</cell>
  </row></table></p>
</figure>
```

In order to reduce the size, we did omit the halign = center attributes for the six `<cell>` elements, and the width=3.5cm, rend=inline attributes of the three non-toplevel `<figure>` elements. The HTML translation is a table in a table, the inner table has no caption, the outer table has an empty caption. One of these two tables could be removed.

Math post-processing Note that a table or a figure is a float (hence is numbered) if and only if its rend attribute is not inline; on the other hand, math formulas never float, but are numbered only if they are non-inline, and have a label. In what follows, we shall distinguish between the math (defined by MathML) and the formula (something we borrowed from the TEI DTD). It is the formula that has a label (hence a number), and this explains why we have some troubles translating environments like `align` and commands like `\tag`. Nevertheless, the translation of the simple formula $K \neq \Gamma$ is

```
<formula type='inline'><math><mrow>
<mi>K</mi><mo>&ne;</mo><mi>&Gamma;</mi>
</mrow></math></formula>
```

For completeness, we show here the XSL-FO output that is used to convert the formula into pdf (complete code for the example above, skeleton for a display-math formula, numbered and unnumbered).

```
<m:math overflow="scroll"><m:mrow>
  <m:mi>K</m:mi>
  <m:mo>&#x2260;</m:mo>
  <m:mi>&#x393;</m:mi>
</m:mrow></m:math>

<fotex:displaymath>
  display-math stuff
</fotex:displaymath>

<fo:inline id="uid11"><fotex:equation>
  numbered math stuff
</fotex:equation></fo:inline>
```

In this case, the Perl script that handles images and math formulas classifies the formula as a simple one, formed of three tokens in a row, a Latin letter, a symbol and a Greek letter. It replaces the formula by the following XML code.

```
<span class='math'><hi rend='it'>K</hi>
  <img src='img_other_ne.png' alt='$\ne$'
   width='14' height='26' align='middle'/>
 <img src='img_Gamma.png' alt='$\Gamma$'
   width='12' height='13' align='bottom'/>
</span>
```

Images for Greek letters and other symbols were precomputed (by `latex2html`). The dimensions given here are nearly twice the size of a ten-point $\neq$ and Γ, except that the depth of the $\neq$ should be 4, and there is no way to indicate a depth in HTML. That's the reason why we indicate a total height plus depth of 26, together with an 'align=middle' attribute. The resulting HTML is ugly (too much blank space with the line and the following one), and uses a deprecated feature, but we do not know how to do better. The resulting HTML is

```
<SPAN class="math">
<SPAN class="it">K</SPAN>
<IMG width="14" height="26" align="middle"
border="0" alt="$\ne$"
src="../images/img_other_ne.png">
<IMG width="12" height="13" align="bottom"
border="0" alt="$\Gamma$"
src="../images/img_Gamma.png"></SPAN>
```

Let's give another example: $L^2 \rightarrow L^\infty$ is translated into MathML as

```
<math><mrow>
  <msup><mi>L</mi><mn>2</mn> </msup>
  <mo>&rightarrow;</mo>
  <msup><mi>L</mi><mi>&infin;</mi></msup>
</mrow></math>
```

The result here is a single image:

```
<span class='math'>
 <img align = 'bottom'
  width ='71' height ='13'
  src='math_image_1.png' border='0'
 alt='Im1 ${L^2\rightarrow L^\infin }$'/>
</span>
```

A formula of the form a^b or a_b is considered simple (and translates to `<sup>` or `<sub>`) in the case where b is an HTML character. Here, one exponent is infinity, hence an image is needed. Note how the alt field is constructed: we try to reconstruct the TeX formula from the MathML element. The first token, the `Im1`, indicates the image number, and is only useful for debugging. Constructing the image is not so trivial. First, a file is created that contains lines of the form

```
<formula id="1"><math><mrow>
  <msup><mi>L</mi> <mn>2</mn></msup>
  <mo>&#8594;</mo>
  <msup><mi>L</mi><mi>&#8734;</mi></msup>
</mrow></math></formula>
```

In fact, the file contains all formulas that need to be converted, with a unique id identifying the image. The xmllint processor is used to replace entity names by their Unicode values. The resulting file is processed by LaTeX, in the same fashion as the main document, in order to get a DVI file. The interpretation of the `<formula>` element (which is absent from the XSL-FO file) uses code borrowed from `latex2html`, which has two side effects. First, the log file contains a line like

```
l2hSize :1:8.14003pt::0.0pt::48.73616pt.
```

and second, each page of the DVI file contains a single math formula in its upper left corner. For each such page a PostScript file is generated by `dvips` and then converted to png via the `pstoimg` utilities, which use the size information shown above. After a magnification factor of 40%, this gives a resulting image of 13 by 71 pixels, and this information is pushed back in the XML file.

The structure of a LaTeX document

Tralics assumes that the document to be translated conforms to LaTeX standards: that there is a line with `\documentclass`, followed by some lines containing `\usepackage` commands, followed by a `document` environment. This is a very special environment, because its content is at brace level zero (as in standard LaTeX), and the tokens read by `\AtBeginDocument` and `\AtEndDocument`
are inserted at the right place, for instance

```
\documentclass{article}
\usepackage{calc}
\AtBeginDocument{start}
\AtEndDocument{end}
\newlength{\foo}
\AtEndDocument{\AtEndDocument{ realend}}
\begin{document}
\setlength{\foo}{1cm+3pt}
\the\foo
\end{document}
This line is not translated
```

would produce essentially the following

```
<p>start
31.45274pt
end realend</p>
```

There is a special hack here: a special `\endinput` token is inserted, whose effect is to stop translation of the current file (after the tokens remembered by the document hook), but the job of the translator continues, because it is at this moment that the bibliography is translated: Tralics has the list of all `\cite` commands, so it can do the equivalent of BibTeX. The resulting bibliography is inserted where the `\bibliography` command is located. It is an error if a `\cite` command is seen after the end of the document (i.e. comes from a BibTeX file).

Parameterization The first job of Tralics is to read the source file and find a `\documentclass` command, in order to apply document specific rules found in the `.tralics_rc` file (the configuration file, it can be in the current or home directory). In the case where the configuration file contains

```
BeginType article
  on package loaded calc CALC = "true"
  on package loaded foo/bar FOO = "true"
End
```

then, whenever the document class is `article` (trailing digits are ignored in the name) the two lines are executed. It follows that the root element of the resulting XML document will have the attribute CALC set to true, in case the `calc` package is loaded, and FOO is set to true in case the `foo` package is loaded with the bar option. By default, the attribute language is set (to french or english) when Tralics is able to determine the main language, either because (as in this document), english is an option to `\documentclass`, or because the `babel` package is loaded with recognized parameters. Finally, the meaning of `\setlength` depends on whether the `calc` package is loaded or not.

Sectioning commands The XML model of Tralics is based on the notion of paragraph (`\par` command, `<p>` element). This element can contain inline stuff (text, images, tables, math formulas); it is at the same level as non-inline stuff (images, tables, math, notes, bibliographic entries), and can be contained in a sectioning command (for instance the paragraph is in a subsection in a section in a chapter in a part). These elements have in general a number (computed by an external program rather than defined by the user, never computed by Tralics), and can be referenced. The translation of a sectioning command is a `<div`i`>` element, where i is an integer between 0 and 6. Example:

```
\section{x}A\label{a}
\subsection{y}
B\label{b}C\label{c}D
\paragraph{z}
\ref{a}\ref{b}
\subsection{t}
```

gives

```
<div0 id='uid1'><head>x</head>
 <p>A</p>
 <div1 id='uid2'><head>y</head>
  <p>BCD</p>
  <div3 id='uid3'><head>z</head>
    <p><ref target='uid1'/>
       <ref target='uid2'/></p>
  </div3>
 </div1>
 <div1 id='uid4'><head>t</head>...
```

By default, `\section` is the top-level division, but chapters are allowed in a report, and parts in a book. The '...' is not part of the output, it just indicates the current position in the XML tree. Note that Tralics is in outer vertical mode, said otherwise, the occurrence of a character will imply the creation of a `<p>` element; in LaTeX, the current mode would depend on the document class. This is one reason why `\ifvmode` is not implemented. The example shows a logical error: there is a `<div3>` element in a `<div1>` element. It would be numbered 1.1.0.1 in LaTeX, and 1.1.1 (without the zeros) using an XSLT processor.

Cross references Tralics implements `\label` and `\ref` but not `\pageref`. The basic idea is to put a mark in the XML document, and use references to this mark. The

mark is an attribute of type ID, and there are some limitations in XML that do not exist in LaTeX (for instance no element type may have more than one ID attribute specified, and an ID cannot start with a digit). In order to remove these difficulties, Tralics uses its own list of IDs (uid1, uid2, etc.). Associated with the mark is a value, produced by \p@foo\thefoo. This mechanism is not implemented in Tralics: the value associated to a label is not in the XML document but must be computed by the application (the style sheet for instance). In order to distinguish between Figure 4 or Table 5, the label must be associated with a figure or a table (a footnote, an item in a list, a formula, a division, that's all). Hence the following changes with LaTeX: inside a figure or table environment, there must be at most one label, and it can be before or after the caption. A math formula can have a label only if it is a display math equation, and it is numbered only if it has a label. See example above. There is no need to call Tralics twice or more: a single pass is enough.

References to external documents are understood. In the following example, we switch to French in order to show the behavior of special characters (like colon and underscore) with or without the \url command:

```
\language=1
\href{http://foo\_ba}{http://foo_bar}
\href{\url{http://foo_ba}}{http://foo_bar}
```

The translation is

```
<xref url='http://foo_bar'>
  http ://foo_ba</xref>
<xref url='http://foo_bar'>http://foo_ba
</xref>
```

Basic formatting tools

The following example is from page 218 of *The TeXbook* [6], with an addition to verify that \trialdivision is really called 132 times.

```
\tracingall
\countdef\td 4 \td=0
\newif\ifprime \newif\ifunknown
\newcount\n \newcount\p \newcount\d
\newcount\a
\def\primes#1{2,~3% assume that #1 >= 3
  \n=#1 \advance\n by-2 % n more to go
  \p=5 % odd primes starting with p
  \loop\ifnum\n>0 \printifprime
      \advance\p by2 \repeat}
\def\printp{, %
  \ifnum\n=1 and~\fi %
  \number\p \advance\n by -1 }
\def\printifprime{\testprimality
  \ifprime\printp\fi}
\def\testprimality{{\d=3 \global\primetrue
  \loop\trialdivision
     \ifunknown\advance\d by2 \repeat}}
\def\trialdivision{\a=\p
  \global\advance\td by 1
  \divide\a by\d
  \ifnum\a>\d \unknowntrue\else\unknownfalse\fi
  \multiply\a by\d
  \ifnum\a=\p \global\primefalse
     \unknownfalse\fi}

The first thirty prime numbers are \primes{30}.
trialdivision macro was expanded \the\td\ times
```

This example works in LaTeX and in Tralics. All TeX primitives that start with 'tracing' are implemented, although most of them refer to behavior that is not implemented in Tralics. If you run Tralics on the previous example, the log file will contain:

```
[706] \countdef\td 4 \td=0
{\countdef}
+scanint for \countdef->4
{\td}
+scanint for \td->0
[709] \newcount\a
{\countdef \a=1550}
```

You can see the input source line whenever it is read, with its line number, and the commands that are evaluated. On page 269 of *The TeXbook*, you can see the definition of an integer. This is so complicated that Tralics prints the value whenever scanned. The last line shows that the \a command will access the internal tables at position 1550.

```
\iterate->\ifnum \n >0 \printifprime
   \advance  \p by2 \relax \expandafter
   \iterate \fi
+\ifnum
+scanint for \ifnum->28
+scanint for \ifnum->0
+iftest true
```

Here we can see that the \loop macro is implemented as in LaTeX and not as in plain TeX. The \iterate token is a private one, the \loop macro does not kill your command (but the loop inside the loop will modify it, and this explains why extra braces are needed).

```
{begin group character}
+stack: level + 2
```

The first line indicates that Tralics has seen an open brace (technically, a character of catcode one), and that it creates a new frame on the save stack (the outer level is numbered 1, as in TeX).

```
{end group character}
{Text:, 5}
+stack: restoring \ifunknown
+stack: restoring integer value 1550 0
+stack: restoring \iterate
+stack: restoring integer value 1549 0
+stack: level - 2
```

And this is done when Tralics sees the closing brace. It restores two commands and the two registers \a and \d (the association between \d and the number 1549 can be found in the log file). A side effect of seeing the closing brace is to flush the XML buffer (what follows the 'Text:' on the second line), for the case where the current font might change.

List structures Standard LaTeX lists are implemented, but they are not customizable. The only non trivial part is that the optional argument of the \item command should be evaluated in a group. As an example:

```
\begin{itemize}
 \item a
 \item [\it b] c
 \begin{enumerate}
   \item d
   \begin{description}
     \item e
   \end{description}
 \end{enumerate}
\end{itemize}
```

This will translate to

```
<list type='simple'>
 <item id='uid14'><p>a</p></item>
 <label><hi rend='it'>b</hi></label>
 <item id='uid15'>
  <p>c</p>
  <list type='ordered'>
   <item id='uid16'><p>d</p>
    <list type='description'>
     <item id='uid17'><p>e</p>
     </item>
    </list>
   </item>
  </list>
 </item>
</list>
```

Here, in order to make the XML output more readable, we added and removed some space and newline tokens. The general rule for Tralics is to output one space (or newline character), whenever TeX would output one space character. In particular, the TeX scanner converts two consecutive newline characters in a space character and a \par token. This space character will be output by Tralics as a newline character.

Footnotes The translation of

```
\footnote{a}
\footnote{Y \AddAttToLast{x}{y}b\par
  \AddAttToCurrent*{place}{here}c}
```

is

```
<note id='uid14' place='foot'>a</note>
<note place='here' id='uid15'>
 <p x='y'>Y b</p>
 <p>c</p>
</note>
```

Two remarks: each note has a uid, hence can be referenced, but there is nothing special about it (no counter, no mark, no restrictions).

On the other hand, the example shows how to add an attribute to an XML element, either the element created latest (here the <p> element that contains the text of the footnote), or the current element (the footnote, since the translation of \par does not create a new <p> element, it is the translation of the character c that forces a second <p> in the note). In the unlikely event that the element already has an attribute of the same name (for instance, a footnote has a default place attribute), the command is ignored — unless you use its starred form to overwrite.

New elements It is easy to use new elements, just say

```
\begin{xmlelement}{main-elt}
\begin{xmlelement}{sub-elt1}
text1
\end{xmlelement}
\begin{xmlelement}{sub-elt2}
text2
\end{xmlelement}
\AddAttToLast{sb2-att}{value1}
\AddAttToCurrent{foo-att}{att-value''}
\end{xmlelement}
```

and you will get

```
<main-elt foo-att='att-value&amp;apos;&amp;apos;'>
<sub-elt1>
text1
</sub-elt1>
<sub-elt2 sb2-att='value1'>
text2
</sub-elt2>
</main-elt>
```

You can also try

```
\hbox{a\it b}
\vbox{c\it d}
\newcommand\AGtest{AG}
\setbox0=\xbox{myelt}
  {\aftergroup\AGtest e\it f}
\copy0 \copy0
```

This will work in Tralics, since either an \hbox or a \vbox is just an unnamed \xbox. The braces serve for grouping, and as argument delimiters. The result is

```
a<hi rend='it'>b</hi>
c<hi rend='it'>d</hi>
AG
<myelt>e<hi rend='it'>f</hi></myelt>
```

```
<myelt>e<hi rend='it'>f</hi></myelt></p>
```

Note: when Tralics constructs an element, the equivalent of a box in TeX, it is in a special mode that does not match any of TeX's modes.

Verbatim material A document like this one uses lots of verbatim material. Tralics is familiar with standard verbatim environments, and some extensions. For instance, the translation of

```
\DefineShortVerb{\|}
you can say |\foo| or \verb*+foo bar+.
\begin{Verbatim}
verb line1
$<&\>
\end{Verbatim}
```

would be

```
you can say <hi rend='tt'>\foo</hi>
or <hi rend='tt'>foo&blank;bar</hi>.</p>
<p noindent='true'><hi rend='tt'>
   verb line1</hi></p>
<p noindent='true'><hi rend='tt'>
   $&lt;&amp;\&gt;</hi></p>
<p noindent='true'>
```

Note here that the `\end` commands check whether the environment is followed by an empty line. If it is not, a `\noindent` token is inserted.

The layout of the page

The essential idea of Tralics is that the result of the translation is independent of the layout of the page.

But if you give `0.8\textwidth` as the width of a figure, Tralics will replace this by `12cm`, because something has to be done. Optional arguments of sectioning commands are ignored: in general one would use them for marking the document. We have implemented the TeX markup commands, but they do nothing; we think of extending the functionalities of the title-page mechanism of Tralics. In fact, if the title-page specifies something like (see opening example):

```
\address p<address> "no address"
```

then there is a command `\address` that takes one argument, whose translation is put in an `<address>` element, and recorded. The 'p' marker says that paragraphs are allowed, an 'E' marker indicates that the command is an environment, a '+' that the command can be issued more than once, etc. As you can see, there is still work to be done.

Tabular material

There are essentially four points to be considered: tables in math mode (described later), standard LaTeX tabular (see earlier example), the tabbing environment (not implemented, because it implies implementing all typesetting algorithms), and the `\halign` primitive, which is much too complicated to implement.

One problem is how to translate a table specification such as `r||l`. Tralics understands that there are two columns, right and left justified, with a rule on the right and the left, but this is impossible to translate into HTML, that knows only `|r|l|` or `rl`. The `\hline` and `\cline` are implemented, but suffer from the same limitations.

Mastering floats

There is in general enough flexibility for adding or removing one line on the current page, so that, for instance, TeX is not faced with the dilemma of either putting a section title at the bottom of a page, or generating a horribly underfull page. The situation is quite different for tables or images, which can be much larger; this is the reason why they can float (i.e. items are put in the output in a different order). Handling of floats is non-trivial; in some cases, the best thing to do is to resize (the image, or even the text).

The philosophy of Tralics is the following: in the case where the XML is translated into HTML, no object has to float. On the other hand, since the author of the document is not the person that does the final typesetting, no fine tuning of floats can be achieved. As a result, Tralics just ignores optional arguments of float environments, float parameters, and things like `wrapfig` environments.

Font selection

The introductory example shows some features of Tralics. For instance, the default input encoding of the document is `latin1`, and this is the same as the output encoding: the translation of `é\oe` will be `éœ`, you can also say `\'e\.E` if you want éĖ; all Unicode characters with code less than `x180` are recognized. If you need other characters, you must use a construct like

```
Hàn Th\xmllatex{&\#x1ebf;}{\'{\^e}} Thành
```

The command `\xmllatex` takes two arguments, the first one is ignored by standard LaTeX, and the second one by Tralics. Note how the sharp sign is protected. See the Unicode manual [9] for the numeric value `x1ebf`. There is no way to construct a logo, like the TeX logo, with `\kern` or `\lower`, but instead, you could say

```
\newcommand\MyLogo{\xmlemptyelt{MyLogo}}
```

and define a `<MyLogo/>` element in your DTD. You could also use a parameterized version of the logo, like this

```
\newcommand\ParLogo[1]{\xmlelt{PLogo}{#1}}
\MyLogo
\ParLogo{2$\varepsilon$}
```

The resulting XML document will contain

```
<MyLogo/>
<PLogo>2<formula type='inline'><math>
<mi>&epsiv;</mi></math></formula></PLogo>
```

As explained above, Tralics knows of the `\font` command, but this command should not be used. It is much better to say something like

```
{\bfseries a\itshape b\small c\ttfamily d}
```

which translates into:

```
<hi rend='bold'>a</hi>
<hi rend='it'><hi rend='bold'>b</hi></hi>
<hi rend='small'><hi rend='it'>
  <hi rend='bold'>c</hi></hi></hi>
<hi rend='small'><hi rend='it'>
  <hi rend='tt'><hi rend='bold'>d</hi>
</hi></hi></hi>
```

As this example shows, a font is defined by a size, shape, series and family, nothing more (is it possible to convert to HTML a document written in Computer Modern Funny Roman using the U encoding?). Tralics understands all ten standard font sizes, but uses only three (a normal size, a larger one, and a smaller one); there are some bugs in the implementation. In an example like this

```
{\it a\par b}
```

a new paragraph is created when the b character is sensed, and a font element must be inserted in this paragraph, since the current font is not the default one. The commands `\it`, `\textit`, `\itshape` have the same meaning as in LaTeX. The `\em` command also (but Tralics does not know if the current font is slanted or not, and may generate the wrong result).

Higher mathematics

TeX has a reputation of producing very high quality mathematics, and people at the AMS have worked hard to make it easy to use. As a result, trying to translate the entire AMS functionality to MathML is nearly impossible (too many features have no equivalent). Even converting the MathML into pdf was a challenge (there are still unresolved problem, like tables in tables, missing characters, etc.), and the rendering by tools like Amaya is a bit strange.

The translation of $\int_0^\infty \mathcal{F}(x)^2\,\mathrm{d}x$ is unsatisfactory to us. In particular, Tralics fails to notice that the exponent applies to the '$\mathcal{F}(x)$', and we wonder whether an expert system should be used (consider the scope of the `<mrow>` element in the translation below).

```
<math>
 <mrow>
  <msubsup><mo>&int;</mo>
    <mn>0</mn> <mi>&infin;</mi>
  </msubsup>
  <mrow><mi>&Fscr;</mi><mo>(</mo>
   <mi>x</mi> </mrow>
  <msup><mo>)</mo><mn>2</mn> </msup>
  <mspace width='3pt'/><mi> d </mi>
  <mi>x</mi>
</mrow></math>
```

LaTeX in a multilingual environment

Inria's Rapport d'Activité is, by nature and law, a French document. The RA, or the raweb, being just its scientific annex, can be in English (with a French summary). It was decided that starting in year 2003, the whole thing (the source document, the web site, the logo, the these-pages-in-French-only pointers) will be in English. Thus, special attention was given to this problem in Tralics. For instance, whether the title of the bibliography should be "References" or "Références" does not depend on Tralics, but on the style sheet (or the DTD). On the other hand, one may imagine a document (like this one) that has an abstract in two languages, and that the first abstract should be in the main language. Using the title page mechanism of Tralics will give you a fixed order for the XML result, but as the introductory example shows, the main language is an attribute of the root element (and it is the `\maketitle` command that selects the current language as main language).

In fact, TeX provides a `\setlanguage` command, which is ignored by Tralics, and a `\language` command that selects a language: Tralics assumes that English has number zero, French has number one, and your favorite language has number two (in fact, only two languages are really supported). The BibTeX translator (see below) uses the value of the current language for the interpretation of strings like 'jan'. The following example shows the differences between French and English:

```
\language =1
<<guill'' ponctuation;
\verb+<<guill'' ponctuation;+
a\xspace b\xspace !
\language=0
<<guill'' ponctuation;
a\xspace b\xspace !
```

The translation is

```
« guill » ponctuation ;
<hi rend='tt'>&lt;&lt;guill
'&ZeroWidthSpace;'&ZeroWidthSpace; 
ponctuation;</hi>
a b !
« guill'' ponctuation;
a b!
```

The important points are the following: a space is added before some punctuation characters (colon, semi-colon, guillemets, etc.), even when the `xspace` package (which

is language independent) thinks it useless. Everything that looks like opening or closing double quotes is converted to French guillemets, with the proper spacing. Note the translation of the \verb command: the two single quotes are followed by a special (invisible) character, the purpose of which is to avoid ligatures in the case where the XML is processed by TeX; the line breaks were manually added in order to avoid overfull lines.

Portable graphics in LaTeX

We tried to implement some commands, for instance

```
\setlength{\unitlength}{.0075\textwidth}
\begin{picture}(90,50)
\put(40,25){\framebox(10,10){$H(s)$}}
\put(19,30){\vector(1,0){9}}
\put(60,15){\vector(-1,0){10}}
\put(17.75,0.5){\oval(1.5,1.5)[r]}
\end{picture}
```

The result is given below. However, for the RA, we copied the content of the picture environment in a file, called LaTeX on it, and replaced the picture by a reference to the PostScript file.

```
<picture width='90.0pt' height='50.0pt'>
<put xpos='40.0pt' ypos='25.0pt'>
 <box width='10.0pt' height='10.0pt'>
 <formula type='inline'><math>
  <mrow><mi>H</mi><mo>(</mo><mi>s</mi>
  <mo>)</mo></mrow></math>
 </formula></box></put>
<put xpos='19.0pt' ypos='30.0pt'>
 <vector xdir='1.0pt' ydir='0.0pt'
  width='9.0pt'/></put>
<put xpos='60.0pt' ypos='15.0pt'>
  <vector xdir='-1.0pt' ydir='0.0pt
   width='10.0pt'/></put>
<put xpos='17.75pt' ypos='0.5pt'>
   <oval xpos='1.5pt' ypos='1.5pt'
  specs='r'/></put>
</picture>
```

Using PostScript

Since our main application uses pdf in preference to PostScript, we will not speak about PostScript here (some people use psfrags in order to replace PostScript fonts in their figures by standard LaTeX ones; this is strange).

Index generation

Tralics offers no specific tool.

Bibliography generation

The design of the bibliography of the raweb is still a subject of research. We were faced with the following problems: first, the raweb uses three different BibTeX files, and the translation of one of these files depends on whether it was put on the web or printed on paper, and the items in the main bibliography file are grouped in different categories, depending on their type; a total of four bst files are used. One idea was to parse the bbl files (but, if you look at the sources of the footbib package, you can see how hard it is; by the way, bibliographic entries are no longer footnotes). The second idea was to modify the bst file in order to generate an environment per item, in order to simplify the parsing. The third idea was to abandon BibTeX; since there is still no XML standard for bibliographies, and no universal tools, we just wrote a BibTeX to LaTeX translator. For instance, the TeXbook entry looks like this

```
\citation{Knu84}{cite:texbook}{year}{book}
\bauteurs{\bpers{D. E.}{}{Knuth}{}}
\cititem{btitle}{The \TeX book}
\cititem{bpublisher}{Addison Wesley}
\cititem{byear}{1984}
\endcitation
```

This part of the translator reads the BibTeX file, expands the BibTeX macros (predefined, or used defined), removes useless stuff, sorts the entries, and returns the LaTeX equivalent (in fact, the stuff is written in a bbl file, but only for you to see what happens in case of error). The same entry processed by BibTeX:

```
\bibitem[\protect\citeauthoryear{Knuth}
{Knuth}{1984}]{texbook}
Knuth, D.~E.
\newblock {\em The \TeX book}.
\newblock Addison Wesley, 1984
\UseExtraLabel{}.
```

A comparison shows that BibTeX adds some tokens that would be very hard to remove: tilda, comma, period, together with some others (\em, \newblock) that can be context sensitive. On the other hand, the LaTeX to XML translator generates

```
<citation from='year' key='Knu84'
   id='cite:texbook' type='book'>
 <bauteurs>
   <bpers prenom='D. E.' nom='Knuth'/>
 </bauteurs>
 <btitle>The <TeX/>book</btitle>
 <bpublisher>Addison Wesley</bpublisher>
 <byear>1984</byear>
</citation>
```

so that it is up to the style sheet to do all the real work (using \em for the title if it's a book, use the plural of 'editor' if there is more than one, etc.). Each style has its own peculiarities: the tugboat style generates a special optional argument for \bibitem, but the raweb tools

need the `from` and `type` attributes for sorting (problem here: citations are already sorted).

Conclusion

We have shown in this paper that a LaTeX to XML translator can be very useful, when you have the tools to manage the resulting XML. We hope that standardization will continue, so that it will become easier to write such tools, and adapt them. Putting high quality mathematics on the web is still a challenge, and we hope that others will appreciate our contribution.

References

[1] David Carlisle, Michel Goossens, and Sebastian Rahtz. De XML à PDF avec `xmltex` et PassiveTeX. In *Cahiers Gutenberg*, number 35-36, pages 79–114, 2000.

[2] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier. Mathematical Markup Language (MathML) version 2.0. `http://www.w3.org/TR/MathML2/`, 2001.

[3] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison Wesley, 1993.

[4] José Grimm. Outils pour la manipulation du rapport d'activité. Technical Report RT-0265, Inria, 2002. `http://www.inria.fr/rrrt/rt-0265.html`.

[5] Yannis Haralambous and John Plaice. Produire du MathML et autres ...ML à partir d'Ω : Ω se généralise. In *Cahiers Gutenberg*, number 33-34, pages 173–182, 1999.

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

[7] Philippe Louarn. Une expérience d'utilisation de LaTeX : le Rapport d'activité de l'INRIA. *Cahiers Gutenberg*, (0):17–24, apr 1988.

[8] Sebastian Rahtz. Passive TeX. `http://www.tei-c.org.uk/Software/passivetex/`, 2003.

[9] The Unicode Consortium. *The Unicode Standard, version* 3.0. Addison Wesley, 2000.

[10] Norman Walsh and Leonard Muellner. *Docbook: The Definitive Guide*. O'Reilly & Associates, Inc, 1999.

Docbook In ConTEXt, a ConTEXt XML Mapping for DocBook Documents

Simon Pepping
Elsevier, Amsterdam, Netherlands
s.pepping@elsevier.com, spepping@leverkruid.nl
http://www.leverkruid.nl/

Abstract

Docbook In ConTEXt is a ConTEXt module that allows one to produce a typeset version of a Docbook XML file, in dvi or pdf format. It takes a Docbook XML file as input for TEX. ConTEXt's built-in XML parser parses the file and maps opening and closing tags to the ConTEXt commands specified in the *Docbook In ConTEXt* module.

The first part of this article describes how one can run ConTEXt on a Docbook XML file using the *Docbook In ConTEXt* module. The second part deals with some aspects of programming this module. It presents the general framework, discusses some of the problems encountered, and highlights the programming of some noteworthy elements.

Résumé

Docbook In ConTEXt est un module ConTEXt qui permet la production d'une version composée d'un fichier XML Docbook, aux formats DVI ou PDF. Il prend un fichier XML Docbook en entrée de TEX. Le parseur XML interne de ConTEXt analyse le fichier et associe les balises ouvrantes et fermantes aux commandes ConTEXt spécifiées dans le module *Docbook In ConTEXt*.

La première partie de cet article décrit comment on peut lancer ConTEXt sur un fichier XML Docbook en utilisant le module *Docbook In ConTEXt*. La deuxième partie traite de certains aspects de programmation de ce module. Nous y présentons le cadre de travail général, nous discutons certains problèmes que nous avons rencontrés, et nous soulignons la programmation de certains éléments dignes de considération.

What is Docbook In ConTEXt?

Docbook In ConTEXt combines two technologies that are widely used by authors of technical literature: the Docbook DTD and the ConTEXt macro package for TEX.

It is a ConTEXt module that allows one to produce a typeset version of a Docbook XML file, in dvi or pdf format.

It takes a Docbook XML file as input for TEX. ConTEXt's built-in XML parser parses the file and applies ConTEXt commands when it reads opening and closing tags. Which ConTEXt commands are applied, and therefore how the output is formatted, is determined by the Docbook In ConTEXt module.

XML, Docbook and stylesheets Docbook documents are XML articles. They contain XML tags, such as `<title>` and the corresponding end tag `</title>`. These two tags mark the enclosed text as the title of the document. This is rather similar to ConTEXt's `\title` command. The difference is that XML more precisely prescribes which information is tagged, and which names are used for the tagging. This is defined in the DTD. For each document an XML author is free to select a suitable DTD, write a new DTD, or go the way of free structuring and do without a DTD.

Docbook is a large DTD for technical literature, books and articles. It defines the possible structure for such documents. When an author chooses to structure his document according to the Docbook DTD, it can be processed with Docbook utilities. Examples are such utilities as stylesheets like Docbook In ConTEXt, or applications which extract information from the document, like the title and the author names.

The example document in fig. 1 shows the structure of a small Docbook article. Everything is between the `article` start and end tags, which indicates that it is an article and not a book or even a set of books. The first part is the `articleinfo`, with the metadata: title and authors. One could add much more information, such as affiliations, revision history, abstract, copyright, etc. Next comes the main text. It consists of sections with titles, subsections and paragraphs. The text in the paragraphs is less structured. It consists of unmarked text, in which special parts are marked, e.g. literal texts, file names, program listings. In XML jargon this is called 'mixed content'. In Docbook one marks especially those parts that are relevant in technical literature. In a history book one would want to mark a different set of notions, and one would therefore use a different DTD defining

```
<?xml version="1.0" ?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
                         "docbookx.dtd" []>
<article>

<articleinfo>
<title>DocBook In ConTeXt, ConTeXt XML mapping for DocBook
documents</title>
<authorgroup>
<author>
<firstname>Simon</firstname>
<surname>Pepping</surname>
</author>
<author>
<firstname>Michael</firstname>
<surname>Wiedmann</surname>
</author>
</authorgroup>
</articleinfo>

<section>
<title>Installation</title>
<para role="first">Change directory to the top directory of one of the
<literal>texmf</literal> trees of your TeX installation,
e.g. <filename>/usr/share/texmf</filename>, and
<command>untar</command> the distribution file
<filename>DocbookInContext.tar.gz</filename>. Then run the command
<command>mktexlsr</command> for that tree, e.g. <command>mktexlsr
/usr/share/texmf</command>.</para>
</section>

<section>
<title>Usage</title>
<programlisting>
\input xtag-docbook
\setupheadertexts[section][pagenumber]
\setupheader[leftwidth=.7\hsize,style=slanted]

% customizations
\setuphead[section][style=bia,number=no,align=right]
\setupepigraph[narrower={1*right},command=\bi]
\setupattribution[command=---]
\setupXMLDBlists[notoc]
\setupXMLDB[background=off]
\def\XMLDBarticleinfotitle#1%
    {\startalignment[middle]\bib #1\stopalignment\blank[1*big]}
\defineXMLattributeaction[para][role][first]{\bf}
</programlisting>
</para>
</section>

</article>
```

Fig. 1: Example of a Docbook article.

DocBook In ConTeXt

Simon Pepping
Michael Wiedmann

Contents

1 Installation

There was things which he stretched, but mainly he told the truth.
— Mark Twain, Huckleberry Finn (1884)

Change directory to the top directory of one of the `texmf` trees of your TeX installation, e.g. `/usr/share/texmf`, and `untar` the distribution file `DocbookInContext.tar.gz`. Then run the command `mktexlsr` for that tree, e.g. `mktexlsr /usr/share/texmf`. Test the result by issuing the command `kpsewhich xtag-docbook.tex`. The reply should be the path of one of the files just installed, e.g. /usr/share/texmf/tex/context/DocbookInContext/xtag-docbook.tex.

2 Usage

Run context on a Docbook XML file using `xtag-docbook.tex` as the map file. Context offers several possibilities to do this, see the context XML manual `example.pdf`.

One option is to construct a TeX file that inputs the mapping file `xtag-docbook.tex`, and in the text block inputs the XML file with the command `\processXMLfilegrouped`. For example:

```
\input xtag-docbook
\setupheadertexts[section][pagenumber]
\setupheader[leftwidth=.7\hsize,style=slanted]
\setuppagenumbering[location=]
\setupitemize[each][packed][before=,after=,indentnext=no]

\starttext
\processXMLfilegrouped{\jobname.xml}
\stoptext
```

Fig. 2: The article of fig. 1 as formatted by ConTEXt with the Docbook In ConTEXt macros.

suitable tags. In the text one may also insert index terms. At the end of the document one could add appendices, bibliographies, etc.

It is the task of a stylesheet like Docbook In ConTEXt to pick up the tags and render their content with appropriate formatting. Program listings are rendered in a monospaced font, and the line layout is preserved. Index terms are saved and used to construct the index. When one looks at the `articleinfo`, one sees that the XML document does not always contain punctuation and spacing. There is no punctuation between the authors. There may be white space between the first name and surname, but it may also be absent. The stylesheet should add punctuation and white space to the rendering as required. Fig. 2 shows how the small document in fig. 1 is rendered by Docbook In ConTEXt.

The Docbook[1] DTD has been available since the early 1990s. Over the years it has evolved into an extensive DTD for all technical literature. Later in the 1990s extensive, customizable stylesheets[2] became available, written in DSSSL. The Jade program, the JadeTEX macro package (and the numerous underlying other LATEX macro packages) and TEX made it possible to format one's Docbook document with the DSSSL style sheets and obtain high-quality printed output. Armed with these free tools one could author, format and print Docbook documents at a time when SGML tools generally were scarce and expensive. With the advent of XML and XSLT more free tools have become available. The Docbook DTD is now available for XML, and the stylesheets have been rewritten in XSLT.

These combined features have made the Docbook DTD the DTD of choice for technical literature. The Linux Documentation Project is one well-known project that switched over from a private DTD to the Docbook DTD. Due to this strong position, the toolset for working with Docbook documents is growing rapidly, see e.g. `http://www.miwie.org/docbookinfo.html`.

How did it start and where is it now?

During EuroTEX 2001 in Kerkrade I had become interested in using ConTEXt because of the beautiful presentation styles used by Hans Hagen and several other speakers. While I was following the ConTEXt email list, I also became interested in ConTEXt's XML capabilities. These seemed so wonderful to me, that I *had* to understand how this could be done using TEX macro programming. I started asking questions. Sometimes Hans answers such questions with the suggestion that one take up some or other project. So he suggested that I start an XML mapping for Docbook.

I really had other plans, but I was so intrigued with ConTEXt's XML capabilities that I could not resist and gave it a start. As an added benefit, I would become more familiar with the Docbook DTD. When I started I certainly was aware that this would not be a small task. Docbook is such a large DTD, allowing its authors to use the hundreds of elements in innumerable combinations. But only while the project evolved did it become evident to me how large it really is.

Michael Wiedmann, who is interested in all possible tools to render Docbook documents, heard about the project soon after I started it. He made several contributions. His support and interest helped me to continue through the difficult phase when a project is no longer new, but you do not yet have anything really usable and you know all too well how much work still has to be done.

Now, a year later, I have some sort of an answer as to how it is possible to program ConTEXt's XML capabilities in TEX macros: Theoretically TEX macro programming is complete in the sense of having the expressive power of a Turing machine. Hans Hagen is one of the few programmers who can turn this theory into practice.

I also have a working XML mapping for DocBook documents in ConTEXt, which I call Docbook In ConTEXt (DIC). It contains good layout instructions for a number of often-used elements in their more common combinations.

Running Docbook In ConTEXt

Before one can typeset an XML file `myfile.xml`, one should create a TEX driver file `myfile.tex`, which should look something like this:

```
\input xtag-docbook

\starttext
\processXMLfilegrouped{\jobname.xml}
\stoptext
```

Then TEX is invoked as: `texexec myfile.tex` to get a dvi file, or as `texexec --pdf myfile.tex` to get a pdf file.

In the driver file `xtag-docbook` is the file name of the module. The XML document is input with the `\processXMLfilegrouped` command. The filename `\jobname.xml` is always correct provided the driver file and the XML file have the same base name.

Alternatively, one can always use the same driver file, in which the name of the XML file is changed each time.

The ConTEXt documentation indicates that one can also run the XML file as

```
texexec --xmlfilter=docbook testxml.xml
```

This will not work because the name of the Docbook In ConTEXt module does not conform to ConTEXt's nam-

1. `http://www.oasis-open.org/docbook/`
2. `http://sourceforge.net/projects/docbook`

ing conventions. It works if the module is renamed as `xtag-doc.tex`.

Customizing Docbook In ConTEXt

A Docbook XML document is a normal ConTEXt document. The commands that make up a ConTEXt document are also at work when a Docbook XML document is processed. They are just one layer away from what the user sees. Therefore the output can be customized with ConTEXt's setup commands as for any ConTEXt document. The setup commands should be given *after* the Docbook In ConTEXt module has been read, so that they override the default setup commands in the module. If you do not give additional setup commands, ConTEXt's defaults are applied. This is an example of a driver file with ConTEXt setup commands:

```
\input xtag-docbook
\setupindenting[medium]
\setupheadertexts[section][pagenumber]
\setupheader[leftwidth=.7\hsize,
             style=slanted]
\setuppagenumbering[location=]
\setupitemize[each][packed]
    [before=,after=,indentnext=no]

\starttext
\processXMLfilegrouped{\jobname.xml}
\stoptext
```

Docbook In ConTEXt also defines a number of setup commands and other customizations of its own. We describe a few of them in the following subsections.

Section blocks ConTEXt always applies pagebreaks around section blocks, and it treats the Table of Contents and the Index as chapters. This behaviour can be changed with the `pagebreaks` option of the `\setupXMLDB` command:

- `\setupXMLDB[pagebreaks=all]`: Default ConTEXt behaviour.
- `\setupXMLDB[pagebreaks=sectionblocks]`: ToC and Index do not start a new page, and they are treated as sections. All other section blocks retain their default ConTEXt behaviour.
- `\setupXMLDB[pagebreaks=none]`: In addition to the `sectionblocks` option, bodymatter, appendices and backmatter do not start a new page.

Titles Titles are formatted with a command of the form `\XMLDB` *element* `title`, where *element* should be replaced with the name of the element to which the title belongs, e.g. `\XMLDBarticletitle`. These commands can be redefined. They take one argument, the title. For example, the article title could be redefined as:

```
\def\XMLDBarticletitle#1%
   {\startalignment[left]
    \bfb #1
    \stopalignment
    \blank}
```

Note, however, that in Docbook documents the article title is often placed in the `articleinfo` part. In that case one should redefine `\XMLDBarticleinfotitle`. Similarly for book and chapter titles. Section, subsection, etc., titles are mapped to ConTEXt's `\section`, `\subsection`, etc., commands. Therefore they can be customized with ConTEXt's usual `\setuphead` command.

blockquote, epigraph and attribution The Docbook elements `epigraph` and `blockquote` have their own setup commands `\setupepigraph` and `\setupblockquote`, which have the following options:

- `narrower`. Both `epigraph` and `blockquote` are formatted using ConTEXt's narrower environment. The value of this option is a list of `left`, `right` and `middle` that is passed on to the `\startnarrower` command. See the ConTEXt documentation for `\startnarrower` for the effect of these settings.
- `quote`. The value is `on` or `off`. When on, quotation marks are applied as with ConTEXt's quotation environment.
- `command`. The value is a command or set of commands, which are applied at the start of the `narrower` environment.

The element `attribution` is customized with the command `\setupattribution`, which has one option: `command`. The value is applied at the start of the attribution.

More customizations Customization has only recently obtained the attention it deserves. By now more setup commands like those for `blockquote` and `epigraph` have been added, and others will follow. The distribution contains a document `Customization.xml` which will contain an up-to-date description of the customization options.

Example of customization Fig. 3 demonstrates the effect of customization. It shows again the article of fig. 1, but this time the driver file contains some extra customization commands.

```
\setuphead[section]
  [style=bia,number=no,align=left]
\setupepigraph
  [narrower={1*right},command=\bi]
\setupattribution[command=---]
\setupXMLDBlists[notoc]
\setupXMLDB[background=off]
```

DocBook In ConTeXt

Simon Pepping
Michael Wiedmann

Installation

There was things which he stretched, but mainly he told the truth.
—Mark Twain, Huckleberry Finn (1884)

Change directory to the top directory of one of the `texmf` trees of your TeX installation, e.g. `/usr/share/texmf`, and `untar` the distribution file `DocbookInContext.tar.gz`. Then run the command `mktexlsr` for that tree, e.g. `mktexlsr /usr/share/texmf`. Test the result by issuing the command `kpsewhich xtag-docbook.tex`. The reply should be the path of one of the files just installed, e.g. /usr/share/texmf/tex/context/DocbookInContext/xtag-docbook.tex.

Usage

Run context on a Docbook XML file using `xtag-docbook.tex` as the map file. Context offers several possibilities to do this, see the context XML manual `example.pdf`.

One option is to construct a TeX file that inputs the mapping file `xtag-docbook.tex`, and in the text block inputs the XML file with the command `\processXMLfilegrouped`. For example:

```
\input xtag-docbook
\setupheadertexts[section][pagenumber]
\setupheader[leftwidth=.7\hsize,style=slanted]
\setuppagenumbering[location=]
\setupitemize[each][packed][before=,after=,indentnext=no]

\starttext
\processXMLfilegrouped{\jobname.xml}
\stoptext
```

Fig. 3: The article of fig. 1 as formatted by ConTeXt with the Docbook In ConTeXt macros and the customization commands discussed in the text on page 393.

```
\def\XMLDBarticleinfotitle#1%
  {\startalignment[middle]%
   \bib #1%
   \stopalignment\blank[1*big]}
\defineXMLattributeaction
  [para][role][first]{\bf}
```

The first command is a normal ConTeXt command, changing the style of the section heads to bold italic, normal size, no number, and right-aligned.

The second command moves the right margin of the epigraph inward by 1 unit (the size of a unit is determined by ConTeXt's `\setupnarrower` command).

The third line does *not* set the font for the attribution to italic, which Docbook In ConTeXt's default setting does.

The fourth command switches the Table of Contents off. The Lists of Figures and Tables are switched

off by default.

The fifth command switches the background of program listings off.

The sixth command redefines the layout of the article title. More precisely it redefines the title in the `articleinfo` element, which is where the title usually lives. There is no setup command for redefining titles. A definition is required of a command that takes one argument, the text of the title.

The seventh command causes the first paragraph of each section to be printed in bold. The command defines an action when the `role` attribute of the `para` element has the value `first`. The action is `\bf`. The stylesheet takes care to put this action in a group. This action works because *this* XML document marks each first paragraph with the value `first` for the `role` attribute (see fig. 1).

Other tools for the same task

Docbook In ConTEXt is not the only tool for typesetting a Docbook document.

The Docbook XSLT stylesheets for printing The canonical tool for typesetting any XML file is XSLT+FO. An XSLT stylesheet is used to define the desired output in terms of Formatting Objects (FO). The FO description can be thought of as a formatter independent layout description. Then an FO processor is used to produce actual printed output, on paper or as an electronic document.

XSLT stylesheets for Docbook, written by Norman Walsh, have been available for several years. They implement a large part of the Docbook elements — implementing all elements seems almost impossible. And they are extensively parametrized, so that users can customize many aspects without modifying the XSLT code.

The objective of XSLT+FO is: one stylesheet, many processors. Several FO processors are available, among which are two free tools: FOP and TEX. FOP is a dedicated FO processor that produces output in PDF and a range of other formats. It is available from the Apache web site.[3]

TEX can be used as an FO processor using David Carlisle's XML parser `xmltex` and Sebastian Rahtz's `passivetex` package. `xmltex` works in much the same way as ConTEXt's XML processor. It allows one to register commands for an element, which will be applied when that element is started or ended. It does not depend on LATEX or ConTEXt.

`passivetex` does the same as Docbook In ConTEXt. It is a collection of commands to be called by `xmltex`. But it does not register commands for the elements of a specific DTD. Its commands apply to Formatting Objects. This is possible because Formatting Objects are written as elements in an XML file. In that way `passivetex` turns TEX into an FO processor.

Unfortunately, neither FOP nor `passivetex` do a good job with the Docbook XSLT stylesheets. The stylesheets make use of features which are not implemented in these FO processors.

The `db2context` project In March 2003 the first release of the `db2context`[4] project was announced. It is a sister project to the `dblatex` project, and shares some code with it.

The `db2context` project uses XSLT stylesheets not to produce FO, nor HTML, but to produce a ConTEXt file. This may at first seem an odd application of XSLT, and it certainly is not as intended. But it is used quite often by LATEX and ConTEXt users, with success. XSLT has good logic to deal with the problems of transforming an XML file. Writing a good transformation in XSLT is certainly easier than writing one in TEX macros.

Another advantage of this approach is that it allows one to customize the resulting ConTEXt file. In that way, what one cannot get out of the `db2context` style sheets, one can achieve by editing the ConTEXt file. This procedure is a major sin against XSLT orthodoxy, because all formatting should, and *can*, be specified in the XSLT stylesheet. But it is a great advantage for those who know how to achieve their desired style in ConTEXt and do not know much XSLT. In such a situation, who is stopped by orthodoxy?

I do not know the current state of the `db2context` stylesheets.

Future plans

Currently, Docbook In ConTEXt is not completely integrated with the ConTEXt distribution. I have strictly used the ConTEXt API wherever I could, and avoided developing my own variants. But I have preferred to develop this module in separation from the development of ConTEXt itself. The time has now come to work on a better integration. I hope this can be achieved over the next year.

If good, customizable XSL stylesheets for Docbook exist, and if ConTEXt could be an FO processor for the resulting output, then why would it be a good idea to spend so much effort on writing a special Docbook stylesheet for ConTEXt?

In the ConTEXt community the idea of a special Docbook stylesheet for ConTEXt has been greeted with enthusiasm. Apparently, here the theory of one stylesheet for many processors succumbs to the practice that users prefer to work with their tools of choice. For a popular set of tools like Docbook and ConTEXt users afford the effort of another style sheet. Such a style sheet

3. `http://xml.apache.org/fop`

4. `http://sourceforge.net/projects/dblatex`

is more manageable for them and running the required tools is easier.

On the other hand, until now, *I* have spent most of the required effort. And *my* answer tends to be: Maybe it is *not* the best way to support Docbook and XML in ConTEXt. Maybe it would be more useful to work on FO mappings in ConTEXt.

Over the past year I have set up this stylesheet. I have investigated the main structure of Docbook and come up with a way to map that to a ConTEXt document. I have implemented a framework for the mapping. I have enjoyed doing all that, and my insight and skills in TEX macro programming have increased immensely. But the time has come that others take this over, add mappings for more elements, add customizations, add new ideas. I plan to move forward to more generic work to support formatting of XML documents using TEX as the typesetting tool.

Availability

Currently, Docbook In ConTEXt is available separately from the ConTEXt distribution, from my web site.[5] Michael Wiedmann's web page[6] with Docbook tools has a link to the Docbook In ConTEXt files.

Programming Docbook In ConTEXt

ConTEXt and XML ConTEXt can take XML documents as input. For that purpose it contains a non-validating XML parser, which recognizes XML tags as markup instructions. And it has an API (Application Programmer's Interface) which allows one to define actions for those tags. This is called mapping XML tags to ConTEXt. A typical mapping instruction is

```
\defineXMLenvironment[element]
   {start action}
   {stop action}.
```

During the start and stop actions one has access to the attribute values of the element. For example, this is how one reads the `align` attribute of an `entry` element (in a table) and issues the corresponding setup command for ConTEXt's `TABLE` environment:

```
\doifXMLvar{entry}{align}%
   {\expanded{\setupTABLE[align=
      \XMLvar{entry}{align}{}]}}
```

ConTEXt's programming interface for XML mapping is robust. Rarely if ever does one get tangled in expansion problems. But, as is seen in the above example, timing the expansion remains an issue: The command to retrieve the attribute value,

```
\XMLvar{entry}{align}{}
```

must be expanded before the setup command can be read by TEX. That is what `\expanded` does.

It is easy, is it not? In principle, writing a mapping for an XML document in ConTEXt is simple. You state which ConTEXt commands you want to use for the start and stop of each element, and ConTEXt takes care of the rest. Practice is more complicated, certainly if you want to write a useful, extensible and customizable mapping for a complicated DTD. In the following sections I discuss a number of noteworthy features of the Docbook In ConTEXt mapping.

Encoding and language

An XML document declares its encoding in the xml declaration at the start of the document. ConTEXt supports several encodings, among them the XML default encoding `utf-8`. Correctly reading an encoding is one thing. Making all characters available that can be addressed by an encoding is quite another thing. Unicode and its utf-8 encoding have brought all characters in the Unicode range, currently more than 50,000, into scope within a single document. At the moment many of these are mapped to 'unknown character'. Work is ongoing to bring more characters within reach of ConTEXt in a single document.

A Docbook document may declare its language in the `xml:lang` attribute of the document element. The Docbook in ConTEXt module contains at the moment translated strings for four languages: English, German, Dutch and Italian. These are used for automatically generated strings, such as the titles of the table of contents, the abstract, and the index.

Features for each element

Context stack Because an XML document has a tree structure, each element in the document has a list of ancestors. I call that the context, or the context stack, which contains the ancestors from the document root to the current element.

An element may push itself onto the context stack when it starts, and pop itself when it finishes. In principle all elements should do so. In practice a number of elements omit this because they or their children do not use the context stack in their formatting.

During formatting, the context stack can be inspected with the following commands:

- `\XMLDBcurrentelement`: The current element's name.
- `\XMLancestor#1`: The name of the ancestor at level `#1`. The current element is at level `0`.
- `\XMLparent`: The name of the current element's parent.

5. `http://www.leverkruid.nl/context`
6. `http://www.miwie.org/db-context/index.html`

- \the\XMLdepth: The depth of the context stack.
- \doifXMLdepth#1: Execute the following instruction if the context stack has a certain depth.
- \XMLDBprintcontext: Print the context stack in the log file (mainly for debugging purposes).

ConTEXt also defines the context stack. I have redefined it because ConTEXt's implementation did not satisfy my plans. Later I simplified my usage of the context stack. ConTEXt's implementation may now be perfectly satisfactory, but I have not checked this.

ConTEXt itself defines \currentXMLelement to hold the name of the current element. But it is only guaranteed to be valid while ConTEXt reads the XML tag. Indeed, the mapping of some start tags in Docbook in ConTEXt emit an \egroup command, which invalidates the value of \currentXMLelement.

Ignorable white space XML has the interesting feature of ignorable white space. It can be used to give the raw XML document a nice formatting and make it fairly readable. (It did not exist in SGML. As a consequence, SGML documents may be practically unreadable in an ASCII editor.) For applications that read the DTD, this feature is rather clear: white space in elements whose content may only consist of elements, is ignorable. For example, when the content model of a section only contains paragraphs, all white space that surrounds the paragraphs is ignorable. Applications like ConTEXt that do not read the DTD must resort to other means to find out whether white space is ignorable or not.

I have introduced a feature that is similar to the mechanism used in XSLT. One can declare that an element preserves white space with the command

```
\defineXMLDBpreservespace#1
```

and that it ignores white space with the command

```
\defineXMLDBstripspace#1
```

For these declarations to work, the elements should be on the context stack, and they and their children should use the command \XMLDBdospaces as the last command in their start and end tags. \XMLDBdospaces has the effect of ignoring spaces following the XML tag if the current element has been declared to ignore spaces.

In practice this is only used by elements that would suffer if white space is not ignored. Note that TEX itself already ignores a lot of white space, viz. all white space that it reads in vertical mode. In the example of white space surrounding paragraphs in a section, TEX would do the right thing by itself.

The correct functioning of \XMLDBdospaces is rather subtle. The following is a generic element mapping:

```
\defineXMLenvironment[xxx]
   {\XMLDBpushelement{\currentXMLelement}
    \XMLDBdospaces}
   {\XMLDBpopelement \XMLDBdospaces}
```

The command \XMLDBdospaces in the start tag is executed while xxx is the current element. So it ignores white space if xxx has been declared to contain ignorable white space. But the same command in the end tag is executed *after* xxx has popped itself from the context stack. So its parent is the current element, and the command ignores white space if that *parent* has been declared to ignore white space. That is indeed exactly what we want, because the spaces following the end tag </xxx> are in the parent's content.

There is a class of ignorable white space that TEX refuses to ignore: blank lines are converted to \par commands by TEX's input scanner, before we can tell TEX whether white space is ignorable or not. Even this does not always matter to TEX, because TEX discards empty paragraphs or paragraphs that consist of white space only. In the above example we could insert blank lines between the paragraphs without ill effect. But a blank line between the start tag of a footnote and its first paragraph has a notably bad effect: it introduces a \par command between the footnote number and the start of the text, so that the footnote number is in a paragraph by itself.

Such harmful blank lines can only be removed by preprocessing of the XML document. I wrote a tool to do that. It is a SAX document handler written in Java, which removes all ignorable white space. I call it 'Normalizer', and it is available on my web site.[7]

The output of this tool is not only good for the ConTEXt mapping. Looking over it is informative for authors of XML documents. Every amount of white space that is left by the tool, is regarded as meaningful white space by XML parsers. Is that really what the author wants?

Every element In principle every element should contain the following commands:

```
\defineXMLenvironment[xxx]
   {\XMLDBpushelement\currentXMLelement
    \XMLDBseparator \XMLDBdospaces}
   {\XMLDBpopelement \XMLDBdospaces}
```

That is, it pushes itself onto the context stack. It checks whether it should typeset a separator. And it checks whether it should ignore following white space. In its end tag, it pops itself from the context stack, and it checks whether its parent should ignore following white space.

The separator is used by such elements as author, which may generate a comma or the word 'and' between consecutive elements. By default it is set to \relax. A parent element should give it a suitable definition to be

7. http://www.leverkruid.nl/context

used by its children, and reset it to the default when it finishes.

Which element is next?

ConTEXt's XML parsing is event based. This means that the parser generates events, such as the start or stop of an element, and calls the associated actions. During the actions one only sees the current event. One cannot look back at past events, except for the data that one saved. One can certainly not look forward to check which elements follow. In contrast, XSLT is tree based. That means that one can scan all elements, preceding and following, in the formatting commands of an element. Event-based parsing may present serious problems to the programmer.

Is there a title? An abstract may but need not have a title. When there is no title, I want to print the default title 'Abstract'. Because of the event-based nature of the parse, one cannot at the start of the abstract look forward to see if a title will follow. One can only try to find a future event at which one may safely conclude that there is no title if one has not yet seen a title.

In an abstract the optional title may be followed by three element types: `para`, `simpara` and `formalpara`. When any of these elements is started, one may safely conclude that either the title has been seen or there is no title.

One solution is to save the title, and to redefine the mappings of each of these three elements, such that they output the title or the default title if there was no title. And then restore their default definitions for the following elements.

Another way to tackle this problem is to save the whole abstract and process it twice. In the first pass we check whether there is a title. During this pass, all output should be suppressed. In the second pass we first output the title or the default title if no title was found in the first pass, and then we output the content. Again this requires a redefinition of the three possible elements that may follow the title, so that they suppress their output at the first pass.

The third option is provided by TEX itself, not by the XML mapping. We redirect the typesetting of the abstract into a `vbox`. At the same time we save the title in the variable `\XMLDBtitletext`, which removes it from the typeset content in the `vbox`. Then we output the saved title or the default title if there is no saved title, and next we output the `vbox`. This is the best option, and I use it.

When I applied this method to other elements with the same problem, e.g. `note`, I noted that the `vbox` disturbs the line spacing. This lessened my satisfaction with this solution.

Just before I finished this article, I realized that the above solutions correspond to XML thinking, in terms of nodes that have or have not yet been seen. A TEX programmer, however, has another option. We can save the content of the abstract, and then check whether the markup string for the title, viz. `<title`, appears in it. I have now implemented this solution. ConTEXt has good string comparison macros. Nevertheless I have written my own solution, because I wanted to be absolutely sure that there is no expansion of the abstract text. I have had my share of expansion problems with accented and other non-ASCII characters.

In a section this solution would be a bit more problematic: we run the risk of saving a large chunk of text. Working with options one or two would not be fun, because there are more elements to be redefined. I think the only viable alternative would be to work with `\everypar`, because `\everypar` is TEX's low-level signal that there is new text. Fortunately, in a section a title is required, so I did not (yet) have to work out this problem.

This is an example of the problems that arise because in an event-based parse it is hard to determine if an optional element is not present. The following section presents an example of the problems that arise because in an event-based parse it is equally hard to determine when a certain group of elements is finished.

Sectioning Like many systems, ConTEXt partitions its document in frontmatter, bodymatter, appendices and backmatter (called section blocks). The section block governs such properties as the numbering of the chapters and sections. I use the end of the frontmatter to print the table of contents.

Docbook does not have the equivalent of section blocks. There is no single element that contains the frontmatter, the bodymatter or the backmatter. Therefore I analysed the top-level structure of a Docbook document, and divided the elements that may occur as top-level elements into frontmatter elements, bodymatter elements and backmatter elements. When the first top-level bodymatter element is seen, the frontmatter is complete and the bodymatter starts. Similarly for the backmatter.

For a book in Docbook the situation is rather clear: The bodymatter starts with the first `part`, `chapter`, `article` or `reference`. For an article the situation is much fuzzier. While I counted only 6 top-level frontmatter elements, I identified 56 top-level bodymatter elements.

The transitions between the other section blocks are fortunately more clearly marked. The complete analysis is contained in the documentation for the module itself.

The situation is programmed using the commands

`\XMLDBmayensurebodymatter`

and

`\XMLDBmayensurebackmatter`

All top-level bodymatter and backmatter elements execute the appropriate command. These commands check if the element is a direct child of the document element, i.e. if the depth of the context stack equals `2`, and if the corresponding section block has not yet been started. The current section block is kept in the variable `\XMLDBsectionblock`.

Note that this means that TEX grouping runs across the XML tree structure. The start of a node may close a section block, i.e., it closes a TEX group.

Specific elements

Tables Docbook uses the CALS table model. The ConTEXt format uses two different table models. One is the tabulate environment, which is based upon TEX's `\halign`. It is quite sensitive to expansion timing errors. The other is the TABLE environment, also called natural tables. It is a very powerful and flexible environment, with many customization possibilities using `\setupTABLE` commands. A special feature of this table model is that rows, columns and cells can be configured both before and after their content has been given, at any time before the end-of-table (`\eTABLE`) command.

Because ConTEXt's natural tables have many similarities to CALS tables, the mapping is in principle very easy: a `row` corresponds to TR, an `entry` to TD, `colspec` elements can be mapped to `\setupTABLE` commands.

There are three main complications.

- The top, bottom, left and right frames of a CALS table are determined by the `frame` attribute of the table; the `rowsep` and `colsep` attributes of the corresponding rows and cells should be ignored.
- CALS tables can have multiple `tgroup` elements, each with their own number of columns, and their own alignment and frame settings (`colspec` elements).
- Each `tgroup` may have its own `thead` and `tfoot` elements, which may contain their own `colspec` elements.

These requirements have resulted in the following model: The `table` element generates a ConTEXt table, i.e. the table float, using the `\placetable` command. Each `tgroup` element generates its own TABLE environment, i.e. the actual table.

The table is not opened by the start tag of the `table`, because at that moment the title is not yet known. Instead, it is opened by the start tag of the first `tgroup` (command `\XMLDBopentable`, which contains the ConTEXt command `\placetable`). The start tag of each following `tgroup` typesets the previous `tgroup` (command `\XMLDBendTABLE`). Before typesetting, the left and right frames are set up. The start tag of the second `tgroup` also sets up the top frame. The end tag of the `table` does the same as the start tag of the next `tgroup` would do. In addition it sets up the bottom frame of the table, and closes the `vbox` of the `\placetable` command.

The rest is careful attribute processing, and issuing the required `\setupTABLE` commands at the right time. Attribute processing generates a lot of overhead, because both the attribute names and their possible values have to be translated from CALS to `\setupTABLE`. That makes the code somewhat less readable, but the logic is quite straightforward.

Issuing the required `\setupTABLE` commands is a precise work.

- The start tag of the first `tgroup` applies the `frame`, `colsep` and `rowsep` attributes of the Docbook table (`\XMLDBopentable`), so that they apply to all TABLEs in this CALS table. The start tag of each `tgroup` applies its own `align`, `colsep` and `rowsep` attributes, within its own TABLE environment.
- `colspec` elements of a `tgroup` apply their attributes to the whole column of this TABLE. The `colspec` elements in the `thead` and `tfoot` elements, on the other hand, must save their attributes (`\XMLDBsavecolspec`); they will be applied per `entry` in the `thead` and `tfoot`.
- `row` elements apply their attributes immediately to the whole row.
- `entry` elements first check whether they are in a `thead` or `tfoot`; if so, they apply the saved `colspec` attributes. Then they apply their own attributes. This order is important. ConTEXt gives precedence to properties set up per cell over properties set up for the whole table or per row or column. But in this case we apply what was originally a column specification per cell, so we must take care of the precedence ourselves.

Revision history The revision history contains a number of revisions. Each revision specifies one or more fields out of five possible fields. I wanted to represent this in a table which should only contain those columns for which at least one revision specifies data. Programming this was my first challenge in this project.

Hans Hagen suggested the solution. The revision history is saved, and then processed twice.

For the first pass of the saved revision history, we define the revision fields such that they register themselves when they occur, but suppress all output. We also count the number of revisions, so that we will know which row must contain the bottom rule of the table.

Now we know which fields occur and we can set up the table and output its header row.

For the second pass we define the revision element such that it outputs the row with the fields. So that the fields are output in the same order as in the header row, regardless of their order in the XML document, we first save the fields in a revision, and at the end tag of the revision we output the whole row in the desired order.

I worked this out both in ConTEXt's tabulate environment and with its natural tables. I decided to keep the solution with the natural tables, because natural tables are more flexible and less prone to expansion errors.

This procedure demonstrates a powerful feature of ConTEXt's XML processing: It is possible to save a node of the XML document with its subtree; in other words, the content of an element, complete with embedded elements, is saved in a variable without parsing. Later one can process the saved subtree as often as one likes. In between one is free to redefine the behaviour of the embedded elements. In TEX's macro language this is quite normal behaviour:

```
\def\savevar#1{\def\var{#1}}
... % redefine \processvar
\processvar{\var}
... % redefine \processvar
\processvar{\var}
```

In other programming languages it is not nearly as easy. Saving a node with its subtree in a SAX content handler so that it can be processed later is not a trivial task.

It is a disadvantage of the above procedure that the code is not easily read, certainly not if one is not used to the procedure. Recently, I have discussed an alternative procedure using Giuseppe Bilotta's `xdesc` module. It would achieve the same result but make the programming more transparent. Another advantage would be that it is more easily customizable by the user.

Program listing and CDATA I have spent an enormous amount of time on program listings. At first it seemed easy: ConTEXt has a verbatim environment which suits our purpose.

Then it was pointed out to me that some program listings contain `CDATA` sections, which were not treated well by my solution. I realized that a program listing is not really a verbatim environment because it does not disable XML tags. I dived deep into ConTEXt's verbatim environment and came up with a variant that supported two types of verbatim: one real verbatim for `CDATA` sections and one that did only line oriented layout for program listings. Moreover, it was nestable, so that it could deal with `CDATA` sections within program listings.

But it remained problematic to get it quite right. When the end of the `CDATA` section or of the program listing element was followed by text on the same line, this text was lost. And my white space tool did exactly that: put the following text right behind the end of the program listing element.

When I revisited the problem a few months later it dawned on me that the whole verbatim approach was wrong. Neither `CDATA` sections nor program listing environments have anything to do with TEX's notion of verbatim. `CDATA` sections just disable XML markup. They may occur anywhere in an XML document, and have no semantic meaning. Indeed, an XML parser does not even report whether `CDATA` sections are used in an XML document; it simply resolves them.

For the program listing I found a simple solution. It avoids scanning a whole line at a time, therefore it avoids scanning the text following the end of the program listing with the wrong catcodes in place. It uses `\obeylines` and `\obeyspaces` and it places struts at the start of a line to prevent the leading spaces to be discarded by TEX's paragraph mechanism. That is all, and it does the job well.

Hyperlinks, URLs and external documents Docbook documents mark hyperlinks with the `ulink` element; the url is contained in its `url` attribute. If we were writing HTML documents it would be easy:

```
<ulink url="URL">text</ulink>
```

would be translated to:

```
<a href="URL">text</href>
```

and the browser would do the rest.

But this does not suffice for PDF documents. Links to PDF documents should be treated differently from links to other documents, and relative links to non-PDF documents are not allowed. Therefore, we have to analyse the URL and complete it if necessary.

In ConTEXt strings can be split into parts with commands like

```
\beforesplitstring
    string\at substring\to\var
```

which splits `string` at `substring` and stores the first part in `\var`. I use this and similar commands to check whether the URL has an "authority" (this is the term used by RFC2396, which specifies URIs; usually it is called the protocol, e.g. `http`) and whether it is an absolute or a relative URL. If a local file is specified, we also check whether it has the extension `pdf`. Links to local PDF documents are created using the ConTEXt command `\useexternaldocument`, links to other documents use the ConTEXt command `\useURL`.

URLs like `slashdot.org` pose a special problem. Is it a web server, or a file in the current directory? Cf. the URL `myfile.html`, which has exactly the same pattern. After the terminology of RFC2396 I call these abbreviated URLs. By default they are not recognized.

Thus `myfile.html` is correctly linked to as a local document, while `slashdot.org` is incorrectly linked similarly. The user can switch recognition of abbreviated URLs on by setting `\XMLDBcheckabbrURItrue`, and can switch if off again with `\XMLDBcheckabbrURIfalse`.

Unfortunately, I do not know how to get the working directory in a ConTEXt run, so relative URLs are currently not properly completed.

Customization

For a long time I did not pay much attention to customization. Recently, I received requests to make a mapping for the `blockquote` and `epigraph` elements. Together with that request a discussion arose on the ConTEXt mailing list about customization. As a consequence, these two elements and their child element `attribution` have proper setup options, namely, the commands `\setupblockquote`, `\setupepigraph` and `\setupattribution` (discussed on page 393). Since then several customization options have been added and surely more will follow.

The same discussion on the ConTEXt mailing list touched upon attributes whose range of values is not constrained. An example is the `role` attribute of any element. It is not possible to define actions for such attributes in the stylesheet, because the possible values are not known. The idea arose to put a hook in the stylesheet for the user's own formatting command, called attribute action. The user can define such an action as follows:

```
\defineXMLattributeaction
    [para][role][first]{\bf}
```

The stylesheet invokes the attribute action within a group in order to limit the user's actions, such as changing the font, to this element. Therefore ConTEXt cannot invoke the attribute action automatically; it cannot know where it should do so. For example, some mappings for the opening tag invoke `\egroup`; if the attribute action had been invoked automatically, its scope would be ended immediately.

The author of a Docbook document may provide a value for the `role` attribute for every element. But I am wary of enabling an attribute action for every element. It would put every element in a group, and I am not sure about the effects. For now attribute action has experimentally been implemented for `para` and `programlisting`.

I am not sure how far customization can go. Enabling extreme customizability would come down to defining a new language for describing the formatting of a Docbook document. This would go too far. On the other hand, customizability is a strong feature of ConTEXt. It is not difficult to add customizability options to the stylesheet; ConTEXt has some good commands for that.

Docbook in `xmltex`

Would it have been possible to write these stylesheets for Docbook based on `xmltex`? Of course.

Since `xmltex` is independent of LATEX or ConTEXt, it would in principle even be possible to write Docbook in ConTEXt in `xmltex`, that is, to use `xmltex` as the parser and base the commands on ConTEXt. But, as expected, there are conflicts when `xmltex` and ConTEXt are used together.

`xmltex` and LATEX would have been a good basis for a Docbook stylesheet. The approach would in principle be the same as that used in the current project. Like the ConTEXt XML parser, `xmltex` allows one to register actions for each element. The syntax regarding attribute specifications is a little different. Both allow one to specify an action for the start and one for the end of an element.

The major difference would be that between LATEX and ConTEXt, which is quite large. Three significant differences may be mentioned.

- XML support in ConTEXt is actively maintained and evolving. `xmltex` is not.
- Unicode and utf-8 support in LATEX is extensive, due to components of the `passivetex` package. In ConTEXt many Unicode symbols are yet undefined at the time of this writing. It would be an advantage if `passivetex`'s Unicode support would be ported to ConTEXt.
- ConTEXt has a strong integration of MetaPost and PDF. I have not made much use of this feature. For an example of what is possible, see the admonition symbols in Docbook In ConTEXt produced with MetaPost.

Acknowledgements

Michael Wiedmann contributed the earliest mappings for several elements, a.o. `mediaobject`, `table`, and `ulink`. He also contributed the implementation of string literal files, and the string literals for English and German. Giuseppe Bilotta contributed the string literals file for Italian, and Pablo Rodriguez contributed the same for Spanish. Richard Rascher-Friesenhausen made several contributions, a.o. the MetaPost admonition icons.

And of course, nothing of this would have been possible without Hans Hagen's ConTEXt. ConTEXt is the framework in which Docbook In ConTEXt runs and a rich source of examples of excellent TEX macro programming.

Intertextualité et typographie numérique : considérations sémantiques sur le livre électronique

Ioannis Kanellos
Département IASC
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
Ioannis.Kanellos@enst-bretagne.fr

Abstract

Visualizing the book of the future, one might consider multi-modal integrations, non-linear reading strategies, Internet distribution, and customizable configurations to suit different user/reader profiles. In the first part of this paper, we discuss such concepts critically. The second part deals with a more profound aspect of electronic documents, one which could modify the very act of reading: the ability to dynamically control intertextuality. The third part of the paper discusses this last notion and its consequences for the future form of the book. We conclude with some brief comments on alternative forms of typography necessary for such a book.

Résumé

En pensant au futur du livre, on met l'accent sur sa capacité d'intégrer la multi-modalité, les stratégies de lecture non-linéaire qu'il permet, sa transmissibilité via le réseau, et sa configurabilité qui rend possible son adaptation à différents profils d'utilisateur. Dans la première partie de l'article nous discutons ces thèses dans un objectif de réfutation. Dans la seconde partie de la contribution on est concerné par un élément plus profond du document électronique, puisque modifiant la pratique même de la lecture : sa capacité de gérer l'intertextualité de manière dynamique. La troisième partie discute de l'intertextualité et de ses conséquences pour la forme future du livre. Nous concluons avec quelques brèves considérations sur une typographie alternative pour un tel livre.

Le livre électronique : fantasme en voie de réalisation ?

On parle beaucoup de nos jours du livre électronique. Peut-être même *des* livres électroniques. Mais probablement on fantasme plus encore. Comme dans toute époque, le paradigme technologique dominant instrumentalise notre pensée, cadre nos catégories d'espoir en matière de développement scientifique, oriente nos expectatives, et exerce, par conséquent, des tensions sur notre manière d'appréhender une réalité — une chose, une situation, un fait, et même une pratique. C'est un tel paradigme qui fonde, le plus souvent, nos projections d'images de progrès, que nous aimons périodiquement déconstruire et reconstruire, en pronostiquant sur ou simplement en préparant l'avenir.

Le concept du livre n'en ferait pas exception. D'autant plus que sa place et sa fonction dans la constitution des savoirs et la médiation culturelle semblent encore importante et sa présence dans les événements culturels récurrente et toujours reconnue. Il devient, par conséquent, un objet susceptible de faire l'écran de nos projections. Il est effectivement aisé de voir que la quasi-totalité des conceptions sur le livre électronique d'aujourd'hui prennent non seulement pour modèle technologique, mais aussi reproduisent au fond, sciemment ou non, un ordinateur. Tout en important, à côté, les pratiques et usages qui lui sont réservés. Le livre futur, dans sa version électronique donc, serait ainsi une reprise particulière, mais pas significative, du document électronique, récemment domestiqué par les déjà vieilles Nouvelles Technologies d'Information et de Communication (NTIC).

Chose étonnante, il n'y avait pas longtemps qu'on prenait le même modèle pour penser le fonctionnement du cerveau humain. La désillusion est acquise et capitalisée en ce qui concerne l'ordinateur en tant que modèle du cerveau humain. Probablement, y en aura-t-il une nouvelle, en ce qui concerne précisément le livre électronique. Du moins, en ce qui concerne les rapports intrinsèques entre la conception traditionnelle du livre et celle du livre électronique. Pour l'instant, et malgré un début toujours à reprendre, et des échecs avoués de part et d'autre de l'Atlantique, les temps portent toujours la fiction, et continuent à alimenter les débats. Et les caisses aussi. Au moins certaines. Le livre électronique continue à intéresser et à nourrir les espoirs, de pouvoir économique à ne pas rater ou même à usurper dans le secteur de l'édition, prioritairement.

Cependant, il n'y a pas seulement l'ordinateur et — si l'on veut être général et abstrait — l'apprivoisement du calcul par les technologies de nos jours, derrière les spéculations dédiées à l'avenir du livre. Il y a aussi tout ce qui est «autour» de l'ordinateur. Notamment, l'homme et toute la problématique de l'interaction entre l'homme et la machine. Et les services escomptés d'une telle interaction, comme valeur ajoutée à la nouvelle forme du livre. Et bien sûr, incontournable, «*the Matrix*»...Je voulais dire, le réseau. Dit autrement, cette société faite d'ordinateurs, d'hommes et de services désormais possible grâce aux réseaux.

C'est dans ce cadre qu'on pense le renouveau obligé du livre. Cependant, nos réflexions sont majoritairement canalisées par les catégories technologiques. Le livre électronique sera bien sûr fondé sur des matériaux modernes qui écarteront le papier comme support (écran, sans doute, mais aussi papier ou encre électronique, *etc.*) ; il n'aura pas nécessairement la forme d'un livre classique (on pense déjà à un «mutant» entre le livre traditionnel et l'ordinateur) ; il intégrera un ensemble d'éléments provenant de cette pratique nouvelle qu'on a déjà acquise en interagissant avec un ordinateur, au moins, présentera-t-il certaines facilités qu'on retrouve désormais de manière standard dans les logiciels de traitement de texte ; il maniera la multimodalité ; il pourra sans doute intégrer des fonctionnalités provenant du Traitement Automatique des Langues (TAL) ; probablement même sera-t-il capable de «parler» (il sera, par exemple, capable de transformer, à la demande, le texte en voix parlée) ; il pourra contenir un nombre grandissime de pages ; il exhibera son contenu probablement libéré de la contrainte de la page (et de tout ce que cette notion amène) ; et, ce faisant, il offrira un ensemble de fonctionnalités nouvelles en matière de lecture au lecteur, qui se verra enfin débarrassé de la dictature, plusieurs fois séculaire, de la lecture linéarité qu'impose la typographie classique sur papier[1]. Et, bien entendu, il sera pertinemment connecté, via le réseau précisément, à des bases de contenus (des textes, prioritairement, mais aussi de toute forme de contenu multimédia) ainsi qu'à d'autres lecteurs-utilisateurs, qui peuvent transformer sa lecture individuelle en lecture collective. De ce fait, le livre électronique finira par réduire la distance entre le concept du livre et celui d'une bibliothèque, et sa lecture se fera en élargissant les pratiques traditionnelles de lecture, par l'intégration d'interactions innovantes ainsi que de services de nature bibliothécaire.

Une telle perspective n'est pas nécessairement négative, elle est même séduisante, même si la «futurologie» hors contexte et filiation épistémologique reste un jeu gratuit. Il reste toutefois à évaluer si une telle vision, qui se trouve plus ou moins à la bouche de tout le monde, a des chances de se voir réalisée, à court ou à moyen terme. C'est-à-dire, si un tel «livre» incorporera des potentialités pour assurer le relais à la fois cognitif et culturel que porte encore le livre sous sa forme traditionnelle. C'est la question de fond. Nous pouvons la reformuler ainsi : dans quelle mesure un tel livre peut-il assurer le rôle d'un médiateur d'importance dans la transmission des connaissances et des cultures ? Gardera-t-il sa fonction sur le plan des processus d'échanges sociaux ou bien mouvant de plus en plus dans un terrain hybride, sera-t-il supplanté par l'avancement d'une culture de l'image qui augmente ses pouvoirs et ses prétentions par la popularisation du multimédia ?

S'il est vrai que nous avons la sensation de vivre les premières années de l'ère «post-Gutenberg», notre impression ne vient pas de la modification des objets mais d'une mutation des pratiques, essentiellement celles qui vont avec la lecture. Certes, le support peut parfois opérationnaliser des catégories de pensée alternatives, voire innovantes, et contribuer, à terme, au changement des pratiques. Cependant, une telle affaire a généralement un rôle relevant plutôt de l'incidence, et s'inscrit dans une temporalité culturelle, dépassant celle d'une vie humaine. Sa socialisation se fait précisément lentement. L'objet ne s'approprie qu'à travers ses pratiques. Le nouvel objet appelle, donc, à de nouvelles pratiques.

D'où vient cette mutation des pratiques ?

Par les «défauts» et les insuffisances de l'édition papier, dit-on habituellement. En effet, on a souvent considéré l'édition sur papier entachée d'un ensemble de «défauts» ou de déficiences, que l'avènement du document électronique révèle mais aussi corrige. En réalité, si nous imaginons le futur livre avec toutes ces fonctionnalités dont nous parlions tout à l'heure, c'est parce que nous connaissons, déjà, ce qu'est le document électronique. Nous le «pratiquons» déjà, tant comme producteurs que comme consommateurs *c.-à-d.* tant comme auteurs que lecteurs de documents «zéro-papier». On sait aujourd'hui qu'on peut lire ailleurs et autrement que sur du papier.

J'ai expliqué ailleurs [6] que de telles accusations du livre traditionnel (et plus généralement du document) sont plutôt précipitées et injustes. L'édition sur papier supporte largement plusieurs formes de multimodalité (au moins en ce qui concerne un ensemble d'éléments de type visuel pouvant accompagner le texte) et elle ne comporte pas de prescription qui astreint à une lecture linéaire[2]. Inversement, l'édition sur papier peut encore

1. Du moins, depuis qu'on est passé du rouleau à la page.

2. Lit-on un journal ou une revue ou une encyclopédie du début à la fin, de manière linéaire ? Même un roman de facture classique ne contient rien qui nous contraint à une lecture linéaire, cette dernière n'étant qu'une modalité de lecture, certes usuelle et stéréotypée, peut-être aussi une hypothèse légitime au moment

attester d'un ensemble d'éléments esthétiques et ergonomiques que la publication électronique ignore toujours et, souvent, est loin de pouvoir réaliser, éléments qui sont également signifiants pour certaines formes de lecture. De l'autre côté, on ne peut pas faire de la quantité et les performances associés à un tel critère (en l'occurrence le nombre des pages et leur gestion) un trait essentiel de la différence entre ancien et nouveau livre[3]. Ni des fonctionnalités associées, au risque de voir d'autres objets (comme par exemple le film cinématographique voire la télévision) revendiquer leurs dividendes des actions investies dans la bourse du nouveau livre. Ce sont là des critères du moyen non pas de la notion.

À mon avis, le livre électronique sera un point d'arrêt au sein d'un ensemble de nouvelles législations sur la matière publiée, traversant les frontières linguistiques et spécificités culturelles, une mutation méta-stable[4] des habitudes commerciales et de consommation en la matière, de la technique et de la technologie à côté des standards imposés et plus ou moins pérennes, bien sûr. Il sera sans doute un nouvel objet, effectif et manipulable, mais qui, rétrospectivement, confirme et renforce, une nouvelle pratique. Une nouvelle pratique de lecture.

On sous-estime souvent cet élément. Probablement parce que le concept de pratique convoque un cadre conceptuel difficilement formalisable dans les normes d'une science «authentique» — c'est-à-dire, «dure». C'est un concept vague, non quantifiable, qui amène vite nombre de thèmes liés à l'empirisme et à la vie, individuelle et socialisée, et se configure par des notions qui tournent autour du sens et de la sémantique — terrains exclus de la bonne science, et accessoirement de l'informatique, rappelons-le. Certes, «pratique» désigne, entre autres, l'application des principes et des règles d'une science ou d'une technique, mais surtout, dans notre cas, une forme de connaissance acquise à travers l'expérience, induisant des habitudes, probablement devenant un «habitus cognitif». Elle fait appel à une activité humaine, doublement subjective et intersubjective, historiquement et culturellement déterminée, qui donne et qui récupère du sens. En effet, tout rapport à un objet, pour être signifiant, doit s'attester à travers une pratique. Une pratique qui décline des genres pour répondre à des exigences de spécificité, faisant ainsi de l'objet «distant» un objet «intime», parce que participant à un vécu orienté par un objectif. Parce que connu et reconnu, appréhendable et utilisable, faisant partie de notre univers et de nos formes de vie — et donc, nos catégories de pensée. Tout objet reste abstrait, seulement théorique et logique s'il ne participe pas à un commerce sémiotique, qui détermine sa valeur et les besoins qui l'appellent. En pratiquant un objet, nous le chargeons de sens, nous lui donnons place à notre vie, et droit à notre compréhension. Hors de la pratique, l'objet est seulement formel.

Dans notre cas toujours, cette pratique est la lecture. En réalité, le livre n'est la forme instrumentale d'un besoin de lecture. C'est ce besoin qui le définit, et, devenant objet socialisé, implique à notre façon de lire des déterminations particulières lui étant propres. Car nous lisons toujours dans des catégories communes, partagées, qui déploient et exploitent perpétuellement, suivant les supports, la même activité générique, celle de l'usage de la langue. Lire ce n'est une modalité de consommation de la langue. Corrélativement, nous écrivons en convoquant les mêmes catégories.

Il est à constater, non sans un certain regret, que peu d'études sont menées sur le rapport entre la typographie et le contenu de la lecture. Peut-être, ne sont-elles pas toujours décisives ou clairement convaincantes dans la mesure où le terrain de la sémiotique, terrain obligé dans de telles études, reste instable sinon encore indécis [1, 2, 4].

Quoi qu'il en soit, nous sous-estimons parfois, dans notre soif de nouveauté, et même encore dans notre traditionalisme typographique, qu'un livre sert (et même est), avant tout, un *contenu*. Notre constat est trivial : il n'y a pas de livre sans contenu, et l'on n'écrit jamais un livre sans vouloir transmettre un contenu, sans l'hypothèse qu'il va être lu. Instrument fondamental dans la transmission des connaissances et plus généralement de la culture, un livre est fait (par quelqu'un) pour être lu (par quelqu'un d'autre). C'est cet objectif que, normalement, le livre, traditionnel ou nouveau, et la typographie qu'il appelle, doit servir. Au mieux. Si la typographie — typographie papier ou électronique — n'arrive pas à aider à mieux comprendre le contenu qu'il véhicule à toutes les dimensions de sa signifiance, au moins, peut-elle se donner comme objectif modeste de ne pas nuire sa compréhension, en nuisant précisément sa lecture. En d'autres termes, le livre, par le biais de la typographie, cherche au niveau de la présentation du contenu dans un espace de pratiques partagées, à rendre une lecture maximalement opérationnelle.

de l'écriture, mais résolument une parmi d'autres. D'ailleurs, il ne faut pas le rôle du feuilletage, qui préfigure la lecture, s'il ne la représente pas carrément en une temporalité réduite, et qui a tout sauf la linéarité. Même sur le plan micro- et méso-typographique, les études en psychologie cognitive mettent en évidence des stratégies de perception visuelle qui n'ont rien à faire avec la linéarité, qui reste encore un mythe logique. Non seulement la mise en page, mais aussi la syntaxe et même la sémantique, et les formes de pré-compréhension qu'elle nécessite, concourent à des parcours de lecture largement non-linéaires.

3. La performance des ordinateurs n'a pas altéré la conception de base, toujours stable, toujours de type «von Neumann», tout comme la contenance d'un support musical n'a pas altéré notre façon d'écouter la musique.

4. *I.e.* revenant toujours à des formes de stabilité renouvelées.

Dans le travail que je citais plus haut[5], j'avais argumenté sur un seul point qui me semblait essentiel dans les tentatives de différenciation entre la forme connue et la forme imaginée du livre, à savoir l'accessibilité à la société des textes dans laquelle revendique une place un texte pour être mieux compris lors de sa lecture. Je me permets de synthétiser cette réflexion pour mieux asseoir mon argument sur ses implications typographiques dans la suite.

Tout d'abord, il faut clarifier le terme mais aussi le thème de la lecture.

Du livre en tant qu'objet de lecture On doit comprendre la lecture comme un ensemble de processus cognitifs (de perception et de structuration prioritairement) orientés par un objectif de compréhension. Ces processus ont pour fonction de reconstruire le lisible, en lui donnant forme interprétable. En le situant, tout d'abord, puis en l'identifiant comme objet à comprendre. Leur fonction est essentiellement structurante voire restructurante. Ils circonscrivent, tout d'abord, l'espace du lisible en définissant le signifiable et sa place dans un environnement sémiotique. Puis ils opèrent sur des éléments signifiants des évaluations, des sélections, des pondérations, des filtrages ou des unifications, induisent des réorganisations, génèrent nombreuses mises en rapport, produisent des suppléances ou des accroissements des données lisibles, tirent les traits vers des thématisations, des virtualisations ou forcent des inhibitions, *etc*. En deux mots, ils opèrent des ré-élaborations de la matière lue en construisant, à divers régimes (thématique, tactique, dialogique, dialectique[6], et à partir d'éléments au moyen desquels se présente (mais aussi se représente) le contenu du livre, des unités signifiantes intégrées, *i.e*. des unités de niveau supérieur [8, 13]. Il s'agit d'un processus complexe, à la fois *bottom-up* et *top-down*, qui mobilise des facultés cognitives tant générales (comme la reconnaissance des formes, l'attention ou la mémoire, par exemple) que spécifiques (organisation des découpages, intégration des unités graphiques élémentaires à des unités plus étendues, *etc*.), dont l'entrée est la matière qui a fait l'objet de préoccupations typographiques en amont, et la sortie une «image» de celle-ci, une ré-élaboration sémantique du livre suivant la forme, la qualité et l'objectif de la lecture [11, passim].

Une fois qu'on a un peu réfléchi sur le concept et le rôle de la lecture, on peut mettre en perspective relative l'avenir du livre. Précisément, le nouveau livre existera lorsqu'on pourra attester une authentiquement nouvelle forme de lecture. De quoi sera-elle faite ?

Je reprends dans la suite cette question en la projetant sur l'écran futuriste du livre électronique, et discerner, accessoirement, quelques traits de son possible horizon typographique.

Un livre, plusieurs livres. La lecture impossible d'un seul livre

Cette idée simple et évidente (que le livre existe pour opérationaliser une lecture) se double d'une autre, aussi simple, aussi évidente, mais étonnamment moins consciente et réfléchie que la précédente : *qu'il est impossible de lire un livre seul* [3]. Formulée positivement, elle stipule que chaque fois qu'on lit un livre, on lit aussi d'autres livres. Réelles ou seulement résidant dans notre mémoire, effectives ou simplement reconstruites, ces lectures mettent en place ce qu'on pourrait appeler de manière quelque peu métaphorique, *le contexte social d'une lecture*. Le principe épistémologique est plutôt classique : il n'y a jamais un seul homme mais des hommes qui font société (et même sociétés) ; on ne rencontre jamais une seule vie mais des vies, organisées suivant des rapports fonctionnels et téléologiques ; il n'y a pas une idée sans l'existence d'autres idées…Il n'y a pas une lecture mais des lectures. Mieux, toute lecture spécifique n'est que le passage d'un faisceau de lectures qui la soutiennent et la rendent opératoire. Tout simplement parce qu'il n'y a jamais de compréhension qui tienne toute seule.

Contrairement à ce qu'on dit souvent et qu'une *doxa* dominante laisse croire, Gutenberg n'a pas inventé comment faire *un* livre. Mais comment faire *des* livres. Pas un livre en plusieurs exemplaires. Mais plusieurs livres en plusieurs exemplaires, candidats à une lecture par une collectivité de lecteurs. Au fond, ce qui s'est produit avec le livre imprimé est une société de livres. Qui dresse le parallèle d'une société humaine en la projetant sur l'acte de lire. Et dont l'effet marquant est celui de la productivité de la lecture, et même *des* lectures. On ne doit pas s'en douter : même dans la typographie classique la pratique de la lecture, nécessairement plurielle, a institué un ensemble de moyens pour faire référence à la société à laquelle appartient le livre, et plus généralement le document, qu'on est en train de lire. Sur les figs. 1–4 on en voit certaines, de manière schématique.

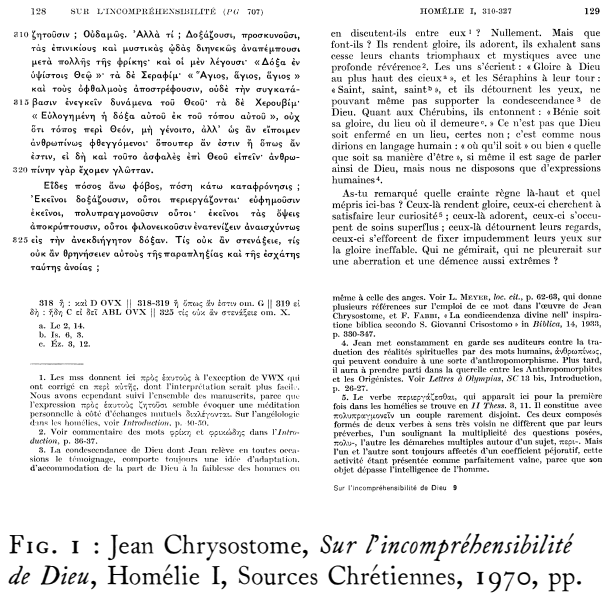
La typographie servant les indications, les références, les commentaires, *etc*., et plus clairement encore, par la mise en page de fragments textes issus d'ailleurs, sont autant de moyens d'explicitation du contexte social de la lecture. Et autant de moyens pour cadrer les parcours qu'elle est susceptible de déclencher et donc, l'interprétation escomptée du texte.

Dans le cadre d'un document électronique, on retrouve également sans difficulté ses repères : A-t-on vraiment innové ?

Sur la forme, sans doute non. On pourrait même penser le contraire. Les structures logiques et physiques

5. [6, p. 102 *sq*.]

6. Cf. [9, première partie], mais aussi, sur un plan plus sémiotique [10] voire rythmique *etc*.



128 SUR L'INCOMPRÉHENSIBILITÉ (*PG* 707) HOMÉLIE I, 310-327 129

310 ζητοῦσιν ; Οὐδαμῶς. Ἀλλὰ τί ; Δοξάζουσι, προσκυνοῦσι, τὰς ἐπινικίους καὶ μυστικὰς ᾠδὰς διηνεκῶς ἀναπέμπουσι μετὰ πολλῆς τῆς φρίκης· καὶ οἱ μὲν λέγουσι· « Ἁγία δόξα ἐν ὑψίστοις Θεῷ » τὰ δὲ Σεραφίμ· « Ἅγιος, ἅγιος, ἅγιος » καὶ τοὺς ὀφθαλμοὺς ἀποστρέφουσιν, οὐδὲ τὴν συγκατά-
315 βασιν ἐνεγκεῖν δυνάμενα τοῦ Θεοῦ· τὰ δὲ Χερουβίμ· « Εὐλογημένη ἡ δόξα αὐτοῦ ἐκ τοῦ τόπου αὐτοῦ », οὐχ ὅτι τόπος περὶ Θεόν, μὴ γένοιτο, ἀλλ' ὡς ἂν εἴποιμεν ἀνθρωπίνως φθεγγόμενοι· ὅπουπερ ἂν ἐστιν ἢ ὅπως ἂν
320 ἐστιν, εἰ δὴ καὶ τοῦτο ἀσφαλὲς ἐπὶ Θεοῦ εἰπεῖν· ἀνθρωπίνην γὰρ ἔχομεν γλῶτταν.

Εἶδες πόσος ἄνω φόβος, πόση κάτω καταφρόνησις ; Ἐκεῖνοι δοξάζουσιν, οὗτοι περιεργάζονται· εὐφημοῦσιν ἐκεῖνοι, πολυπραγμονοῦσιν οὗτοι· ἐκεῖνοι τὰς ὄψεις ἀποκρύπτουσιν, οὗτοι φιλονεικοῦσιν ἐναντίζειν ἀναισχύντως
325 εἰς τὴν ἀνεκδιήγητον δόξαν. Τίς οὖν ἂν στενάξειε, τίς οὐκ ἂν θρηνήσειεν αὐτοὺς τῆς παραπληξίας καὶ τῆς ἐσχάτης ταύτης ἀνοίας ;

318 ἦ : καὶ D OVX || 318-319 ἦ ὅπως ἂν ἐστιν om. G || 319 εἰ δὴ : ἤδη C εἰ δεῖ ABL OVX || 325 εἰς οὐκ ἂν στενάξειε om. X.

a. Le 2, 14.
b. Is. 6, 3.
c. Ez. 3, 12.

1. Les mss donnent ici πρὸς ἑαυτούς à l'exception de VWX qui ont corrigé en περὶ αὐτῆς, dont l'interprétation serait plus facile. Nous avons cependant suivi l'ensemble des manuscrits, parce que l'expression πρὸς ἑαυτούς ζητεῖν semble évoquer une méditation personnelle à côté d'échanges mutuels développés. Sur l'angélologie dans les homélies, voir *Introduction*, p. 30-50.

2. Voir commentaire des mots ὁρᾶσις et φρικώδης dans l'*Introduction*, p. 36-37.

3. La condescendance de Dieu dont Jean relève en toutes occasions le témoignage, comporte toujours une idée d'adaptation, d'accommodation de la part de Dieu à la faiblesse des hommes en

en discutent-ils entre eux[1] ? Nullement. Mais que font-ils ? Ils rendent gloire, ils adorent, ils exhalent sans cesse leurs chants triomphaux et mystiques avec une profonde révérence[2]. Les uns s'écrient : « Gloire à Dieu au plus haut des cieux[a] », et les Séraphins à leur tour : « Saint, saint, saint[b] », et ils détournent les yeux, ne pouvant même pas supporter la condescendance[3] de Dieu. Quant aux Chérubins, ils entonnent : « Bénie soit sa gloire, du lieu où il demeure[c]. » Ce n'est pas que Dieu soit enfermé en un lieu, certes non ; c'est comme nous dirions en langage humain : « où qu'il soit » ou bien « quelle que soit sa manière d'être », si même il est sage de parler ainsi de Dieu, mais nous ne disposons que d'expressions humaines[4].

As-tu remarqué quelle crainte règne là-haut et quel mépris ici-bas ? Ceux-là rendent gloire, ceux-ci cherchent à satisfaire leur curiosité[5] ; ceux-là adorent, ceux-ci s'occupent de soins superflus ; ceux-là détournent leurs regards, ceux-ci s'efforcent de fixer impudemment de leurs yeux sur la gloire ineffable. Qui ne gémirait, qui ne pleurerait sur une aberration et une démence aussi extrêmes ?

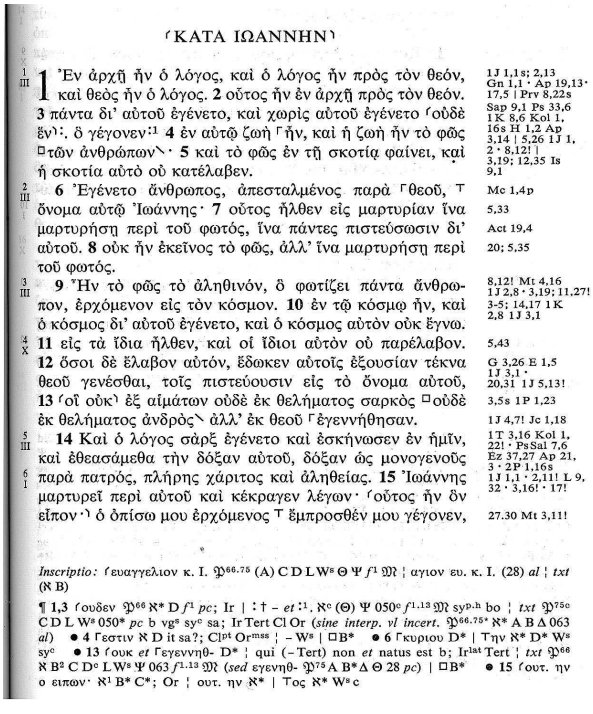
même à celle des anges. Voir L. Mayue, *loc. cit.*, p. 62-63, qui donne plusieurs références sur l'emploi de ce mot dans l'œuvre de Jean Chrysostome, et F. Fann, « La condiscendenza divine nell' inspirazione biblica secondo S. Giovanni Crisostomo » in *Biblica*, 14, 1933, p. 330-347.

4. Jean met constamment en garde ses auditeurs contre la traduction des réalités spirituelles par des mots humains, ἀνθρωπίνως, qui peuvent conduire à une sorte d'anthropomorphisme. Plus tard, il aura à prendre parti dans la querelle entre les Anthropomorphites et les Origénistes. Voir *Lettres à Olympias*, *SC* 13 bis, Introduction, p. 26-27.

5. Le verbe περιεργάζεσθαι, qui apparaît ici pour la première fois dans les homélies se trouve en *II Thess.* 3, 11. Il constitue avec πολυπραγμονεῖν un couple rarement disjoint. Ces deux composés formés de deux verbes à sens très voisin ne diffèrent que par leurs préverbes, l'un soulignant la multiplicité des questions posées, πολυ-, l'autre les démarches multiples autour d'un sujet, περι-. Mais l'un et l'autre sont toujours affectés d'un coefficient péjoratif, cette activité étant présentée comme parfaitement vaine, parce que son objet dépasse l'intelligence de l'homme.

Sur l'incompréhensibilité de Dieu 9

FIG. 1 : Jean Chrysostome, *Sur l'incompréhensibilité de Dieu*, Homélie I, Sources Chrétiennes, 1970, pp. 128–129.



ΚΑΤΑ ΙΩΑΝΝΗΝ

1 III 1 Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος. 2 οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν θεόν. 3 πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο ⌜οὐδὲ ἕν⌝· ὃ γέγονεν· 4 ἐν αὐτῷ ζωὴ ⌜ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς □τῶν ἀνθρώπων⌝· 5 καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

2 III 6 Ἐγένετο ἄνθρωπος, ἀπεσταλμένος παρὰ ⌜θεοῦ, ⊤ ὄνομα αὐτῷ Ἰωάννης· 7 οὗτος ἦλθεν εἰς μαρτυρίαν ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. 8 οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.

3 III 9 ⊤Ἦν τὸ φῶς τὸ ἀληθινόν, ὃ φωτίζει πάντα ἄνθρωπον, ἐρχόμενον εἰς τὸν κόσμον. 10 ἐν τῷ κόσμῳ ἦν, καὶ ὁ κόσμος δι' αὐτοῦ ἐγένετο, καὶ ὁ κόσμος αὐτὸν οὐκ ἔγνω.

4 X 11 εἰς τὰ ἴδια ἦλθεν, καὶ οἱ ἴδιοι αὐτὸν οὐ παρέλαβον. 12 ὅσοι δὲ ἔλαβον αὐτόν, ἔδωκεν αὐτοῖς ἐξουσίαν τέκνα θεοῦ γενέσθαι, τοῖς πιστεύουσιν εἰς τὸ ὄνομα αὐτοῦ, 13 ⌜οἳ οὐκ⌝ ἐξ αἱμάτων οὐδὲ ἐκ θελήματος σαρκὸς □οὐδὲ ἐκ θελήματος ἀνδρὸς⌝ ἀλλ' ἐκ θεοῦ ⌜ἐγεννήθησαν.

5 III 14 Καὶ ὁ λόγος σὰρξ ἐγένετο καὶ ἐσκήνωσεν ἐν ἡμῖν, καὶ ἐθεασάμεθα τὴν δόξαν αὐτοῦ, δόξαν ὡς μονογενοῦς παρὰ πατρός, πλήρης χάριτος καὶ ἀληθείας. 15 Ἰωάννης 6 1 μαρτυρεῖ περὶ αὐτοῦ καὶ κέκραγεν λέγων· ⌜οὗτος ἦν ὃν εἶπον·⌝ ὁ ὀπίσω μου ἐρχόμενος ⊤ ἔμπροσθέν μου γέγονεν,

1J 1,1a; 2,13 Gn 1,1 · Ap 19,13· 17,5 | Prv 8,22s Sap 9,1 Ps 33,6 1K 8,6 Kol 1, 16s H 1,2 Ap 3,14 | 5,26 1J 1, 2 · 8,12† | 3,19; 12,35 Is 9,1

Mc 1,4p

5,33

Act 19,4

20; 5,35

8,12† Mt 4,16 1J 2,8 · 3,19; 11,27† 3-5; 14,17 1K 2,8 1J 3,1

5,43

G 3,26 E 1,5 1J 3,1 · 20,31 1J 5,13†

3,5s 1P 1,23

1J 4,7† Jc 1,18

1T 3,16 Kol 1, 22† · PsSal 7,6 Ez 37,27 Ap 21, 3 · 2P 1,16s

1J 1,1 · 2,11† L 9, 32 · 3,16† · 17†

27.30 Mt 3,11†

Inscriptio: ⌜ευαγγελιον κ. Ι. $\mathfrak{P}^{66.75}$ (A) C D L W^s Θ Ψ f^1 $\mathfrak{M}$ ¦ αγιον ευ. κ. Ι. (28) *al* ¦ *txt* (ℵ B)

¶ 1,3 ⌜ουδεν $\mathfrak{P}^{66}$ ℵ* D f^1 *pc*; Ir | : † – *et* :[1], ℵ^c (Θ) Ψ 050^e $f^{1.13}$ $\mathfrak{M}$ sy$^{p.h}$ bo ¦ *txt* $\mathfrak{P}^{75c}$ C D L W^s 050* *pc* b vg^s sy^c sa; Ir Tert Cl Or (*sine interp. vl incert.* $\mathfrak{P}^{66.75*}$ ℵ* A B Δ 063 *al*) ● 4 ⌜εστιν ℵ D it sa?; Cl^{pt} Or^{mss} ¦ – W^s | □B* ● 6 ⌜κυριου D* | Τηv ℵ* D* W^s sy^c ● 13 ⌜ουκ *et* ⌜εγεννηθ- D* ¦ qui (– Tert) non *et* natus est b; Ir^{lat} Tert ¦ *txt* $\mathfrak{P}^{66}$ ℵ B^2 C D^c L W^s Ψ 063 $f^{1.13}$ $\mathfrak{M}$ (*sed* εγενηθ- $\mathfrak{P}^{75}$ A B*Δ Θ 28 *pc*) | □B* ● 15 ⌜ουτ. ην ο ειπον· ℵ^1 B* C*; Or ¦ ουτ. ην ℵ* | Τος ℵ* W^s c

FIG. 2 : Évangile Selon Jean, 1–14 ; Nestlé-Aland, *Novum Testamentum Graece*, Deutsche Bibelgesellschaft, Stuttgart, 1979.

[٨٩] ﴿كتابٌ من عند الله﴾ القرآنُ ﴿يَسْتَفْتِحون﴾ يستنصرون على المشركين بالنبي المنتظَر، وكانوا يتوقعون أن يكون من بني إسرائيل ﴿فلما جاءهم ما عرفوا﴾ فلما جاءهم محمدٌ على الصفات التي يعرفونها في التوراة

سُورَةُ البَقَرَةِ ٢

١٤

وَلَمَّا جَآءَهُمْ كِتَٰبٌ مِّنْ عِندِ ٱللَّهِ مُصَدِّقٌ لِّمَا مَعَهُمْ وَكَانُوا۟ مِن قَبْلُ يَسْتَفْتِحُونَ عَلَى ٱلَّذِينَ كَفَرُوا۟ فَلَمَّا جَآءَهُم مَّا عَرَفُوا۟ كَفَرُوا۟ بِهِۦ ۚ فَلَعْنَةُ ٱللَّهِ عَلَى ٱلْكَٰفِرِينَ ﴿٨٩﴾ بِئْسَمَا ٱشْتَرَوْا۟ بِهِۦٓ أَنفُسَهُمْ أَن يَكْفُرُوا۟ بِمَآ أَنزَلَ ٱللَّهُ بَغْيًا أَن يُنَزِّلَ ٱللَّهُ مِن فَضْلِهِۦ عَلَىٰ مَن يَشَآءُ مِنْ عِبَادِهِۦ ۖ فَبَآءُو بِغَضَبٍ عَلَىٰ غَضَبٍ ۚ وَلِلْكَٰفِرِينَ عَذَابٌ مُّهِينٌ ﴿٩٠﴾ وَإِذَا قِيلَ لَهُمْ ءَامِنُوا۟ بِمَآ أَنزَلَ ٱللَّهُ قَالُوا۟ نُؤْمِنُ بِمَآ أُنزِلَ عَلَيْنَا وَيَكْفُرُونَ بِمَا وَرَآءَهُۥ وَهُوَ ٱلْحَقُّ مُصَدِّقًا لِّمَا مَعَهُمْ ۗ قُلْ فَلِمَ تَقْتُلُونَ أَنۢبِيَآءَ ٱللَّهِ مِن قَبْلُ إِن كُنتُم مُّؤْمِنِينَ ﴿٩١﴾ وَلَقَدْ جَآءَكُم مُّوسَىٰ بِٱلْبَيِّنَٰتِ ثُمَّ ٱتَّخَذْتُمُ ٱلْعِجْلَ مِنۢ بَعْدِهِۦ وَأَنتُمْ ظَٰلِمُونَ ﴿٩٢﴾ وَإِذْ أَخَذْنَا مِيثَٰقَكُمْ وَرَفَعْنَا فَوْقَكُمُ ٱلطُّورَ خُذُوا۟ مَآ ءَاتَيْنَٰكُم بِقُوَّةٍ وَٱسْمَعُوا۟ ۖ قَالُوا۟ سَمِعْنَا وَعَصَيْنَا وَأُشْرِبُوا۟ فِى قُلُوبِهِمُ ٱلْعِجْلَ بِكُفْرِهِمْ ۚ قُلْ بِئْسَمَا يَأْمُرُكُم بِهِۦٓ إِيمَٰنُكُمْ إِن كُنتُم مُّؤْمِنِينَ ﴿٩٣﴾

الجزء ٢

﴿كفروا به﴾ .. حسداً، لأنه من العرب وليس من بني إسرائيل كما توقعوا [٩٠] ﴿اشتروا به أنفسهم﴾ باعوا به أنفسهم ﴿بغياً﴾ حسداً ﴿فباؤوا بغضب من الله﴾ فرجعوا بغضب من الله بسبب كفرهم ﴿على غضب﴾ على غضب استحقوه من قبلُ بتضييع التوراة والكفر بعيسى عليه السلام [٩١] ﴿وهو الحق﴾ .. الثابت الصحيح (القرآن) ﴿ويكفرون بما وراءه﴾ .. بما سواه [٩٢] ﴿بالبينات﴾ بالمعجزات الدالّة على صدقه كفرق البحر وتظليل الغمام ﴿اتخذتم العجل﴾ جعلتموه إلهاً معبوداً [٩٣] ﴿سمعنا وعصينا﴾ سمعنا قولك وعصينا أمرك ﴿أشربوا في قلوبهم العجل﴾ امتزج بقلوبهم حبُّ عبادة العجل

الدعاء: اللهم إنا نسألك بحق محمد النبي الأمي الذي وعدتنا أن تخرجه لنا في آخر الزمان إلا نصرتنا عليهم. فكانوا إذا التقوا دعوا بهذا فيهزمون غطفان، فلما بُعث النبي عليه الصلاة والسلام كفروا به، فأنزل وكانوا يستفتحون بك يا محمد على الكافرين. وأخرج ابن أبي حاتم من طريق سعيد عن عكرمة عن ابن عباس، أن يهود كانوا يستفتحون على الأوس والخزرج برسول الله ﷺ قبل مبعثه، فلما بعثه الله من العرب كفروا به وجحدوا ما كانوا يقولون فيه، فقال لهم معاذ بن جبل وبشر بن البراء وداود بن سلمة: يا معشر اليهود: اتقوا الله وأسلموا، فقد كنتم تستفتحون علينا بمحمد ونحن أهل شرك، وتخبروننا بأنه مبعوث وتصفونه بصفته، فقال سلام بن أبي مشكم أحد بني النضير: ما جاءنا بشيء نعرفه، وما هو بالذي كنا نذكر لكم؛ فأنزل الله ﴿ولما جاءهم كتاب من عند الله﴾ الآية.

FIG. 3 : *Coran*, Sourate 14, Dar ar-Rachik, Damas–Beïrout.

du document électronique qui font référence aux rapports «intertextuels» peuvent se montrer pratiquement équivalentes de point de vue de leur effet sur la lecture. Cependant, le document électronique se distancie sur les *aspects dynamiques* de leur réalisation. En effet, le passage au support électronique réduit la distance entre le texte sous lecture et les textes qui l'environnent rendant possible cette relation de l'égard par rapport à d'autres productions textuelles (ou multimédia, plus généralement) au sein de la même séquence de travail. Cette réduction de la distance entre *un texte* et *les textes*, qui font avec lui société, cette remise en scène *en temps présent*, à l'intérieur même de la lecture, de son propre environnement documentaire, n'est pas un effet incident ou mineur, dans la mesure où il offre la possibilité de rendre explicite ce qui était jusqu'alors implicite, et tend à substituer la vision atomiste et unitextuelle de la lecture par une autre, plus juste, plus pertinente, plus globale, enfin clairement située. Une lecture ordinaire se donne ainsi les moyens d'une lecture confirmée ou experte, en internalisant les voies d'explication de son héritage documentaire, et de ses conditions, régimes, normes et limites d'intelligibilité. Certes, une lecture experte éduquée et entraînée à l'intertextualité n'y trouve pas grand bénéfice, sauf, peut-être, une économie d'effort. Mais pour une lecture non experte, il y a occasion de transformation de genre. En d'autres termes, le concept de lien intertextuel, comme apport du document électronique, ouvre la lecture novice vers les modalités d'une lecture expérimentée, peut-être

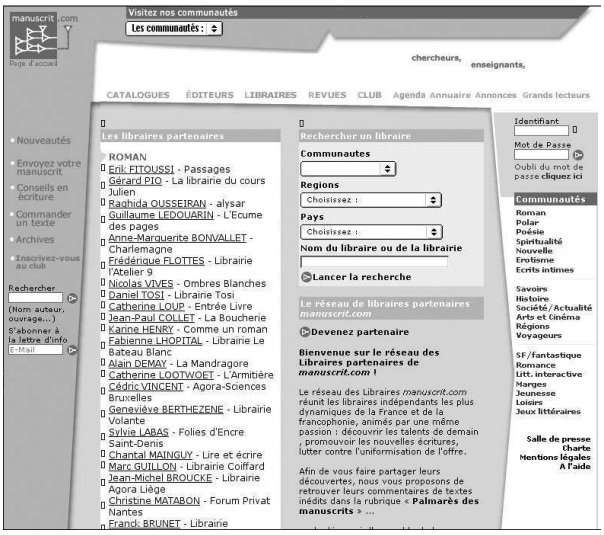


Fig. 4 : WEB Classical Texts : `http ://www.manuscrit.com`.

même savante, et ceci à moindre coût[7].

Ce dernier n'est bien sûr possible qu'à travers le concept-support de réseau. La seule différence fondamentale donc entre lecture classique et lecture sous support électronique serait que l'explicitation des apports d'interprétance, qui retracent les dettes et les filiations intellectuelles du document sous lecture par rapport à un ensemble de sources externes, devient désormais d'indirecte directe, de médiate immédiate, et de statique dynamique. Ce n'est pas une invention, mais simplement un changement de statut opératoire, donc d'utilisation qui, très vraisemblablement, induit des déterminations d'une nouvelle pratique en instrumentalisant différemment l'activité de lecture.

Dans tous les cas, la mise en rapport de lectures au moyen d'une société de textes, que cette société soit imaginée, reconstituée, suggérée ou même réelle, est le *sine qua non* d'une pratique de lecture du point de vue sémantique. C'est seulement elle qui coordonne, affine et corrobore le projet de compréhension que nous associons à notre lecture. Au fond, on le disait tout à l'heure, c'est pour cette raison essentiellement qu'on lit. Pour apprendre, pour connaître, pour vérifier ou affiner un thème, pour trouver une solution à une question qu'on se pose, parfois simplement pour avoir du plaisir ou du réconfort, pour s'échapper aussi, voire même pour s'inspirer. Mais toutes ces activités sont coordonnées par une autre, générique, celle de la compréhension. Qui suppose, à son tour, la faculté d'interprétation. Cette dernière est précisément le résultat d'une stratégie de lecture appliquée sur le document (le livre en particulier), mais aussi, en parallèle, et de façon aussi signifiante, sur la société des documents (textes) dans laquelle il acquiert un statut de document signifiable. La compréhension n'est en réalité que l'aboutissement cognitif d'un parcours de lecture, nécessairement pluriel : à travers le document (livre) cible et à travers les documents qui le situent.

Typographie et intertextualité

Ces idées, qu'on pourrait sans honte qualifier «d'inspiration interprétative», visent à remettre en place l'acteur qu'on a précipité cependant dans la trappe dans le spectacle typographique : le lecteur. Une «typographie interprétative» serait alors plus qu'une tendance, un engagement en faveur d'une reconsidération de l'objet typographique dans une perspective inversée, où il n'est pas pensé comme autonome, dissocié du sujet. Un recentrage de l'argument typographique non pas sur une lecture tout court, mais sur une lecture inscrite dans un projet d'interprétation (et plus avant, de compréhension). Elle me semble essentielle pour penser la typographie d'aujourd'hui, tiraillée entre le nomadisme pluri-culturel et les pressions de la globalisation. Pour réfléchir aussi, plus avant, sur le cours mais aussi sur l'orientation d'un modèle pour le document électronique. En particulier, pour le livre électronique.

La forme traditionnelle de la typographie, peut-être parce que fondamentalement liée au support qu'elle utilise, s'approprie fatalement de manière statique ce rapport à une société des livres. De l'autre côté, elle est foncièrement une typographie «mono-document». Sa conception fondatrice réside sur l'hypothèse d'un document (un livre en particulier), un style typographique, et un lecteur[8]. Elle est, en quelque sorte, «objectivante» (au sens qu'elle définit ses objets indépendamment des sujets, et souvent loin des pratiques réelles), en un sens, il s'agit d'une typographie «monothéiste et abstraite». La place du lecteur en tant qu'agent interprétatif opérant sur de la matière typographique est plutôt sous-évaluée, le plus souvent ramenée à des catégories générales, toutes abstraites. Parfois même, est-elle carrément supprimée. L'hypothèse récurrente est que le lecteur est en possession de pratiques conformes à une lecture standard (que la typographie sert à travers les modèles qu'elle adopte). Or, avec l'avènement du document électronique et la généralisation des pratiques d'écriture et de lecture, désormais à la possession de chacun, les normes d'écriture ont basculé. De lecture aussi. De compréhension, par conséquent, dans la mesure où les parcours sur lesquels la lecture se fonde pour ériger une interprétation ne sont plus les mêmes.

7. Bien sûr, avec les risques connus, afférents à l'utilisation par un novice d'une technique experte !

8. Peut-être pourrait-on aussi rajouter «une lecture».

Sous l'idée qu'un document (un livre) subsume une société de documents (livres), d'où il hérite cohérence interprétative et intelligibilité, il convient de s'interroger sur les conséquences de sa version électronique. Car — il n'est pas inutile de le répéter — cette «sociabilité documentaire» préfigure, précisément, un virage significatif dans la conception du document puisqu'elle redéfinit ses usages, suivant des pratiques de lecture modifiées. Et, finalement, les formes et les modes de son appropriation sémantique, point convergent de toutes les préoccupations qui commencent par la conception du contenu jusqu'à la distribution du livre.

Le document (le livre en particulier) n'est plus le même. Non pas parce qu'il a changé en soi. Non pas parce qu'il diffère sur un ensemble d'aspects physiques ou logiques. Mais parce que certaines de ses nouvelles fonctionnalités appellent à une rupture par rapport aux pratiques associées précédemment. Elles modifient, par conséquent, son rapport au sujet qui l'utilise. Donc, sa nature sémiotique. Un document (livre) qu'on commencera à lire différemment n'est plus le même document (livre). Car, lire différemment c'est conduire un autre projet d'interprétation. Donc, comprendre différemment. Et les choses (les documents et les livres en particulier) sont ce qu'on comprend d'eux (et sur eux).

Aux antipodes d'une vision logiciste du document (et du livre), qui font de lui un agrégat de traits structurés extraits d'un langage de description formelle, le document (le livre) est ce qui reste invariant à travers un ensemble de transformations d'usage[9]. Cet usage, du point de vue de son utilisateur ultime et en quelque sorte son «consommateur», est prioritairement fondé sur la lecture que ce dernier peut réaliser. Le document (le livre) qui ne reste pas invariant sous différentes lectures, *i.e.* sous différentes formes d'appropriation sémantique, n'est pas le même document. En d'autres termes, le livre est ce qui résulte d'un régime de lecture qui lui est adressé. À différentes lectures, correspondent, par conséquent, différentes conceptions du livre. La vraie, donc, et peut-être la seule question du document électronique est celle de la modification des pratiques de lecture. Une conclusion presque directe d'une conception «interprétative» de la notion de document (et de livre), mais qui a l'avantage de ne pas dissocier l'objet document (livre) de son rapport à l'homme et les pratiques qui lui sont réservées — pratiques socialement stabilisées.

Précisément, l'incorporation des rapports intertextuels suivant une telle conception semble une authentique voie dans la mutation du concept de document. Et partant, du livre.

9. Je reprends, relativisée, l'idée suivant laquelle un objet n'est pas une construction logique, mais l'invariant d'un ensemble de transformations : [15, 16, 17, 18].

Intertexte et typographie : l'âme même du futur livre

Discutons quelque peu, dans l'objectif de l'affiner, cette notion d'intertextualité, du point de vue de ses effets sémantiques. Autrement dit, du point de vue de ses apports sur le plan de l'interprétation. Et subséquemment, de la production de sens. En s'éloignant d'une aperception logique, qui consisterait tout simplement à envisager, à la place d'un document ou d'un livre, une classe de documents ou des livres, ainsi qu'en discutant de ses raisons et de ses incidences sur le plan des facultés du lecteur de faire du sens, nous pourrons mieux comprendre la tension que cette notion décidément «extra-typographique» exerce sur l'avenir du livre.

La notion de texte, matière empiriquement première de toute préoccupation typographique, semble généralement évidente. Il s'agit d'une sériation d'éléments linguistiques issus de divers niveaux d'analyse morpho-syntaxique, agrémentée par ailleurs d'un ensemble d'éléments de type structural. Dit plus naïvement, c'est des lettres et des mots, à la rigueur des phrases, tous mis côte à côte, suivant un certain ordre. Cette façon de voir, qui admet sans doute des précisions et des affinements à volonté, reste cependant compositionnelle, et définitivement logique. Elle n'est assurément pas fausse. Mais elle ne recouvre pas toute la vérité du texte, parce que limitée à son extériorité. Aux antipodes, une vision *top-down*, consisterait à voir le texte comme une sorte de sédimentation typographique d'une attitude interprétative globale. Cette vision, dont l'histoire est à rechercher aux techniques herméneutiques, rehausse cette autre nature du texte, faite d'éléments signifiants. Pour une telle vision, le texte n'est pas fait d'éléments de rang linguistique et formel inférieur, mais, précisément, de textes [8, 14]. Pour adhérer à une telle vision du texte, il est nécessaire d'avoir signé sous au moins deux principes, tous deux fondamentaux [7, 12] :

1. Le premier vise à exprimer une opposition au modèle de la communication hérité de la théorie du signal. Volontairement quantitatif, ce modèle suppose l'unicité de sens et la possibilité d'une communication entre l'auteur et le lecteur qui se ferait par la reconstruction du sens de l'auteur par le lecteur, au travers de ses lectures, à l'identique. Une fois devant le texte, l'acteur fondamental dans la génération du sens est le lecteur, et dont le rôle dans la facture sémantique n'est plus passif. Ce principe cherche à remettre en scène le lecteur et à restaurer son rôle et son importance dans la chaîne de production du sens. Il dévoile, par ailleurs, la raison du caractère sémantiquement ouvert de tout texte.
2. Le deuxième prône l'inversion du rapport habituel, entre constituant et constitué, en reprenant à son

> compte le thème herméneutique de la détermination du local par le global : le sens est toujours contraint par une donnée sémantiquement globale. Ce dernier concerne tous les niveaux de composition du texte[10].

L'intertexte est ce nécessaire premier et global, voulu et créé par un lecteur en besogne sémantique lors de sa lecture d'un texte, pour y induire précisément un sens possible, peut-être aussi, plausible (son propre sens). Il s'agit d'une forme textuelle d'une fraction du contexte[11] ou, mieux, une rationalisation sous forme textuelle de quelques aspects contextuels, somme toute réglés par un dessein interprétatif. Ce n'est pas une encyclopédie ni un thésaurus, même si localement il peut présenter des affinités opératoires. Il ne s'identifie pas non plus complètement à la notion de corpus, dont il reprend une quotité pour supporter un faisceau de parcours de lecture.

L'intertexte est, au fond, *une qualité*. Construire un intertexte c'est, en réalité, (se) donner *une matrice de compétence interprétative*. Ainsi que le cadre de la validité de son interprétation (donc, de sa compréhension). C'est à l'intertexte que l'on a recours pour légitimer le rapport entre sa compétence et sa performance interprétatives. C'est, en quelque sorte, une tentative pour poser les repères et les conditions de fonctionnement d'un programme d'interprétation. L'intertexte réalise et achève le texte tout comme le groupe complète le contenu de l'individu. Il montre que texte ne peut être clos en lui-même, et que ce qui est autour de lui formant sa société, est autant signifiant que ce qui est à l'intérieur de lui. Seul, le texte, est en réalité sémantiquement indécidable. D'autre part, le rapport entre texte et intertexte devient aussi explicatif. En effet, en plongeant un texte dans son intertexte, on lui intègre les principes de sa propre explication. L'intertexte devient, au fond, le support d'une présomption sur l'auteur construite par le lecteur : en essayant d'interpréter un texte au sein d'un intertexte que nous construisons, nous sous-entendons que son auteur évolue dans l'espace des idées que nous pensons retrouver dans cet intertexte.

Rendre opératoire, donc, l'intertexte comme une forme tangible de l'authentique nature du texte, *i.e.* du texte du point de vue sémantique, c'est parler, déjà, d'un autre livre. C'est, aussi, opter déjà pour d'autres formes typographiques, puisqu'on passe d'une représentation mono- à une représentation multi-livres. C'est vraisemblablement ce dernier qui complètera le progrès de la page : non pas la sériation des pages, mais la constitution d'un réseau de pages appartenant à des textes différents.

En deux mots, le livre à venir ne sera pas, à mon sens, un amas de pages ou quelque chose qui matérialise un écran à dérouler. Ce sera le correspondant d'un réseau, de facture somme toute intertextuelle. La typographie qui l'accompagnera le suivra aussi. Elle complètera ses catégories classiques avec des catégories proprement interprétatives : *intra-typographiques* (explicitant les rapports des parties du texte entre elles) et *inter-typographiques* (faisant référence aux rapports entre les parties du texte avec (des parties) d'autres textes).

Il n'y aura pas seulement dématérialisation du support et représentation dynamique de la page, mais aussi de l'objet livre. Une dématérialisation qui s'annonce déjà par l'introduction de la valeur intertextuelle dans les liens des documents électroniques. Non pas seulement un détachement du support papier, mais aussi de la vision mono-documentaire et objectale, voici l'horizon d'un livre au service de l'interprétation. D'un livre enfin centré au lecteur et ses besoins de lecture.

C'est, à mon sens, le schéma du livre qui frappe à la porte des NTIC. Quelqu'un ira-t-il ouvrir ?

Petit épilogue

La représentation d'une nouveauté se fait généralement avec la donation d'un nouvel objet, du mode de sa correspondance avec l'ancien, souvent l'original, mais aussi de la valorisation de sa lecture. Lorsque le nouvel objet est seulement imaginé, les conditions de l'imagination l'emportent sur le mode la correspondance et de la lecture. Aujourd'hui, ces conditions, parce que peut-être plus lisibles, sont majoritairement portées par les aspects technologiques. Ainsi, le livre de demain continue à être matérialisé de la même manière. La conception de sa typographie le suit avec des changements timides.

Si les logiciels nobles (dont le grand-père TEX) ambitionnent de réhabiliter le sens de la typographie à une époque où la vulgarisation apparaît comme la fille unique de la mondialisation, si ces aristocrates d'antan et d'aujourd'hui souhaitent ne pas perdre le train du livre électronique, ils doivent sans doute aligner leurs prétentions et leurs fantasmes. En admettant qu'ils le doivent, on peut se demander s'ils le peuvent. Mon opinion personnelle est que la bataille finale devient d'un jour à l'autre inégale. Pour des raisons de fondation et d'objectif, essentiellement. La confrontation entre une vision mono-livre et une vision multi-livres n'est pas une confrontation uniquement technique ou seulement quantitative. Elle pointe vers une différence de nature. Mon opinion personnelle est, donc, qu'à l'ère du livre électronique, ces logiciels reculeront. Au moins fonctionnelle-

10. Il constitue, en un sens, la reprise interprétative d'un principe cognitif général, suivant lequel il n'y a pas de détermination et d'identification sans un acte préalable de localisation, de mise en contexte, en quelque sorte.

11. Bien entendu, le contexte, généralement, ne se réduit pas à la seule donnée d'un intertexte, qui ne fait qu'en énoncer une part textuellement opératoire.

ment. Sans doute, feront-ils toujours autorité sur un ensemble de principes salubres concernant le traitement du texte électroniquement, respectueux des traditions typographiques — principes qu'il ne faut pas délaisser. Mais ils ne pourront, tels quels, répondre à l'exigence interprétative, qui viendra se superposer sur eux.

Quel candidat pour leur suite, donc ? Peut-être une idée comme celle des *vario-documents* [5], concept encore embryonnaire mais intéressant, à mi-chemin entre l'exigence du respect typographique et l'adaptation au profil de l'utilisateur. En supposant (en espérant plutôt) qu'ils arrivent à penser le livre comme intertexte et les modes de réhabilitation non seulement typographique mais aussi sémantique des préférences du lecteur. Ou, peut-être encore, une synthèse de quelques méthodes typographiques standards dans un environnement de travail collaboratif. Car — je me permets de le répéter pour conclure — la mutation des pratiques de lecture ne sera pas seulement au niveau du symbolique et du typographique. Elle sera essentiellement située, toujours à mon humble avis, sur le registre de la confection sémantique. Même avec un livre électronique à la main, on lira toujours pour comprendre.

Références

[1] U. Eco. *A Theory of Semiotics*. Indiana University Press, 1976.

[2] U. Eco. *The Role of the Reader — Explorations in the Semiotics of the Text*. Indiana University Press, 1979.

[3] U. Eco. *Lector in fabula*. Grasset et Fasquelle, 1985.

[4] U. Eco. *Sémiotique et philosophie du langage*. PUF, 1988.

[5] Yannis Haralambous and Johan Nonat. Un prototype de lecteur de vario-document. In *Comptes-rendus du colloque DVP*, Brest, 2002.

[6] I. Kanellos. À propos de l'héritage critique du document électronique : multimodalité sémiotique, stratégies de lecture et intertextualité. In *CIDE'99 : deuxième colloque international sur le document électronique*, pages 97–109, Damas, 5–7 juillet 1999.

[7] I. Kanellos. De la vie sociale du texte. L'intertexte comme facteur de la coopération interprétative. *Cahiers de praxématique*, 33 : 41–82, 1999.

[8] F. Rastier. *Sémantique interprétative*. PUF, Paris, 1987.

[9] F. Rastier. *Sens et textualité*. Hachette, Paris, 1989.

[10] F. Rastier. La triade sémiotique, le trivium et la sémantique linguistique. *Nouveaux actes sémiotiques*, 9, 1990.

[11] F. Rastier. *Sémantique et recherches cognitives*. Philosophie d'aujourd'hui. PUF, Paris, 1991.

[12] F. Rastier. Communication ou transmission ? *Césure*, 8 : 151–195, 1995.

[13] P. Ricœur. *Le conflit des Interprétations—Essais d'herméneutique*. L'ordre philosophique. Seuil, 1969.

[14] P. Ricœur. Qu'est-ce qu'un texte ? In R. Bubner *et alii*, dir., *Hermeneutik und Dialektik*, volume 2, pages 184–221. Lawrence Erlbaum Assoc., Tübigen, Mohr, 1970.

[15] R. Thom. *Stabilité structurelle et morphogénèse*. InterEditions, Paris, 1977. Première parution aux éditions Benjamin, 1972.

[16] R. Thom. *Paraboles et catastrophes*. Champs. Flammarion, Paris, 1983.

[17] R. Thom. *Esquisse d'une sémiophysique*. InterEditions, Paris, 1988.

[18] R. Thom. *Prédire n'est pas expliquer*. Champs. Flammarion, Paris, 1993.

Nouveaux signes de lecture et d'écriture pour les documents électroniques

Ghassan Mourad
Laboratoire LaLICC (Langage, Logique, Informatique, Cognition et Communication, UMR 8139)
96, bd Raspail, 75006 Paris, France
Ghassan.Mourad@paris4.sorbonne.fr
http://www.lalic.paris4.sorbonne.fr/~mourad

Abstract

This paper examines the link between knowledge representation and new technologies, and the integration of new signs into typography. Knowledge representation evolves with the evolution of technology. Text manipulation strategies, multimedia, mobile telephones and SMS are not just witnesses to this evolution; they also show the large number of users and the needs that they generate. With respect to typography, we cannot ignore the "new signs" that appear in our daily lives. Each technological revolution provokes new behaviors. Just as printing influenced typographical signs and reading, the new technologies provoke changes in the same field. These innovations have given birth to new methods of knowledge representation. Moving from print to the cathode ray tube (from paper to screen), from reading on paper to reading on screen — these have instilled a need to create new signs or adapt existing ones.

Résumé

Dans la présente communication, je m'interroge sur la représentation des connaissances liée à l'inovation et les nouvelles technologies et sur l'intégration dans le système typographique de nouveaux signes. Concernant la typographie, nous ne pouvons pas ignorer ces «signes nouveaux» qui tendent à s'inscrire dans un contexte quotidien. Chaque révolution technologique engendre de la sorte de nouveaux comportements. Telle l'imprimerie et ses influences notamment sur les signes typographiques et la lecture, les nouvelles technologies entraînent, bien sûr, des changements sur ces mêmes terrains. La description des signes dans cette perspective concerne l'index pointé et certains signes de navigation hypetextuelle.

À tout problème que pose la transcription de la pensée, la typographie se doit d'apporter au moins une solution ; elle en offre plusieurs dès qu'on la sollicite de faire valoir des nuances ou des subtilités.
(Ch. Gouriou)

La main «pointant l'index» est utilisée métaphoriquement comme dans ce tableau[1] de Raphaël (*Platon et Aristote*).

Alors que Platon, qui a les traits de Léonard de Vinci, tient son index pointé vers le ciel, son élève, Aristote, a la paume de la main tournée vers le sol. C'est ainsi que Raphaël oppose les préoccupations métaphysiques de Platon à la méthode scientifique prônée par Aristote.

Introduction

La représentation des connaissances est en évolution par rapport à l'évolution technologique. Les techniques de manipulation de textes, le multimédia, le téléphone portable et les SMS (Short Message Service), témoignent non seulement de ces évolutions mais également de l'utilisation d'un très grand nombre d'utilisateurs et du besoin qui peut être provoqué chez eux.

Ces innovations ont donné naissance à de nouveaux moyens de représentation des connaissances. Le passage de l'imprimé au cathodique, du codex au volumen, de la lecture sur papier à la lecture sur écran, etc. ont donc entraîné, notamment, le besoin de créer et d'adapter de nouveaux signes.

En effet, c'est le cas des signes typographiques «virtuels» qui sont pour certains une adaptation de signes typographiques déjà existants. Exemples, l'usage massif de signes de ponctuation dans l'écriture des *e-mails* exprime des sentiments ; l'index pointé qui, après diverses évolutions sur lesquelles nous allons revenir, signifie souvent dans les imprimés modernes «tourner la page» a pris la signification de «cliquer» ; le soulignement bleu et violet constituent désormais des signes typographiques à part entière.

Parce que nous sommes dans le contexte des textes électroniques, je voudrai illustrer les transformations

1. http://www.unesco.org/phiweb/fr/raphael/

Fig. 1 : Image de Platon et Aristote.

liées à la typographie. Evidemment, cette illustration concerne seulement l'index pointé et les soulignements graphiques de liens hypertexte, mais il existe beaucoup d'autres questions liées aux représentations des connaissances et à leur traitement, tant du point de vue de la lecture que de l'écriture.

Nous insistons sur le fait que ces signes relèvent du champ sémantico-énonciatif qu'il faut prendre en compte lors de l'écriture et de la lecture. Ils définissent des relations discursives de hiérarchisations et de structuration du discours. Ces signes ouvrent une nouvelle réflexion sur la description de la langue et de sa représentation, voire de sa compréhension.

L'index pointé

Historique L'index pointé est un signe graphique, qui selon Drillon aurait été utilisé par Dolet au XVI^{e} siècle (1540) pour désigner la partie d'un document. Cependant, dans son ouvrage «La signature entre la lettre et l'image», Fraenkel signale un usage plus ancien de ce signe qui aurait été utilisé par des scribes pour corriger les brouillons préparatoires de textes juridiques comme en témoigne la figure 2 (empruntée à [5, p. 151]).

En effet, nous pouvons affirmer avec Fraenkel que ce signe était déjà employé comme un signe technique. Sa présence dans un manuscrit servait à indiquer l'endroit d'une erreur et la nécessité de la corriger. C'est donc un signe qui fait référence. Dès cette époque il avait un caractère «conventionnel», car seule la communauté «scribale» en connaissait la valeur. Ce signe pouvait également

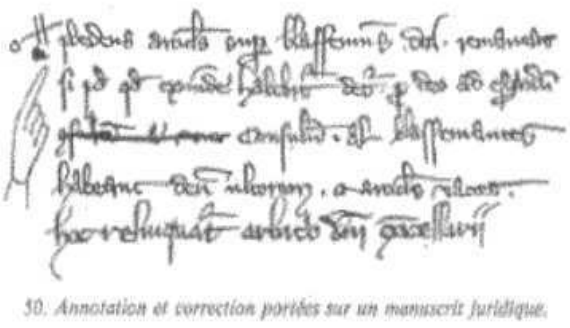


Fig. 2 : Annotation sur un manuscrit juridique.

être «employé comme seing» et «est donc une sorte de clin d'œil à la caste des scribes».

Ainsi, nous pouvons nous demander si cette indication du doigt n'était pas finalement une forme de mise en page utilisée pour montrer le paragraphe dans lequel le scribe est intervenu !

Par ailleurs, ce même signe aurait été utilisé dès le XIII^{e} siècle par les notaires en tant qu'indicateur. Il faisait référence selon Fraenkel à «celui qui l'a tracé [...] Utilisé comme signature, il exprime l'autorité de celui qui sait, ou encore traduit la volonté de celui qui ordonne» [5].

Cependant nous pouvons avancer une autre interprétation qui consisterait à dire que ce signe servait à désigner l'endroit où l'on devait apposer sa signature. Quoi qu'il en soit, l'index pointé a une «fonction monstrative». La fig. 3 presente quelques exemples de signatures (emprunti à [5, p. 150]).

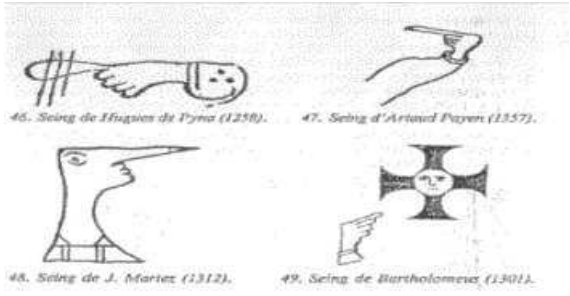


Fig. 3 : Différentes formes de signatures.

Ce signe du fait de sa fonction gestuelle[2] est devenu le symbole indicateur par excellence. Il permet notamment d'éveiller l'intérêt. Ainsi, il sert à indiquer une direction («par là»), il a le sens d'un impératif (comme dans

2. Dans certaines cultures on désigne sa propre personne en pointant l'index sur le nez (Japon), dans d'autres sur le cœur. En langue des signes au Liban, l'index pointé sur la tempe, symbolise la liberté (pour dire «je suis libre»).

certaines cultures où l'on «parle avec les mains»). Il exprime une mise en garde, une menace («attention !»). Dans ces cas c'est un signe gestuel qui peut accompagner la parole.

L'index pointé actuellement Actuellement, dans certains types d'ouvrages imprimés on a recours à l'index pointé (fig. 4) pour renvoyer par exemple à la page suivante, à une autre page, à un paragraphe. Quelques fois ce signe est accompagné de «flèche», «voir ici», «aller à», etc. Cet index pointé est également utilisé dans les documents électroniques ; il a les mêmes significations que dans les documents imprimés, il en a aussi la même représentation. En effet, c'est une image fixe de l'index qui existe sur l'écran et sur la page. Il peut être accompagné d'un hyperlien pour renvoyer à une autre page.

Fig. 4 : Index pointé 1.

L'index pointé de la fig. 4 a les mêmes fonctions que le premier. Mais à la différence de celui-ci, son image n'est pas fixe, et il ne fait pas partie du corps de la page, ni de son contenu. Ce n'est que l'image virtuelle de l'index[3].

Fig. 5 : Image virtuelle.

L'index pointé virtuel L'index pointé virtuel représente l'image de la main posée sur la souris. Il indique qu'on doit cliquer, en nous montrant le geste.

Cet index pointé permet de faire le lien entre deux messages linguistiques et est lui-même un message linguistique : «aller là», «tourner la page», «voir là bas», «cliquer ici», etc.

Ce signe est universel, c'est-à-dire que tout le monde est capable de le comprendre sans connaître la langue du texte. Il est vrai que lorsque l'on effectue des recherches sur l'Internet, il peut arriver que la langue des pages chargées soit inconnue. Le seul signe reconnaissable, identifiable (s'il est présent) est alors l'index pointé sur lequel on clique directement : en d'autres termes c'est un signe de monstration.

Cet index joue un rôle majeur pour la lecture hypertextuelle. C'est un marqueur de connexion entre les unités d'information, il unit sémantiquement deux segments qui sont physiquement séparés.

L'index pointé est un Signe L'index pointé est un signe iconique, il possède toutes les caractéristiques du signe chez Pierce. C'est un symbole : d'une manière conventionnelle, il symbolise l'accès au lieu et vers l'objet sur lequel il pointe (objet virtuel, mais qui existe réellement en tant que code physique). Il peut être vu comme une icône : dans le sens où il représente l'action CLIQUER ; et comme l'indice d'un changement d'une situation à une autre. Le fait de cliquer va changer la représentation de l'espace et du temps.

D'autres signes typographiques liés à l'hypertexte

Le bleu Nous définissons dans cette catégorie le bleu et le violet que nous considérons comme des signes typographiques liés aux nouvelles technologies.

Les marquages typographiques ou la mise en relief de certains segments textuels mettent en évidence les expressions que l'on veut différencier des autres segments. Un segment textuel souligné graphiquement en bleu est un lien actif qui désigne que derrière cette couleur il y a une autre information. Ce signe représente des marques d'énonciation, autrement dit, un discours sur le discours. C'est le signe par excellence de l'hypertextualité. Ainsi, la couleur ne désigne plus la couleur en tant que telle, mais un lien hypertexte. Aussi, l'objet auquel il réfère existe réellement : c'est un code informatique.

C'est un signe selon la définition de signe de Pierce. Ainsi comme nous le savons, le sens se définit par une relation triadique entre le signe, son objet et son interprétant. Le bleu est un indice qui a une relation avec son objet : qui est mis en relief (le segment bleu) : c'est un signe linguistique ; il est lié à l'objet auquel il réfère ; c'est un signe symbole.

Le violet Le violet est aussi un signe de lien hypertextuel qui indique que le lien a déjà été activé. Lorsque le pointeur (en général une flèche)[4] de la souris passe sur un hyperlien (bleu), la flèche devient soudainement un index pointé (c'est-à-dire que l'adresse de la page est disponible) ; il suffit de mimer la forme de l'index pour que l'adresse soit activée et que la couleur du lien soit changée en violet. Le violet sert donc à indiquer que la page à laquelle il se réfère a déjà été «visitée».

3. Il existe plusieurs formes d'index pointés, mais nous nous intéressons seulement à celui de la fig. 4.

4. Nous considérons aussi que la flèche «baladeuse» est un signe de «ponctuation virtuel» qui a le rôle de pointeur et qui indique la phase préparatoire de l'action «cliquer».

Conclusion

Nous avons voulu montrer brièvement que l'émergence des nouvelles technologies a changé notre façon de travailler et de nous exprimer. Elles ont donc une influence sur notre façon de représenter les connaissances et de les traiter, notamment par le biais des nouveaux signes que sont l'index pointé, le soulignement, le bleu et le violet. Nous signalons que la plupart de ces signes ne sont pas nouveaux mais sont plutôt une adaptation aux nouvelles technologies. Par exemple, si les guillemets désignent la présence d'un texte dans un autre texte, nous pouvons actuellement le présenter grâce à un lien. Cette présentation nécessite de nouveaux signes comme ceux cités plus haut. Ces signes doivent obéir, plus ou moins, pour leurs manipulations, à des règles connues par tous. Le lecteur peut voir un exemple sur la fig. 6.

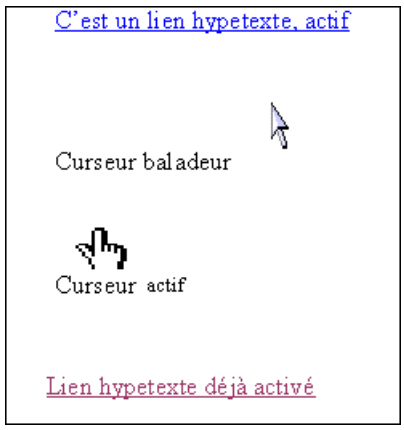


Fig. 6 : Signes de navigation.

Ces règles sont parfois des processus comme le processus de changement lié au langage iconique, dont la compréhension nous conduit à prendre en compte la représentation sensorielle, notamment visuelle (bleu → flèche → index pointé → violet). Ce phénomène est engendré par ces icônes et entraîne la production d'une action. Ces signes représentent le champ sémantique lié aux concepts «remarquer», «montrer», «indiquer» et «déceler».

Enfin nous insistons sur le fait que ce sont des signes qui relèvent du champ sémantico-énonciatif qu'il faut prendre en compte lors de l'écriture. Ils définissent des relations discursives de hiérarchisation et de structuration du discours. Ces signes ouvrent une nouvelle réflexion sur la ponctuation et sur la typographie ainsi que sur la description de la langue et de sa représentation voire de sa compréhension.

Ces formes ne sont pas rigides, mais elles constituent une sorte de «norme» et sont actuellement utilisées par défaut.

Références

[1] Catach, N., *La Ponctuation*, Presses Universitaires de France, Paris, 1994.

[2] Defays, J.-M., *et al.*, «À qui appartient la ponctuation ?», Actes du colloque international et interdisciplinaire de Liège (13–15 mars 1997), dans *Champs linguistique*, Paris, Bruxelles, p. 437–447, 1998.

[3] Dolet, E., *La Ponctuation de la langue francoise*, Lyon, 1540. Rééd. Obsidiane, dans *La manière de bien traduire d'une langue en aultre*, 1990.

[4] Drillon, J., *Traité de la ponctuation française*, Gallimard, collection «Tel», 1991.

[5] Fraenkel, B., *La Signature, genèse d'un signe*, nrf, Gallimard, 1992.

[6] Gouriou, Ch., *Mémento typographique*. Éditions du Cercle de la librairie, Paris, 1990.

[7] Hubert, M., *Le Vocabulaire de la ponctuation aux temps médiévaux*, Archivum Latinitatis Medii Aevi, 28, p. 57–166, 1972.

[8] Huchon, M., «Les Divergences de ponctuation dans les éditions de Rabelais de 1546 à 1553», dans Nina Catach (éd.), *Punctuation* II, p. 123–139, 1979.

[9] *Lexique des règles typographiques en usage à l'imprimerie Nationale*. Imprimerie Nationale, Paris, 1990.

[10] Mourad, G., «La Ségmentation de textes par l'étude de la ponctuation», Acte de *CIDE*'99, *Document Électronique Dynamique*, p. 155–171, Damas, Syrie, 1999.

[11] Numberg, G., *The linguistics of punctuation*, CSLI Publications, Stanford, CA, 1990.

[12] Orlandi, E. P., *Les Formes du silence dans le mouvement du sens*, éd. Des Centres, Paris, 1996.

The XEmTeX Project

Marie-Louise Chaix
EDP Sciences, Paris
mlchaix@edpsciences.com
http://www.fptex.org/xemtex/

Fabrice Popineau
Supélec, Metz
fabrice.popineau@supelec.fr
http://www.fptex.org/xemtex/

Abstract

We are presenting the XEmTeX project which is funded by the French government under the RNTL program (National Network on Software Technologies). The purpose of the project is to provide a high-quality, user-friendly, free integrated typesetting platform to typeset scientific or technical documents.

Résumé

Nous présentons ici le projet XEmTeX qui a été retenu pour être financé par le gouvernement français dans le cadre du Réseau National des Technologies Logicielles. Ce projet a pour objectif de fournir une plateforme typographique intégrée, libre, facile d'accès et de haute qualité pour la mise en page de documents scientifiques et techniques.

Why XEmTeX?

TeX has been available and used by many people for 25 years by now, so it must have some strength and it must outperform its competitors. But there are not so many TeX vendors. It is not mainstream among desktop publishing software or text processing software for several reasons:

- TeX is a programming language, which makes the learning curve much steeper,
- TeX is useful for complex typesetting jobs, with lots of mathematics, or complex typography, but many people have jobs that do not require the complexity of TeX,
- TeX source code is free since it has been given to the community by its author, D.E. Knuth. There are not so many commercial, supported versions of TeX; it is essentially used in the academic community, in its free form.

TeX distributions have been floating around for free for a very long time. They used to be difficult to install and to maintain. Much progress has been made in the recent years.

One of the very first complete and well-designed distributions of TeX was named emTeX (from its author, Eberhard Mattes) and targeted MS-DOS. This distribution of TeX had the advantage of fitting on only a few floppy disks. It was solid, well integrated with the environment, and had a lot of success.

Then teTeX brought Unix users a complete and sound set of programs and style files, much more extensive than the set provided with emTeX. Since 1994, Thomas Esser has done a great job in smoothing over the installation of TeX, especially if we consider that in the tradition of Unix programs, you get the sources and it is up to you to compile and install everything.[1] Thomas Esser's distribution is based on the Web2C sources of TeX maintained by Karl Berry and now by Olaf Weber, and it is completed by several scripts and programs helping a lot with use, configuration and maintenance of TeX.

Windows users have different requirements than Unix users. The level of technical knowledge is not generally the same. For example, Windows users do not expect to interact with the computer through the command line — don't even think of making them compile their programs! Windows and MacOS advertise their graphical user interface. Unlike MacOS (up to version 9), Windows has a builtin command interpreter, even quite a fancy one under the NT/2K/XP flavours of Windows, but although it exists, only a few users know how to use it. TeX being a compiler which is invoked through the command line, Windows and MacOS users expect it to

1. Things have changed only recently with the advent of Linux distributions and the availability of precompiled packages.

be wrapped up in a GUI application. Dedicated text editors have been devised to this effect, other ones have been adapted to it.

MiKTeX and fpTeX are the main free versions of TeX for Windows. MiKTeX is an entirely new port of TeX to Windows, whereas fpTeX is based on the very same sources as teTeX and Web2C. These distributions are now much larger than emtex: they do not fit on floppies, but on CD-ROMs! In the meantime, the TeX community has developed lots of new tools and macro packages, which explains the inflation.

During all these years, another project was born to deliver a full, ready-to-run version of TeX on CD-ROM: the TeX Live project. This project aimed at being the most complete set of TeX programs, macro packages and style files, or related tools. But the novelty was that you could run the tools from the CD-ROM without installing anything. Moreover, almost all frequently used architectures were available: several Unixes, Windows, MS-DOS and Amiga up to some point, Mac OS X nowadays.

But all these distributions target only the TeX programs and related tools. They do not target the full environment needed to actually typeset documents. To our knowledge, only the 4AllTeX project had this ambition in the past. The 4AllTeX CD-ROMs brought you a complete environment under Windows to type in, compile and print your documents. Everything was included, from the editor to the printer drivers, and including tools to handle images or to draw graphs, a spell checker, etc. The wonderful idea was to write a set of *wizards* that assisted you in your most common tasks. Everything was configurable, so that you could choose your text editor or your TeX engine and so on. The result was a nice environment, really easy to work with. Unfortunately, this project stopped circa 1999.

Having worked on the TeX Live project for many years, trying to bring Windows users with a comprehensive, up-to-date and sound TeX distribution, we wondered how to turn this distribution into something closer to a Windows application than has been the case till now. Windows users like Microsoft Word (or any other DTP program) because when it is installed, there is only one icon on the desktop to click, and all functionality is available from there. The application is compact and the interaction is clear. Obviously, TeX being much older and free software, it cannot compete with such a facility of use. However, it seemed that it should be possible to reduce the gap between both kinds of programs.

There are free text editors that are both well integrated with their environment and with cross-platform availability. The most well-known freely available text editor is probably GNU Emacs. But we would not qualify it as well integrated with its environment. Using GNU Emacs under Windows may seem a bit strange to Windows users, mostly because it does not take advantage of native controls (native toolbars or other common controls). On the other hand, there is extensive support for TeX documents inside Emacs. Fortunately, there is an alternative to GNUEmacs: XEmacs is a forked project from GNU Emacs 18 series. It started as Lucid Emacs, and then became XEmacs. The difference between the emacsen is mostly in the internals and the external aspects, but they share many features and a lot of the "feel" part of the look-and-feel. The look part is much more in keeping with the environment under XEmacs, especially if we consider the Windows native version and the GTK version.

So the big idea came that XEmacs could act as an integrated environment for writing, compiling and printing TeX documents. Well, this is not so big an idea for experienced computer users, but the fundamental concept is that this environment could be distributed to less experienced computer users. Let's reserve some of the details for the moment. The main point is that the user could access his typesetting environment just by clicking on an icon on the desktop, as is the case for most DTP programs, and that's a big selling point in our opinion. TeX and all the machinery would remain hidden behind the scene.

Free software and funding

Not everyone is interested in TeX, and TeX is not the right answer for everyone. However, for certain types of jobs or types of documents, TeX has no competitors. Education is a target for TeX, for several reasons:

- obviously for mathematics and science teachers, because of the high quality result in typesetting maths,
- stability and long lasting documents: TeX has been available for 25 years and the input language does not change with each version like most commercial products,
- it is free and available on all platforms.

For a few years, there has been a growing demand from teachers — especially from high-schools — towards GUTenberg for hints on how to start with TeX. Part of this demand is due to the growing popularity of free operating systems like Linux. GUTenberg had a project to build a dedicated CD-ROM and distribute it through the national education system in every high-school in France. However, this was a free project, and like many free projects, it was to be developed in the free time of people who were already busy, so it came to nothing.

As we wanted to ground this project a bit more than the average free software project, the question arose about the viability of the project and how to fund it. At the same time, the French government brought us an

opportunity though the so-called National Network on Software Technologies. In short, projects eligible to be funded must involve at least one company and one academic institution. Projects can fall in one of several categories:

1. pre-competitive projects, which should lead to final products at the end of the project,
2. research projects to develop new technologies or new algorithms.

The XEmTeX project did not fall into either of these categories in any obvious way. However, the RNTL allowed for a third kind of projects to be presented: free software projects. They even released an economic study of the various business models for free software distribution with the clear intention to support some free software projects.

So this was the opportunity we were looking for, and partners to develop and submit the project were quickly found:

- Fabrice Popineau teaches at Supelec (`http://www.supelec.fr`), an academic institution in the field of electrical engineering. He has been working on the TeX Live project for several years,
- Marie-Louise Chaix is Project Leader at EDP Sciences (`http://www.edpsciences.org`), a French publisher specializing in scientific journals, books and electronic publishing. In her company, many people are potential professional users of the resulting product and will bring their desires and expertise into its design.

We submitted the XEmTeX project with the following arguments.

First of all, there are very few commercial DTP and text processing programs that are very widely spread, the most common being MSWord. So any new product has little chance to gain an audience. But in the particular domain of scientific and technical documents, those widespread commercial programs have poor performance. If you want high quality typesetting, then you must resort to TeX. It will provide the required quality in whatever notation system you need: maths, physics, languages from all around the world, music, etc. Being programmable, TeX has been able to adapt to new technologies like PDF and HTML.

When it comes to text editors, you can't avoid speaking about the Emacs family. The emacsen stability and versatility is well-known. As they are programmable, they can be tailored to anyobdy's use. Even if the standard way to interact with an emacs is difficult to learn, and maybe not that intuitive today (key sequences), it is possible to reprogram this and turn it into something more suited to our needs.

So all the pieces exist, free of proprietary rights, to assemble a high quality typesetting platform that would outperform many commercial programs for our kind of documents — and that is the first goal of the project.

The second goal of the project is to widen the TeX audience by providing people with an integrated platform, easy to use even for novice users. As a matter of fact, for many people, TeX is more difficult to install and maintain than it is to use. Typing in some LaTeX document can be explained quite easily and it does not require any system administrator skill. It is not the same when it comes to actually putting the TeX software or any of the related tools needed on the machine. Any TeX distribution is made up of thousands of files, and conflicts can arise easily. Many get frustrated when they fail to install it at first try, and it gets even worse if they actually spend time trying to fix problems, unless they manage to do it. By using XEmacs as an integrator, we can isolate our TeX system from the rest of the software installed on the machine, and the user will more likely get a working program. So we claim that if TeX was as easy to install as other programs are, TeX would have more users.

The project will rely on software components already available from the community and any new developments will be made available to the community free of rights. In the end, the XEmTeX project will be integrated to the TeX Live project. The XEmTeX CD-ROM resulting from the project will be sent to all high-school teachers (mathematics and science) by the GUTenberg association.

The road to XEmTeX

Framework All the needed tools to build a complete environment are freely available. TeX Live under Windows used to try to help people with installing the most frequently needed tools:

- a text editor to select among half a dozen (WinShell, WinEdt, etc.)
- the Ghostscript PostScript interpreter, either in its free or non-free version,
- image files tools like NetPBM or ImageMagick,
- the ISpell spell-checker with dictionaries,
- Perl because many scripts use it and it is not available by default under Windows.

But this list of supplementary programs was difficult to maintain, mainly because these products were not repackaged and their installation procedure kept changing. Also, they were not mandatory, and their installation was not delegated to the TeX Live system: they installed as standalone products, at the risk of conflicting with other versions.

Anyway, the starting point of the project is a subset of the TeX Live 7 CD-ROM, augmented with XEmacs

and the packages relevant to TEX typesetting, and the supplementary tools cited above. The goal of the project is by no means to write lots of new programs, but rather to put glue between existing programs and package the result so that it is easy to use for anybody. Relying on several large, complex products like TEX, XEmacs, ..., we are forced to play safe with updates to these products. That means we can't patch them heavily to suit our needs, except if we can make sure the patches will find their way in the main distribution. That is why it is better to use the glue strategy than the patch strategy.

For its TEX part, XEmTEX will be a proper subset of TEX Live: only the most commonly used packages, as few binaries as possible, nothing related to bitmap PK fonts. The other parts have already been named: XEmacs for integrating everything, Ghostscript, ImageMagick and NetPBM.

The project has been split into several tasks which are described below. The goal is to get progressively closer to the final XEmTEX product.

Installer Lots of work has already been done in the TEX Live project regarding installation, especially about specification of packages and writing so-called TPM files. XEmTEX should be very simple to install, hence should offer as few options as possible to the user, contrary to TEX Live, which offers lots of options to the user.

TEX Live installers for Windows and for Unix are very different: it is an application with a graphical user interface under Windows and a shell script running in text mode under Unix. It has been unclear for a long time if a portable installer was feasible and desirable.

Given that the XEmTEX installer should not offer many options, we may think of something much simpler for both cases. However, it appears that those platforms are really different in several system aspects — file associations, icons, menus and so on — so that it seems difficult to have only one installer. More precisely, each platform offers services to install applications, and these services are different. So the answer is that a native installer should be built on each platform. Given our targets, that means a Debian package will be built under Debian Linux, and a MSI (Microsoft Installer) package will be built under Windows.

Editing text The starting point here is XEmacs and the AUC-TEX and Preview-LATEX packages. As powerful as XEmacs may be, some aspects of the user interaction may seem a bit strange or uneasy to novice users — think about dialog through the minibuffer, or complex key sequences for example. This is partly due to the fact that emacsen have been used mainly by programmers that understand this way of thinking, and partly by the fact emacsen can run in console mode, and no effort has been made to build a simplified user interaction mode. Emacsen are complex tools for complex jobs. But it would be a shame if the most powerful text editor could not be tailored to suit the needs of less experienced people. So this part of the project will tackle the problem of ergonomics and definition of menus, toolbars and keymaps suitable for our goal. Following the general philosophy, we won't rewrite big parts of existing stuff, but rather tailor and wrap up the existing tools.

As an exception to this rule and as a specific subpart, an equation editor has been considered. Given our targeted audience, it could be a fancy and useful tool to provide people with. A first sketch has been written in wxPython to be portable. If it appears that the result is stable enough under both platforms, it will be integrated into the XEmTEX product.

Apart from typing in TEX documents, the user will have to handle other kinds of files: images and graphics. The related tasks will also be identified, and the common ones will be offered through menus and toolbars. For example, MetaPost is a nice tool to draw graphics and support should be provided for it inside XEmacs.

Viewing documents TEX is not a WYSIWYG tool, which is a bit disturbing for people who don't know it yet. Specific attention will be paid to coupling the compiler and the viewer. Up to now, DVI and PDF have been the output formats considered. The DVI format has the advantage to be fast to display on screen, but the drawback not to be self-contained: fonts, images are stored externally. For the PDF format, it is the opposite.

The status of viewers available on TEX Live is not the same under Windows and under Unix. The XDvi viewer for Unix is able to display Type 1 fonts and to use source specials to map a location in the viewer to the same place in the source file. Windvi available for Windows does not yet have those features.[2] However, we can wonder if the DVI format should be advertised as the format of choice for new users, especially if it were possible to view with the PDF format and the same source specials mechanism feature. Heiko Oberdiek has implemented such a mechanism in his vpe package. What needs to be investigated is: can it be reasonably used? Is it fast, is it convenient? If yes, no doubt the PDF format should be advertised over the DVI format.

What we want to avoid at any price is the use of bitmap font files. The mechanism to build these fonts is too complex, too error prone. Nowadays, with scalable fonts, we can completely avoid using bitmap fonts, and thus avoid installing METAFONTand all the programs calling it, like `mktexpk`. In any case, the Windvi viewer will eventually be brought up to the same level of features as the XDvi viewer.

2. The source special mechanism is available, but not documented.

Documentation Documentation is the weak point about TeX: there is too much of it, and it is difficult for a new user to get started. So a specific effort will be devoted to write a clear set of documentation to bring people in the game. We won't rewrite a TeX or LaTeX primer, but namely:

1. an installation guide which will describe precisely the installation, the removal and the maintenance of the product on each platform;
2. a quick starting guide which will describe how to typeset a document as quickly as possible;
3. a user manual which will describe in details what is possible at each step of the creation of a document, and the technical aspects of XEmTeX.

Current status of the project

The project has officially started November 2002 for 18 months. We did not require a lot of manpower and it is a small project among the RNTL funded projects. To make progress, we are adopting an iterative process: rather than tackle the whole problem at once at the risk of ending nowhere, we prefer to finalise small parts and release often.

Hopefully, the first 0.1 release of XEmTeX will be available for the 2003 EuroTeX conference![3] Once the initial framework is set up, we will build much more easily on top of it.

Conclusion

We think that TeX deserves to get a much broader audience than is the case now. The price to pay to get high quality typesetting is not that expensive. It may even be cheaper than the price to pay for using some commercial products which are less reliable. The national funding we have got enables us to make a further step in simplifying the access to TeX for many people through the use of free software. We hope that the TeX community will stand together and require bigger funding to revive the TeX ideas into much more modern programs that could find their way in the mainstream of DTP programs.

3. Updated since; you can download the current version from `http://www.fptex.org/xemtex/`.

iTEXMac, an Integrated TEX Environment for Mac OS X

Jérôme Laurens
Université de Bourgogne
Laboratoire Analyse Appliquée et Optimisation
9, Avenue Alain Savary
BP 47 870
21 078 Dijon CEDEX
jerome.laurens@u-bourgogne.fr
http://www.u-bourgogne.fr/Monge/jlaurens

Abstract

At the end of year 2000, Apple released the first public beta of its new generation operating system named Mac OS X, based on former NeXT technologies. More than just a new release, it opened the world of the Unix tools to the Mac community. Very early, a port of the standard Unix teTEX distribution to the Mac OS X platform was available, made by Gerben Wierda, and the first open source Mac OS X front-end dedicated to TEX, named TEXShop, by Richard Koch. Last year, the first public release of iTEXMac came out when it became clear that TEXShop's developer policy was to keep it basic software. The purpose of this article is to present iTEXMac, an integrated TEX environment, as part of a wider open source project which aims to make all our powerful TEX tools easier to use, with as Mac-like a look and feel as possible.

Résumé

À la fin de l'année 2000, Apple a rendu publique la première version de son système d'exploitation de nouvelle génération appelé Mac OS X, basé sur des technologies NeXT. Plus qu'une simple nouvelle version, cela ouvrit les portes du monde des utilitaires Unix à la communauté Mac. Très tôt furent disponible une version de la distribution Unix standard teTEX adaptée à la plateforme Mac OS X par Gerben Wierda, et la première façade dédiée à TEX sur Mac OS X appelée TEXShop distribuée sous licence open source par Richard Koch. L'année dernière, la première version public de iTEXMac est sortie quand il est clairement apparu que TEXShop allait rester un logiciel d'utilisation basique. Le but de cet article est de présenter iTEXMac, un Environnement TEX Intégré, comme partie d'un projet plus général distribué en open source, dont le but est de rendre plus accessibles tous ces puissants outils TEX en gardant autant que faire ce peu la patte Mac.

Presentation

iTEXMac is an integrated suite of software including a text editor, a PDF viewer, a project manager and a front-end to the tools and engines available in a teTEX-like distribution. Each part of the suite will be discussed in more detail in separate sections of the article; here we point out what makes iTEXMac a unique product.

Regardless of the platform, different kinds of TEX-oriented software are available to compose a document. One can choose a completely WYSIWYG option using TEXmacs, a WYSIWYM option using LyX, or the original way of life with just a code editor, a set of macros and a viewer. It seems that no solution can be fully satisfying: on one hand a really nice user interface is constrained by a proprietary file format, on the other hand using and configuring an emacs-like text editor presents a challenge.

Among all the features one can expect from a good product, conflicts exist between look and feel, power and efficiency. Such problems are partially resolved by using mutability, to adapt the software to the end user. Examples of flexible software are emacs and alpha, each one using its underlying scripting language — Lisp and Tcl, respectively. With them, the user can add new features targeting specific tasks like TEX source editing, but has only limited control over displaying. Moreover, these programs serve the much more general purpose of code editing, so that they are not very easy to use. From a TEX point of view, they are extremely powerful, but some features are useless, and others are lacking. So, the long term aim of iTEXMac is to elaborate an all-in-one modular environment to edit, typeset, preview and print TEX-related files. This software achitecture will also be interesting if the different components are sufficiently tied together to result in a better user experience.

All this is fully implemented in iTEXMac using

Objective-C as an underlying scripting language and Mac OS X Application Kit, because Apple's implementation of this object oriented C variant is particularly suitable. Since this application programming interface named Cocoa evolved from NeXTSTEP and OPENSTEP, iTEXMac might be partly ported to other systems using the free software GNUstep.

iTEXMac is provided with English, French and German localizations, the latter by Keith J. Schultz.

Installing iTEXMac

teTEX-like distribution. As a graphical front-end, iTEXMac needs a teTEX-like distribution to be fully functional. For Mac OS X, three distributions are available. Gerben Wierda maintains both an enhanced port of teTEX, with additional packages and programs from TEX Live:

`http://www.rna.nl/tex.html`

Another teTEX 2.0.2 port is available from fink, through one of its graphical user interfaces, finkcommander.

`http://finkcommander.sourceforge.net/`

Once installed, the standard tools are available from the command line interface of Mac OS X Terminal application, but there is no need to fix any environment settings, which hides the Unix layer in Mac OS X.

With the new operating system, Apple replaced its old graphic format named PICT with PDF, such that iTEXMac is naturally oriented toward pdfTEX instead of TEX. However, DVI and PostScript file formats are supported, provided a Ghostscript package is installed for the latter. One is available from the previous url's.

For those who are not familiar with the Unix side of TEX (and Mac OS X), very important information about Gerben Wierda's teTEX distribution is available in the local folder `/Library/teTeX/`:

- `README.texmf.macosx`
- `README.texmf.howtexfindsfiles`

iTEXMac package and goodies. While iTEXMac may be available on CD, the latest version should be downloaded from the official site hosted by the open source software development website sourceforge at:

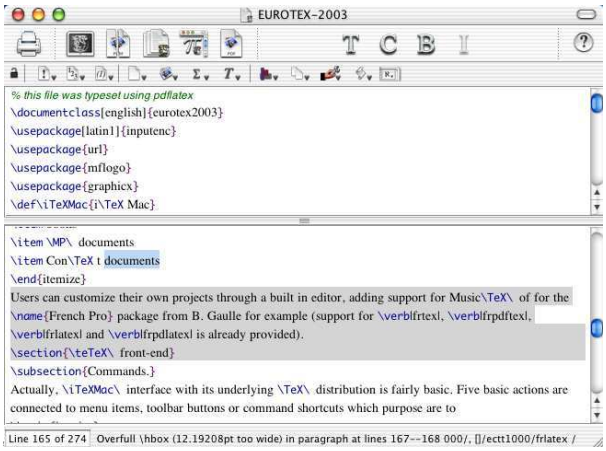
`http://itexmac.sourceforge.net/`

One can find in the download section the disk images for the following products:

- iTEXMac, both stable and developer release
- an external editor for iTEXMac key binding
- *Hypertext Help With LATEX*, wrapped as a searchable Mac OS X help file.
- the TEX Catalog Online, wrapped as a searchable Mac OS X help file.

An updater allows checking easily for new versions. To install iTEXMac, just download the disk image archive, double click then follow the instruction of the readme file.

The built in text editor



Editor window with two views of the same text, a main toolbar to switch to the terminal, the output or log documents and launch the typesetting commands, a small toolbar of menus to insert text templates, navigate through the source and launch third party extensions.

The text editor built into iTEXMac heavily relies on the word processing facilities of the Mac OS X Application Kit. However, editing has been enhanced for TEX coding by various means like enhanced text selection schemes, automatic LATEX environment closing and other tools detailed in following subsections. Moreover, iTEXMac allows the use of an external text editor if the built in one does not suit.

Editing. Like many code editors, iTEXMac actually uses text attributes to highlight syntax, changing possibly both the color and shape of the fonts used. Text selection knows about TEX groups, commands and LATEX environments. Balancing parentheses, brackets and braces are highlighted according to user preference.

Character encoding. As an open platform, Mac OS X supports a huge set of character encodings (more than 80, including Unicode) and iTEXMac implements the user interface to manage them. This allows editing files in different languages or from different platforms.

The main problem concerning character encoding is transparency: except for Unicode, there is no rule to automatically recognize the character encoding of a text file. So, iTEXMac has a customizable default encoding used to read and write text files when no external information is given. Moreover, iTEXMac parses the text files for hints about the real character encoding, and when it

finds in the file contents an explicit declaration of a character encoding it reads the file again with that encoding. For this purpose, iTeXMac recognizes the following kinds of hints:

1. an inline instruction at the very beginning of a source file:

   `%!iTeXMac(charset):`*character encoding*

   As a TeX comment, it is ignored by the various typesetting engines. The *character encoding* is simply the name of the character encoding with a standard syntax (e.g. ISO Latin 1, code page 1251, Mac OS Roman...)
2. the use of the LaTeX package `inputenc.sty`, parsing the optional argument:

   `\usepackage[`*encoding*`]{inputenc}`
3. the use of the ConTeXt `\enableregime` command, parsing the optional argument:

   `\enableregime[`*encoding*`]`
4. emacs scheme on the first input line:

   `%-*-coding:`*character encoding*`;-*-`
5. Mac OS X hidden internals.

With this management one is able to split a source document into many different files, each using its proper character encoding. This is very similar to the way emacs manages encoding with its `-*-coding:` *name* `-*-` option, with the addition of `inputenc.sty` and `\enableregime` related hints. In a forthcoming version, iTeXMac is planned to also support one more hint: meta data stored in an external project file.

Spell checking is known to be a hard problem when applied to TeX input due to the complex definition of words and word delimiters. For Mac OS X, dedicated software has been ported from previous products or newly developed, namely Rick Zaccone's Excalibur at

`http://www.eg.bucknell.edu/~excalibr/`

and Anton Leuski's cocoAspell, a port of the free and open source spell checker GNU aspell, at

`http://www-ciir.cs.umass.edu/~leouski/cocoaspell`

While both are TeX-aware, the former is based on older Mac word services and will be used as a separate application. The latter also knows about HTML code and is automatically integrated to iTeXMac and virtually any other application when properly installed. Moreover, it is possible with cocoAspell to automatically check spelling as you type, misspelled words being underlined in red. But when used with another text editing application (TextEdit, TeXShop, Mail), this continuous spelling check feature will definitely mark all TeX commands as misspelled, which is unwanted. In order to fix this very annoying problem, iTeXMac code patches some deep Mac OS X Application Kit internals: this is the only TeX aware text editor with working automatic spell checking.

Although spell checkers are generally designed with a default language, we can imagine that each text file could contain informations about the dictionary and language to be used for spell checking. This is planned for a forthcoming version, in an implementation very similar to the one for encoding. The ultimate feature should be to switch the spelling language inside one input file according to the context, but this needs a huge amount of coding... Splitting the sources according to their natural languages and encodings should be more than sufficient.

Macros menus. The text editor window gives access to five macros menus to help inserting TeX commands for the document, text, math, graphics and everything else. Those menus are configurable and their contents can change on a per document basis. All the standard LaTeX commands are available through these menus, whereas ConTeXt and METAPOST support is not complete (due to a lack of knowledge of the author). However, users can customize the interface according to their needs with the aid of a dedicated graphical menu editor.

Navigation menus Three other menus facilitate navigation through source code. The text is parsed to recognize special tags. One menu is dedicated to LaTeX or ConTeXt sectioning commands and also works for METAPOST figure keywords, another menu is dedicated to labels and references, allowing to easily insert `\ref`'s and `\eqref`'s. Finally, each line beginning with

`%!iTeXMac(mark):`*comment*

gives an entry titled *comment* in a third menu, where one is free to mark any part of the text.

Key binding. To speed up text entry, iTeXMac allows customizing the system key mapping. It is possible to map essentially any key combination to macro actions, including scripting. There is an external editor to manage those mappings in a flexible and friendly way.

Some Apple documentation leads people to believe that any text editor (e.g. TextEdit) written with Mac OS X Application Kit naturally gets the benefit of emacs-like key bindings. This is only the case for a very limited set of shortcuts. iTeXMac expands this feature significantly.

Toward WYSIWYG. The built in text editor does not provide any WYSIWYG support while editing, and is not meant *a priori* to ever become completely WYSIWYG. However, it can show a character palette with 42 menus gathering text and mathematical symbols, as they would appear in the output. It is a graphical front-end to the standard LaTeX packages

- `amsfonts.sty`

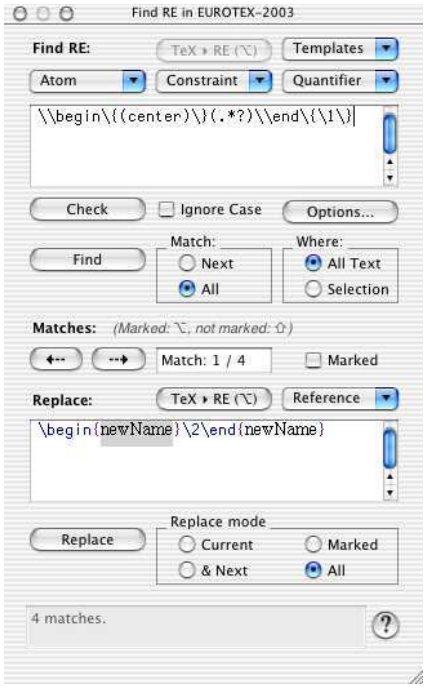
- `amssymb.sty`
- `mathbb.sty`
- `mathrsfs.sty`
- `marvosym.sty`
- `wasysym.sty`

which makes thousands of symbols available in one click. Due to a lack of knowledge, there is no counterpart for the ConTEXt world.

Some kind of assistant should also provide the user with a list of standard equation and table templates with both output and input forms. This kind of WYSIWYG support would be a shortcut to the large number of different mathematical environments, very useful for beginners. It is planned for a further release, unless it becomes available as an external plug-in.

Regular expressions. The built in text editor implements Unicode aware regular expressions. A dedicated find/replace panel has an elaborate graphical user interface to help the user construct, check and use regular expressions. A limited set of built in regular expressions is provided (e.g. to change LATEX environment names), and the user can save frequently used regular expressions.

Despite the regular expressions used being advanced ones, they are not Perl compatible. Once a library of Perl compatible and Unicode-aware regular expressions becomes available, code will be added to support them.



Advanced regular expressions panel with find and replace fields, and various controls to help in elaborating, checking and using the regular expression.

Extensions. An AppleScript implementation and a plug-in architecture are the two means provided to let the user expand iTEXMac's built in text editor.

AppleScript, which is Apple's designated scripting language, is fully supported. The basic features provided by Mac OS X application kit are extended to fully support undo management and string replacement. It is then possible to create high level editing scripts to expand ad infinitum iTEXMac functionality. However, those scripts are of limited use in practice when they deal with very heavy text editing tasks, due to performance reasons.

Plug-in architecture. While using the Mac OS X Objective-C application kit framework, iTEXMac provides a very efficient plug-in architecture. This was originally designed to expand the editor possibilities by using assistants. In fact, an array assistant is available which helps in inserting array code for LATEX. This is a very basic assistant written for demonstration purposes only. An open source development kit is available but there are no developers yet.

Here are some ideas for assistants that would be of great help to new users or for experimentation:

- a layout helper via a `geometry.sty` front-end
- a layout helper for list environments
- a WYSIWYG helper for fonts
- a helper for customizing the sectioning commands
- ...

The project manager

The notion of project comes naturally when working on a ConTEXt or LATEX document because many files are created as compilation side effects. So, project management is a task assumed by one of the core components of iTEXMac. Each document created should have an associated project file (with extension `pTeXMac`) where meta data should be stored. Due to historical reasons, the project implementation has been partially hidden, so as to mimic the user interface of former Mac OS TEX editors. This leads to elaborating a small list of default projects that would fit the widest range of situations, including for example

- LATEX documents
- LATEX documents with PDF engines
- books
- METAPOST documents
- ConTEXt documents

Users can customize their own projects through a built in editor, adding support for MusicTEX or for the French Pro package from B. Gaulle for example (`frtex`, `frlatex`, `frpdftex` and `frpdflatex` support is already provided).

teTEX front-end

Commands. Actually, the iTEXMac interface with its underlying TEX distribution is fairly simple. Five basic actions are connected to menu items, toolbar buttons or command shortcuts which do the following:

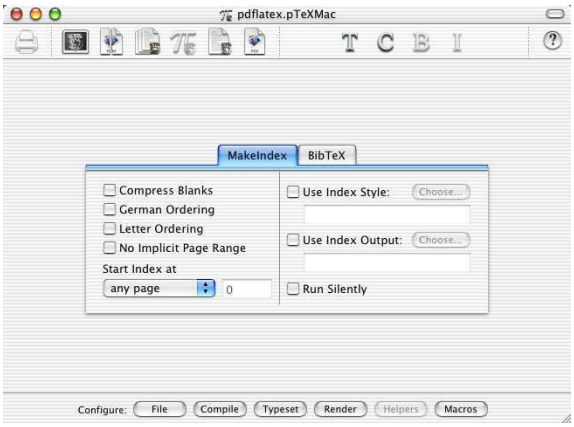
- compile (e.g. running `pdflatex`)
- typeset (e.g. running `pdflatex` twice)
- make the bibliography (e.g. running `bibtex`)
- make the index (e.g. running `makeindex`)
- render graphics (e.g. running `dvipdf`)

The compile action should be used while writing a document, the typeset action should concern the final stage and may include an index or bibliography build phase. By default, all these actions are themselves connected to shell scripts stored on a per project basis. If necessary, the end user can edit project files and customize the shell script launched by iTEXMac. He also can alter the whole action performed in one of two ways:

- customizing the project document
- inserting an in-line instruction at the very beginning of a source file:

  `%!iTeXMac(compile):` *one line shell script*

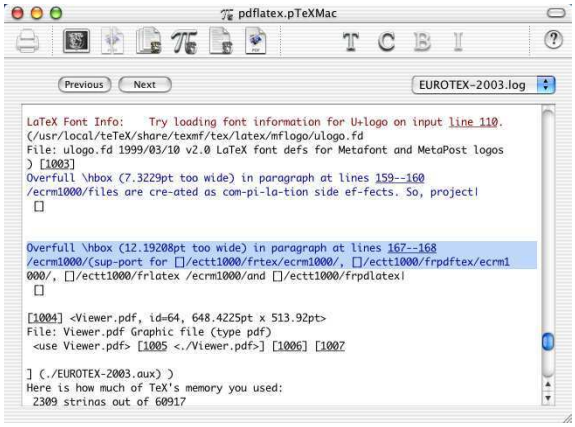
As shown below, a graphical user interface is available that wraps the command line options of `makeindex`. One is also available for `bibtex`, and others are planned for the standard tools. Apple Events allow third party applications like Alpha X to pilot iTEXMac.



Project panel wrapping the `makeindex` command line options.

Log files are a valuable source of information when a problem arises, not only for beginners but for TEX gurus too. iTEXMac displays and parses both the output of the typesetting action and the corresponding log file. Warnings and errors are displayed with different colors than the rest of the text to point out relevant information, and the corresponding line numbers are displayed as HTML links. They are linked to the line of the input file where the error or warning occurred: clicking on that link will open the file in the text editor, displaying a light gray background at the given line. See below.

This procedure does not work in a very limited number of situations, especially when the TEX source extract explaining the error interacts with the "normal" log contents. At present, the `--file-line-error-style` option of the typesetting engines is not supported, but it is planned for a future version.



Log file window outlining errors, warnings or page numbers and displaying navigation links.

A navigation facility from log file to output is also provided. When iTEXMac parses a string like `[`*a number*`...]`, *a number* is displayed as an HTML link. When clicked, the window of the output file is brought to the front and switched to the right page. Like the previous feature, it is not completely reliable because of the lack of log syntax rules, however, it works fairly well.

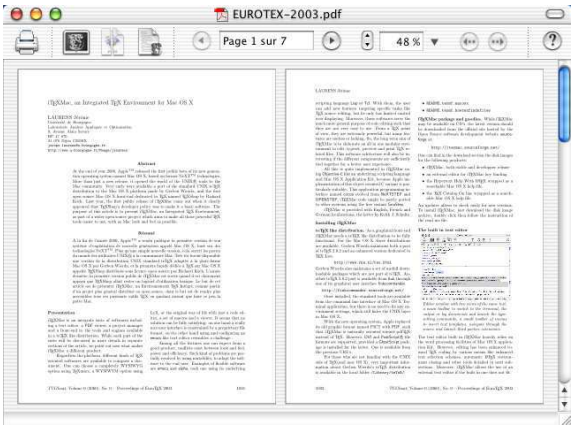
Parsing the log file in the background allows for informing the user when some problem occurs while typesetting, with an audio alert and an informational message at the bottom of the text editing window. Despite this not being absolutely reliable (see above), it can be of great help.

This feature is resource consuming and can be turned off in the preferences.

Documentation and help, provided with teTEX, is available though a menu. Moreover, Graham Williams' TEX Catalog Online and Sheldon Green's HyperText Help With LATEX are available as Mac OS X searchable help books accessible from the built in Mac OS X Help Viewer. The former lists about 1100 TEX, LATEX, and related packages and tools.

As a side effect, iTEXMac can open PS, EPS and DVI files with a double click, by converting them to PDF. In a way, it therefore plays the role of a PostScript viewer.

The built in PDF Viewer



Viewer window in the facing pages mode, with a main toolbar to switch to the terminal, the source or log documents, navigate through the pages and manage magnification.

A basic PDF viewer is provided in iTEXMac. Once again, this is a wrapper of built in Mac OS X technologies that implements the basic features of a PDF viewer. Unfortunately, Mac OS X support for this file format is sometimes limited with the output of the `pdftex` command. In such a problematic situation where the PDF can not be properly rendered by iTEXMac, an alternate solution is offered by using an external PDF viewer. This alternate solution can be permanent or temporary.

Using an external PDF viewer is also welcome because Mac OS X technologies do not support any link related or indexing features, thus ignoring the benefits of the `hyperref.sty` package and ConTEXt facilities, for example. Before these are implemented in iTEXMac's own code, or made public by Apple, a standalone previewer can be used.

Navigation between input and output is one of the great features missing in iTEXMac. This "Synchronicity" feature was first available in *Textures*, a Mac product from Blue Sky Research. Then a less efficient implementation was provided for LATEX users through the use of `srcltx.sty` which added `src:` specials in the DVI files. This task is now assumed by typesetting engines through the command line option `-src-specials`.

But all this is useless since the preferred output file format used by iTEXMac is PDF and all translators from DVI to PDF filter out this information. In fact, there is a real need for `pdftex` support of the `-src-specials` option, not only from the Macintosh community but also from `gsview` and `xpdf` users. It remains evident that the implementation of such a feature must be revisited, because PDF contents is not a stream. Nothing has been done yet in this direction.

However, a poor man "Synchronicity" feature has been implemented to help in switching from source to output. It is based on a tiny LATEX package written by Pietro D'Ancona and provided with the iTEXMac distribution, which essentially contains

```
\newtoks\@tempoutput
\global\@tempoutput\expandafter
    {\the\output\immediate\write15
    {[\thepage] \the\inputlineno}}
\global\output
    {\expandafter\the\@tempoutput}
\AtBeginDocument
    {\immediate\openout15=%
        \jobname.pageLineIdx
    \immediate\write15{[1]
        \the\inputlineno}}
\AtEndDocument
    {\immediate\write15{[1]
        \the\inputlineno}%
    \immediate\closeout15}
```

This LATEX code must be inserted in each master LATEX document. It opens an auxiliary file in which are stored some hints about the correspondence between input file line numbers and output file page numbers. The PDF viewer parses this auxiliary file and uses the hints to display the proper part of the document, either input or output.

Future directions

In addition to the forthcoming improvements previously mentioned, here we detail other features to come. Some of them just reflect the natural development policy of iTEXMac, others come from user requests. This should convince interested people to contribute to the iTEXMac project.

Bibliography management. At present, iTEXMac is missing a built in support for such a feature. The bibliography database management is now provided by Michael McCracken's BibDesk (beta version 0.8), another sourceforge project at

`http://bibdesk.sourceforge.net/`

This application also implements a Mac OS X service which allows any application to automatically insert or complete bibliography references in the currently edited text. There's no way iTEXMac will offer in the near future these features as built in, however, a better integration with a BibDesk-like data base application is planned.

This solution is useful for big bibliography data bases but it does not manage small ones declared explicitly in `thebibliography` LATEX environments. iTEXMac should provide a feature devoted to this task. In order to improve efficiency, this inline bibliography should be declared in a standalone file called by the master TEX document.

TEX macros are of frequent use as soon as one becomes familiar with TEX. While standard macros are managed by iTEXMac, nothing is done for user defined TEX macros. While parsing the whole source files for `def`'s and other `newcommand`'s would certainly be rather difficult and time-consuming, users should gather their macro declarations in one file called by the master TEX document, just like the inline bibliography above.

Completion is a standard feature of code editors but is not built in to iTEXMac. However, one can install TextExtras from

`http://www.lorax.com/FreeStuff/TextExtras.html`

It is a tiny framework written by Mike Ferris, whose purpose is to give all Cocoa applications useful features, including code completion. It is done by using a dictionary shared by all applications, declaring shortcuts to be completed, and expected output. It also scans the preceding text for words with the same prefix but ignores the backslash introducing TEX macros.

The source code is available; it should be borrowed and extended for iTEXMac's purposes, mainly by considering TEX specifics.

Equation Service and LATEX Equation Editor The former is a very nice system service to typeset only parts of a document. Its author Doug Rowland makes it available at

`http://www.esm.psu.edu/mac-tex/EquationService`

When some text is highlighted in any Cocoa application, Equation Service can typeset it as LATEX commands and either put the result on the clipboard or replace the original text by the resulting PDF or TIFF output.

LATEX Equation Editor was written by J. McKenzie Alexander to add an editor layer to the previous service. This basic editor is downloadable from

`http://evolve.lse.ac.uk/software/EquationEditor`

While iTEXMac's built-in text editor is not graphics oriented, it does not support such a system service, and so should implement a similar feature (from the client side). Moreover, these three packages provide approximately the same functionality, and iTEXMac will implement its own equation service and equation editor functionality.

Technical enhancement. The next major release will benefit from better internal code design. The core components of iTEXMac — built-in editor, PDF viewer, teTEX front end — will become code libraries (frameworks in Mac OS X terminology), such that virtually all the features will be accessible for third party coding. It will be useful for iTEXMac plug-ins but other software will be able to use them too, including the application described above.

Conclusion

After more than one year, iTEXMac has proven to be an interesting product, and the future promises to be even more interesting. (Many significant improvements have been made since this writing.)

Many people have contributed to the project, their credits are found with their contribution in the iTEXMac distribution.

iTEXMac logo:

Description of Knowledge of Mathematical Programs with TEX and XML

Balázs Vecsei
BUTE Mathematical Institute,
5th year student
1111 Budapest, Egry József u. 1, Hungary
vecseib@math.bme.hu
http://www.math.bme.hu/~vecseib

Abstract

There are many free mathematical programs and databases on the Internet, but use of these programs can be difficult. In this paper I would like to show an XML-based mathematical knowledge description language. In this language I will describe the ability of different kinds of mathematical programs. In my opinion, XML is a useful base for this language, because it is well standardized, and there are tools for handling it.

Résumé

Il y a une multitude de programmes et bases de données mathématiques sur Internet, mais leur utilisation peut être difficile. Dans cet article nous présentons un langage de description de connaissance mathématiques basé sur XML. Dans ce langage nous décrivons les fonctionnalités de divers programmes mathématiques. Nous pensons que XML est un fondement utile de ce langage, grâce à sa normalisation et à l'existence d'outils de traitement de celui-ci.

Introduction

I think we need a mechanism to describe the knowledge about free mathematical programs, because more than 50 mathematical programs are on the Internet, under the GPL or other free licenses.

The documentation of the free mathematical programs is often written in a TEX-based language, and we can download it, along with the programs' source.

I use this documentation to describe the programs. I would like to write a more widely applicable program recommendation system, connected with other projects in this area. This recommendation system needs a machine readable description of free mathematical programs. Some general categories:

- *activemath*: The goal of the project's research and development is a web-based interactive learning system (for mathematics). The project provides an architecture, basic knowledge representation, and techniques for new-generation on-line interactive mathematics documents (textbooks, courses, tutorials) and e-learning [9].
- *mathweb*: This project supplies an infrastructure for web-supported mathematics [6].
  - a software system that connects a wide range of mathematical services via a common mathematical software base.
  - a set of mathematical services that support all aspects of doing mathematics on the web.
  - an interlingua for mathematical communication (OMDoc)
  - a portal for potential users and a discusssion forum for developers
- *openmath*: OpenMath is an emerging standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the web [11].

GAP (Groups, Algorithms and Programming) and TPTP (Thousands of Problems for Theorem Provers) provide a selection mechanism to choose the most useful method or program.

TPTP system recommendations The TPTP Problem Library is a library of test problems for automated theorem proving systems [15]. It can convert between a lot of theorem provers:

Bliksem	CoDe	CLIN-S
Dedam	DFG	DIMACS
FINDER	ILF	KIF
leanTAP	3TAP	METEOR
MGTP	OSCAR	OTTER
PROTEIN	PTTP	Satchmo
SCOTT	SEM	SETHEO
SPRFN	THINKER	Waldmeister

This table shows we need system recommendations for theorem provers, and I think, not only in logic.

GAP method selection GAP is a GPL-licensed computer algebra system. Its main area is a discrete algebra, especially groups. GAP has a method selection algorithm, and can select the optimal methods from among the applicable methods to given problems. They assert that a specific method will be better than a general method [1].

Describing mathematical resources

We can use the program documentation to describe the mathematical resources provided. There are a lot of different `toc` (table of contents) structures, so I think it is helpful to convert the `toc` to an XML-based language, because it is standardized.

The knowledge description file has two part, the `metadata` and the `toc`. The metadata contains data about the program or resource. The toc contains the converted `.toc` file. I use Emacs Lisp programs to convert the `toc` files to XML.

Metadata I use the Dublin Core Metadata standard in our mathematical knowledge description (mkd) language. I made it by hand with Emacs psgml, because I think that is the simplest method [5].

```
<resource>
<metadata>
<title>gap</title>
<creator>gap group</creator>
<date>2002-12-04</date>
<type>software</type>
<description>
gap is a system for computational discrete
algebra ...
</description>
<format io="in">plain txt</format>
<format io="out">plain txt</format>
<source>http://www.gap-system.org</source>
<rights>GPL</rights>
</metadata>
```

TOC This part is automatically generated from `manual.toc`. Details from `manual.toc`, the main manual of gap:

```
\chapcontents {17}{Combinatorics}{141}
\seccontents{17.1}{Combinatorial Numbers}{143}
...
```

Details from description of gap:

```
<toc>
<from>doc/ref/manual.tex</from>
<chapter title= "Combinatorics"
  chap= "17" page= "141">
<title chap = "17" sec = "1" page = "143">
  Combinatorial Numbers
</title>
...
</chapter>
</toc>
</resource>
</mkd>
```

Document Type Definition The DTD file defines the syntax of a particular XML language.

Details from `mkd.dtd`:

```
<!ELEMENT mkd (metadata, resource+)>
<!ELEMENT resource (metadata, toc+)>
<!ELEMENT toc (from, chapter+, title+)>
<!ELEMENT chapter (title+)>
```

Generate the question

In this process I suppose the question is in MathML or OMDoc format. But these formats are similar to assembly code in programming. We can't write or read the MathML or OMDoc source directly, so we need authoring tools.

- *omdoc*: OpenMath is a standard for representing mathematical data in as unambiguous a way as possible. It can be used to exchange mathematical objects between software packages, or via email, or as a persistent data format in a database. It is tightly focused on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this [7]. QMath is a batch processor that produces an OMDoc file from a plain Unicode text document [12].
- *mathml*: MathML is an XML application for describing mathematical notation and capturing both its structure and content.

  The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web [2].

  There are a number of programs which can convert LaTeX to MathML, or can save in MathML format:

  - OpenOffice
  - Amaya, the W3C browser
  - LaTeX2HTML, with MathML package
  - Maple
  - Mathematica
  - TeX4ht, a general TeX to SGML/XML translator
  - TtM, a TeX to MathML translator
  - WeM, an MathML editor that converts a subset of LaTeX to MathML.

Tools

In this section I enumerate some useful tools for handling XML files. All of them are free. The Emacs Lisp programs are at `www.emacswiki.org/cgi-bin/wiki.pl`.

- *emacs* and *Emacs Lisp* [8].

- *psgml* [13], an Emacs major mode for editing SGML and XML documents.
- *xml.el*, a full XML parser, parses a file and returns a list; it is now part of GNU Emacs.
  - `(car (xml-parse-file "gap.mkd"))`
  - `(xml-node-name node)`
  - `(xml-node-attributes node)`
  - `(xml-node-children node)`
- *dom.el*, generates a document object model tree from an XML data structure.
  - `(dom-make-document-from-xml (car (xml-parse-file "gap.mkd")))`
  - `(setq oms (car (dom-document-get-elements-by-tag-name doc1 'OMS)))`
  - `(mapcar 'dom-attr-value (xpath-attribute-axis oms))`
- *xpath.el*, *xpath-parser.el*, handles XPath expressions. They are based on an Emacs Lisp implementation of a semantic parser (similar to *yacc* or *bison*).
- *rxp*, XML parser and validator, Debian package, Windows.
- *xsv*, XML schema validator, Red Hat package, Windows.
- *xsltproc*, XSLT transformer, Debian package, Windows.

Summary

In this paper I have presented a possible description of mathematical programs, as a main part of a program recommendation system. The program recommendation procedure is based on the following steps.

1. describe the mathematical resources;
2. generate an OMDoc and/or MathML question;
3. select the relevant data from descriptions and questions;
4. select a program;
5. write or convert the code;
6. select a service where we can run our code, or download the program.

References

[1] The GAP Group. *GAP — Groups, Algorithms, and Programming, Version* 4.3, 2002. http://www.gap-system.org.

[2] Ron Ausbrooks, Stephen Buswell, and David Carlisle et al. Mathematical markup language (MathML) version 2.0. Technical report, W3C, 2003. http://www.w3.org/TR/2003/WD-MathML2-20030411/.

[3] Neil Bradley. *Az XML kézikönyv*. Szak Kiadó, 2000.

[4] Robert J. Chassell. *Programming in Emacs Lisp: An Introduction*. FSF, October 1997. http://www.gnu.org/manual/emacs-lisp-intro.

[5] Dublin Core Metadata Initiative. *Dublin Core Metadata*, 2003. http://dublincore.org.

[6] Andreas Franke, Michael Kohlhase, Paul Libbrecht, and Jurgen Zimmer. *Mathweb*, 2003. http://www.mathweb.org/.

[7] Michael Kohlhase. OMDoc: An open markup format for mathematical documents. Technical report, Computer Science, Carnegie Mellon University, February 2003. http://www.mathweb.org/omdoc/omdoc.ps.

[8] Bil Lewis, Dan LaLiberte, and Richard Stallman. *GNU Emacs Lisp Reference Manual*. FSF, May 1998. http://www.gnu.org/manual/elisp-manual-20-2.5/.

[9] Paul Libbrecht. *Activemath*. DFKI Saarbruecken, 2002. http://www.activemath.org/.

[10] Brett McLaughlin. *Java és XML*. O'Reilly, Kossuth, 2001.

[11] The OpenMath Society. *OpenMath*, 2002. http://www.openmath.org/.

[12] Alberto Gonzalez Palomo. *QMath OMDoc authoring tool*, 2003. http://personal.iddeo.es/ret009t0/qmath/.

[13] Lennart Staflin. *Editing SGML with Emacs and PSGML*, 1996. http://www.lysator.liu.se/~lenst/about_psgml/.

[14] Richard Stallman. *GNU Emacs Manual*. FSF, August 1998. http://www.gnu.org/manual/emacs-20.3/emacs.html.

[15] Geoff Sutcliffe and Christian Suttner. *TPTP Manual*, 2003. http://www.cs.miami.edu/~tptp.

Real-Time Grid Fitting of Typographic Outlines

David Turner
david@freetype.org

Werner Lemberg
Kl. Beurhausstr. 1
D-44 137 Dortmund, Germany
wl@gnu.org

Abstract

This paper describes an auto-hinting algorithm to grid fit glyph outlines. Instead of generating new hints for font files or font editors, the computations are done in real-time while rendering the glyph, posing strong efficiency constraints.

A freely available implementation can be found in the FreeType font rendering library.

Résumé

Cet article décrit un algorithme d'auto-*hinting* pour des contours de glyphes définis sur une grille. Au lieu de générer des nouveaux *hints* pour des fichiers de fonte ou des éditeurs de fonte, les calculs sont effectués à la volée au moment de l'affichage du glyphe, ce qui pose des sérieuses contraintes d'efficacité.

On trouvera une implémentation librement disponible de cet algorithme dans la bibliothèque de sous-routines d'affichage de fonte FreeType.

Overview

The auto-hinting algorithm is roughly divided into two distinct parts.

- A *feature analysis* phase to study each glyph outline in order to detect interesting 'features' in them.
- A *grid fitting* phase to adjust the position of such features to the device pixel grid, and to align the outline points to these.

In the following we directly reference files and structures of the auto-hinter in the FreeType library, version 2.1.4. This should help in understanding how the algorithm is implemented.

Feature Analysis

To produce well-hinted glyphs it is not sufficient to improve the shape of a glyph as much as possible. It is also necessary to find font-wide parameters.

Global Analysis The first step is to find global metrics. In the following it is assumed that the font is Latin-based. Font specific details regarding how to map an input character to an output glyph are omitted for simplicity.

- Compute the standard stem widths and heights of the glyphs. This is done by loading the letter 'o' and running the feature analysis on it to get the sizes of its 'stems'.

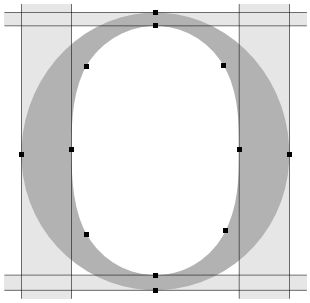


FIG. 1: The stem widths and heights of the glyph 'o'.

- Compute the *blue zones*. Outlines of certain characters such as 'Z', 'C', etc., are loaded, comparing the top-most and bottom-most coordinates to find the small horizontal zones which envelop them.

All of this is done in the C source file `ahglobals.c`. These global parameters are computed only once per font, then stored for later use during the grid fitting process.

Glyph Analysis An *outline* is the ordered set of all points defining a glyph. It consists of one or more *contours*. A contour is a closed curve; points can be control points or

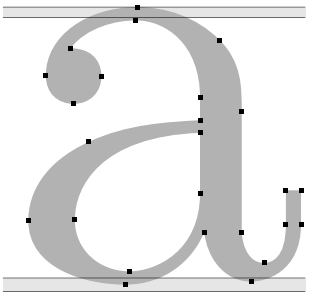


Fig. 2: Two blue zones relevant to the glyph 'a'. Vertical point coordinates of *all* glyphs within these zones are aligned.

on the contour. For example, the outline of glyph 'a' in figure 2 consists of two contours.

Each outline is decomposed into an `AH_Point` array.

```
struct AH_Point
{
  AH_Flags      flags;
  FT_Pos        ox, oy;
  FT_Pos        fx, fy;
  FT_Pos        x,  y;
  FT_Pos        u,  v;

  AH_Direction  in_dir;
  AH_Direction  out_dir;

  AH_Point      next;
  AH_Point      prev;
}
```

The field `flags` specifies the type of the current point: a point on the contour, a conic control point for a quadratic Bézier curve as used in TrueType fonts, or a cubic control point for third-order Bézier curves as used in Type 1 fonts. `ox` and `oy` are the original scaled coordinates, `fx` and `fy` the coordinates in font units, and `x` and `y` hold the hinted coordinates (which we are really interested in).

`in_dir` and `out_dir` give the direction of the vectors from the previous and to the next point in the current contour. Finally, it is easy to guess now that the fields `next` and `prev` hold pointers to the previous and next contour point.

Both the global and the feature analysis are performed twice: first vertically (modifying y coordinates and horizontal stems), then horizontally (modifying x coordinates and vertical stems). The results are independent: It is easy to disable hinting in one dimension without impacting the results of the other dimension.

Note that the `AH_Point` structure holds two fields, `u` and `v`, which are used to store coordinates whose meaning changes depending on the context. See the function

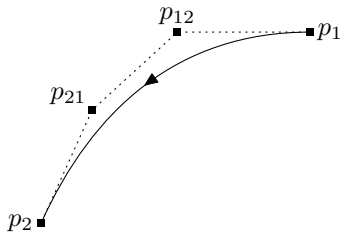


Fig. 3: A cubic Bézier curve with two control points. The 'out' direction of p_1 is `AH_DIR_LEFT`; all other directions are of type `AH_DIR_NONE`.

`ah_set_uv` in file `ahglyph.c` for details. The following 'modes' are supported:

`AH_UV_FXY`: (u, v) contains the original (x, y) point coordinates expressed in font units.

`AH_UV_FYX`: Same as above, except that 'rotated' coordinates (y, x) are used instead.

`AH_UV_OXY`: The pair (u, v) contains linearly scaled (ox, oy) point coordinates in 26.6 fixed-point format — not modified by grid fitting. '26.6' means that of a 32-bit integer, 26 bits are used for the integer part and 6 bits for the fractional part, providing a granularity of $\frac{1}{64}$ units.

`AH_UV_OYX`: Same as above, with 'rotated' (oy, ox) linear coordinates.

`AH_UV_OX`: A special case used during grid fitting. u corresponds to hinted coordinates x, and v to linear ones ox. This is used to perform light interpolation in the horizontal dimension.

`AH_UV_OY`: Same as above, but uses y and oy instead.

`AH_UV_XY`: (u, v) contains the scaled and hinted point coordinates (x, y). This is currently unused.

`AH_UV_YX`: Same as above, with 'rotated' (y, x) coordinates (also currently unused).

Directions are identified by with the following constants.

```
#define AH_DIR_NONE    4
#define AH_DIR_RIGHT   1
#define AH_DIR_LEFT   -1
#define AH_DIR_UP      2
#define AH_DIR_DOWN   -2
```

These values are carefully chosen so that $dir_1 + dir_2 = 0$ only if dir_1 and dir_2 correspond to opposite directions. This is useful in speeding up certain comparisons. In any case, these values should not be changed, since other sub-algorithms depend on them (for example, `ah_outline_compute_segments` assumes that `AH_DIR_RIGHT` is positive).

More details on the glyph analysis, like weak point flags, and major and minor directions will be explained later.

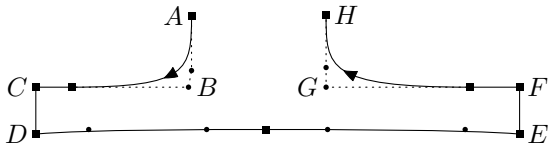


Fig. 4: A serif. Contour and control points are represented by squares and circles, respectively. The bottom 'line' DE is approximately aligned along the horizontal axis, thus it forms a segment of 7 points. Two other horizontal segments are BC and FG, while AB, CD, EF, and GH form vertical segments.

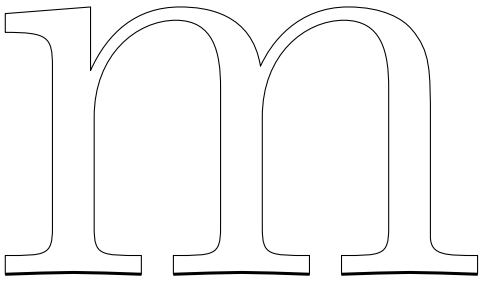


Fig. 5: The three segments marked with thick lines form an edge.

Segments `ah_outline_compute_segments` is the function used to find *segments* in an outline. A segment is a series of consecutive points that are approximately aligned along a coordinate axis. The allowed maximum deviation from a straight line is $\arctan \frac{1}{12} \approx 4.7°$ (this is a heuristic value).

A segment must have at least two points, except in the case of 'fake' segments that are generated to hint metrics appropriately, and which consist of a single point.

Each segment has a coordinate in the dimension's 'main' direction (field `pos` in the `AH_Segment` structure) and coordinates in the 'other' dimension which specifies its extrema (fields `min_coord` and `max_coord`).

As soon as segments are defined, the auto-hinter groups them into *edges* (figure 5). An edge corresponds to a single position on the main dimension that collects one or more segments (allowing for a small threshold).

The auto-hinter first tries to grid fit edges, then to align segments on the edges unless it detects that they form a *serif* (see figure 4).

Segments need to be 'linked' to other ones in order to detect *stems*. A stem is made of two segments that face each other in opposite directions and that are sufficiently close to each other. Using vocabulary from the TrueType specification, stem segments form a *black distance*. For vertical stems we have the following two cases (horizontal stems use the y axis instead for hinting):

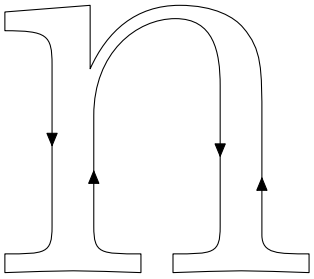


Fig. 6: The outline orientation of a glyph in a Type 1 font. TrueType fonts use the opposite direction.

- In TrueType fonts, the leftmost segment points upwards, and the rightmost points downwards.
- In Type 1 fonts, the reverse convention is used, i.e., the leftmost segment points downwards, and the rightmost points upwards.

Unfortunately, some fonts do not respect the fill convention of their own format, and others even contain a mixture of both conventions. To know precisely how to form black distances we thus need to analyze the glyphs, via the auxiliary function `ah_get_orientation`. It tries to guess the correct convention using the orientation of the points that make the glyph outline's bounding box.

This function is called from another function, `ah_outline_load`, which also sets two fields in the `AH_Outline` structure, namely `vert_major_dir` and `horz_major_dir`. They correspond to the direction of the leftmost or bottommost stem segments along the vertical and horizontal axis, respectively.

The algorithm to link segments is 'greedy' (that is, it will link as many segments as possible) and tries to find the 'closest' segment in the opposite direction for each candidate. Here, 'closest' means that the segments are sufficiently aligned in the 'other' dimension, and close in the 'main' one. The current code to determine this is as follows.

```
FT_Pos  min = seg1->min_coord;
FT_Pos  max = seg1->max_coord;
FT_Pos  len, dist, score;

if ( min < seg2->min_coord )
  min = seg2->min_coord;
if ( max > seg2->max_coord )
  max = seg2->max_coord;

len = max - min;
if ( len >= 8 )
{
  dist = seg2->pos - seg1->pos;
```

```
    if ( dist < 0 )
      dist = -dist;

    score = dist + 3000 / len;
    if ( score < best_score )
    {
      best_score   = score;
      best_segment = seg2;
    }
  }
```

Each segment has at most one 'best' candidate to form a black distance, or no candidate at all. Notice that two distinct segments can have the same candidate, which frequently means a serif, as in figure 4. The best candidate for both AB and CD is GH, while the best candidate for GH is AB. Similarly, the best candidate for EF and GH is AB, while the best candidate for AB is GH.

A stem is recognized by the following condition:

$$segment_{1_{best}} = segment_2 \quad \wedge$$
$$segment_{2_{best}} = segment_1$$

On the other hand, a serif has

$$segment_{1_{best}} = segment_2 \quad \wedge$$
$$segment_{2_{best}} \neq segment_1$$

where $segment_1$ corresponds to the serif segment (CD and EF in figure 4).

Stem segments store their best candidate in the `link` field of the structure `AH_Segment`, while serif segments use the field `serif` (their `link` field is set to zero).

Segments are also *round* or *flat*, depending on the series of points that define them. A segment is round if the next and previous point of an extremum (which can be either a single point or sequence of points) are both (conic or cubic) control points. Otherwise, a segment with an extremum is flat. In figure 4, the segment DE is flat because the previous point C and the next point F are both on the contour (and thus they aren't control points). In figures 1 and 2, the top segments are round.

Round and flat segments are collected for selected glyphs, then the average is taken to define blue zones and overshoot values (see figure 7).

Edges As mentioned earlier, edges are used to collect segments along coordinates in the 'main' dimension. There are three types of edges.

- Free edges which are not linked to other ones.
- Stem edges which contain at least one stem segment.
- Serif edges, which only contain one or more serif segments.

Edges can be flat (if they contain at least one flat segment) or round (if they contain only round segments).

Finally, edges can be linked to blue zones, depending on their position and the orientation of the segments they contain.

Fig. 7: The difference between the height of a glyph with a flat top segment and a glyph with a round top segment is called top *overshoot*. Analogously, the difference in depth is called bottom overshoot. Glyphs from the string 'xzroesc' are used compute the blue zone and overshoot value of the top and bottom of lowercase letters.

Grid Fitting

The collected data so far is now used to move the glyph's outline points to positions which improve the overall shape for a specific output device resolution. This complex process can be divided into the following steps.

Globals Each time the x or y scale changes, the globals are rescaled and fitted globally. This allows us to deactivate certain blue zones when they become too small or too big.

Edge Grid Fitting First of all, the edges are fitted to the pixel grid, in the following order.

1. In the vertical dimension, all edges linked to blue zones are aligned to the zone's position (we call them *blue edges*). This assures consistent glyph heights.
2. Stem edges are fitted to the pixel grid. The stem width and position are computed more or less independently.

   If a stem edge is linked to a blue edge, its position is directly computed. Otherwise, subtle alignments may occur, beyond our scope here.
3. Finally, serif edges are aligned.

The function `ah_hinter_hint_edges_3` in the file `ahhint.c` should be consulted for more details.

Computing the position and size of stems is very sensitive to tuning, containing a lot of heuristic constants. Most of the patches to the auto-hinter were refinements to the involved routines.

Segment Grid Fitting After aligning all edges the corresponding segments are fitted to the same position. This forces all points on these segments to adopt the edge's position. The auxiliary function `ah_hinter_align_edge_points` has more details.

Each point has a flag `AH_FLAG_DONE` which is set when it is aligned to a specific location. This is used for

FIG. 8: An angle point (left) and an inflection point (right).

the two distinct interpolation algorithms described below.

Strong Points Experience has shown that points which are not part of an edge need to be interpolated linearly between their two closest edges, even if these are not part of the contour of those particular points. Typical candidates for this are

- *angle* points (i.e., points where the 'in' and 'out' direction differ greatly);
- *inflection* points (i.e., where the 'in' and 'out' angles are the same, but the curvature changes sign).

`ah_hinter_align_strong_points` is the function which takes care of such situations; it is equivalent to the TrueType 'IP' hinting instruction.

Weak Points Other points in the outline must be interpolated using the coordinates of their previous and next unfitted contour neighbours. These are called *weak points* and are touched by the function `ah_hinter_align_weak_points`, equivalent to the TrueType 'IUP' hinting instruction. Typical candidates are control points and points on the contour without a major direction.

The major effect is to reduce possible distortion caused by alignment of edges and strong points, thus weak points are processed after strong points.

Hinting Metrics To properly hint the advance widths of glyphs, `ah_outline_compute_segments` creates two 'fake' segments corresponding to the position of the leftmost and rightmost points in the outline (for the horizontal position only).

These segments are grid fitted, and the resulting distance between them is used to correct the scaled advance width.

See file `ahhint.c` for details; the code looks like this:

```
FT_Pos   old_advance, old_rsb, old_lsb,
         new_lsb;
/* leftmost edge */
AH_Edge  edge1 =
  outline->vert_edges;
/* rightmost edge */
AH_Edge  edge2 =
  edge1 + outline->num_vedges - 1;

old_advance = hinter->pp2.x;
old_rsb     = old_advance - edge2->opos;
old_lsb     = edge1->opos;
new_lsb     = edge1->pos;

/* round result to the
   nearest integer pixel */
hinter->pp1.x =
  ( ( new_lsb    - old_lsb ) + 32 ) & -64;
hinter->pp2.x =
  ( ( edge2->pos + old_rsb ) + 32 ) & -64;
```

where rsb and lsb represent the right and left side bearing, respectively. pp_1 and pp_2 are the auxiliary points which control the advance width. All computing is done with 26.6 fixed-point numbers.

Conclusion

The general design of FreeType's auto-hinter has been proven to be very stable; as mentioned above, only small changes have been applied, most of them to refine the computations of the position and width of stems, not the basic segment linking architecture.

In the future, we will implement additional constraints to compute more sophisticated global parameters. Here is a list of improvements which we will likely implement in the near future.

- Use more glyphs than the letter 'o' to find default stem widths and heights. This is a script dependent feature.
- Assure that the advance widths of normal digits have the same value.
- Extend blue zones to cover non-Latin languages such as Arabic. This is, select character groups for this feature dependent on the script.
- Autohinting of CJK characters needs additional analysis steps to assure constant whitespace between multiple parallel stems. Currently, only stems in 'm'-like glyphs are handled this way.

References

[1] Adobe Systems, Inc. *Adobe Type 1 Font Format*. Addison-Wesley, 3rd edition, 1993.

[2] Microsoft Corporation. OpenType specification version 1.4. Available from `http://www.microsoft.com/typography`, 2002.

[3] David Turner, Werner Lemberg, and Robert Wilhelm. The freetype 2 font rendering library. Available from `http://www.freetype.org`, 2003.

L'utilisation du *Mauro*-TEX pour l'édition critique des œuvres scientifiques de Francesco Maurolico

Jean-Pierre Sutto
Département de Mathématiques, Université de Pise
sutto@mail.dm.unipi.it

Pier Daniele Napolitani
Département de Mathématiques, Université de Pise
napolita@mail.dm.unipi.it

Abstract

This paper presents the *Mauro*-TEX system, developed and used in the Maurolico Project, which aims to prepare critical edition of Francesco Maurolico's (1494–1575) scientific works. It is intended to help an editor through all phases of creating a critical edition (transcription, collation, editing, etc.).

Mauro-TEX allows the editors of large and complex projects, such as the Maurolico, to produce consistent source files, print high-quality hardcopy, and export the sources into the common electronic formats (PDF, HTML, RTF, etc.).

Mauro-TEX generates a complete mark-up of the edited texts, which makes it possible to exploit them, using a varity of software tools, to analyze and extract data: pulling out the text of a specific version from a group of versions, analyzing the error concordances between versions, doing *stemma* analysis, etc.

Résumé

Nous présentons dans cette communication le système *Mauro*-TEX, développé et utilisé pour le Projet Maurolico, dont le but est l'édition critique des œuvres scientifiques de Francesco Maurolico (1494–1575). Il est conçu pour assister un éditeur durant toutes les phases de l'élaboration d'une édition critique (transcription, collation, édition, etc.).

Le *Mauro*-TEX permet aux éditeurs de projets importants et complexes comme le Projet Maurolico, de produire des travaux homogènes, d'obtenir une version imprimée de haute qualité et d'exporter les sources vers les formats électroniques les plus courants (PDF, HTML, RTF, etc.).

Le *Mauro*-TEX produit un *mark-up* complet des textes édités permettant l'utilisation (et le développement) d'outils d'analyses (semi-)automatiques et l'extraction d'informations : extracteur d'un témoin parmi un groupe, analyse des concordances d'erreurs entre témoins, analyse du *stemma*, etc.

Introduction

Il y a cinq ans environ, commençait un projet d'édition des œuvres scientifiques de Francesco Maurolico, un important mathématicien italien de la Renaissance (1494–1575). L'édition, à son terme, fera l'objet de 10 volumes pour environ 5 000 pages. Elle réunira environ 250 textes, allant de simples fragments d'une demi-page à une œuvre entière divisée en plusieurs livres. La production de l'auteur couvre 60 années de travail et tout le champ de la mathématique de l'époque : mathématique proprement dite, mais aussi optique, astronomie, mécanique, gnomonique, etc. Il refonde, quelques fois à partir de quelques bribes, de nouvelles éditions des auteurs anciens : Euclide, Archimède, Apollonius, Théodose, Ménélaüs, etc., qu'il complète de travaux personnels particulièrement originaux. Peu publiés de son vivant, ils exerceront une influence certaine sur les mathématiciens de la fin du XVI^e siècle et du siècle suivant, en Italie via le milieu mathématique jésuite, mais aussi dans toute l'Europe.

L'édition réunit les compétences de vingt cinq personnes environ, principalement des historiens des mathématiques. Ils habitent différents pays, ont des qualification différentes, travaillent sur des machines, des systèmes d'exploitation, des logiciels différents. Le nécessaire travail d'équipe a poussé à l'exploration et exploitation des moyens informatiques disponibles aujourd'hui, mettant en place une procédure permettant à tous les chercheurs engagés de travailler de manière uniforme. Au démarrage de ce projet, différents choix étaient possibles, mais 1) les membres du projet Maurolico ne les connaissaient pas à l'époque ; 2) ils sont très différents de ce qui a été développé dans le cadre de ce projet. *edmac* est un ensemble de macros LATEX aidant à la mise en page d'une édition critique[1] ; *tustep*, de l'université de Tübin-

1. http://www.ucl.ac.uk/~ucgadkw/edmac/index.html

gen, est un système professionnel pour produire des éditions critiques. Il est complet mais relativement complexe et demande un apprentissage certain[2] ; *cte* est un système *wysiwyg* autrichien de composition et de visualisation des éditions critiques pour Microsoft Windows[3].

Le projet ayant commencé dans un Département de Mathématiques (celui de l'Université de Pise), il était somme toute naturel que l'on considére LaTeX. Le *Mauro*-TeX est un système d'édition critique des textes. C'est à la base un ensemble de macros LaTeX permettant d'élaborer un balisage, un *mark-up* des textes édités. LaTeX et les macros «philologiques» construites permettent ensuite de construire automatiquement le texte critique et son apparat.

Texte critique et apparats

Comment nous parviennent les textes des auteurs ? Les situations sont très diverses. Le seul manuscrit autographe de l'auteur peut nous être parvenu, à l'inverse le texte peut avoir eu une très large diffusion. Il a pu être copié, quelquefois à des centaines d'exemplaires et il a pu se constituer une *tradition* textuelle. À partir de la fin du XV^e^ siècle, il a pu être imprimé. Tous les exemplaires parvenus entre les mains de l'éditeur d'aujourd'hui constituent les *témoins* du texte.

Les témoins sont généralement loin d'être tous identiques. L'auteur lui-même a pu écrire différentes versions de son texte. Il a pu chercher à l'améliorer. Si le texte a été copié, le copiste a pu faire des erreurs. Et l'expérience montre que le nombre d'erreurs des copistes est toujours très important. Il oublie des mots, saute des lignes ou des folios et peut modifier ainsi profondément un texte, quelquefois jusqu'à le rendre incompréhensible. Au moyen âge, le copiste est typiquement un moine qui sait à peine lire et écrire. Les copistes se sont aussi trouvés devant des situations qu'ils ont gérées de manière très diverses. Par exemple, s'ils copiaient un texte ayant des notes marginales, écrites quelquefois par d'autres auteurs, beaucoup choisissaient d'intégrer sans avertissement les notes marginales au texte. Il faut encore tenir compte de difficultés plus matérielles, des ravages du temps, des trous dans le papier, de folios ou pages perdus, de l'encre qui s'efface, de taches d'humidité rendant illisible le texte, etc. Enfin, le nombre d'erreurs est évidemment multiplié exponentiellement selon le nombre de témoins existants.

L'éditeur a en charge de fournir au lecteur, à partir de tous les témoins, un *texte critique*, c'est-à-dire un texte qui représente les différents témoins et ce qu'il pense être la volonté de l'auteur. Le texte critique est accompagné d'un *apparat critique* qui doit donner au lecteur la possibilité de reconstruire : 1) le chemin parcouru par l'éditeur pour construire le texte critique à partir des témoins ; 2) le contenu de chacun des témoins. Il doit signaler en premier lieu dans l'apparat toutes les *variantes* entre les témoins : mots, phrases différentes, dispositions différentes du texte, notes marginales, ajouts en interligne, dans la marge, lacunes, etc. Il doit encore signaler ses propres interventions suppléant à des carences de tous les témoins, les mots, les phrases qu'il a dus éventuellement supprimer, ajouter ou corriger pour garder un sens au texte critique. Il doit enfin distinguer les mains qui ont écrit le manuscrit, c'est-à-dire signaler les changements d'écriture et d'encre qui montreraient qu'il y a eu plusieurs personnes, ou une même personne à des temps différents, intervenant sur la composition du manuscrit.

Dans le cadre d'une édition avec une tradition très riche, c'est-à-dire avec de nombreux témoins, dont les dates de composition peuvent être très étalées dans le temps, l'éditeur essaie de donner les liens de filiation qui existent entre les témoins, le *stemma codicum*. Ce travail peut d'ailleurs participer à la phase d'édition. Si l'on peut par exemple montrer de façon certaine qu'un des témoins est affilié directement à un autre, c'est-à-dire qu'il en est assurément une copie sans qu'il ait de rapport avec aucun autre témoin, il est courant que l'éditeur décide de ne plus en tenir en compte dans la fabrication de son texte critique, ce qui, dans le cas d'une tradition riche, peut simplifier de manière considérable son travail.

Les éditions scientifiques présentent des particularités par rapport aux éditions classiques. Les textes scientifiques ont souvent une forte structuration dont l'éditeur doit rendre compte. Les textes anciens font appels à de nombreuses références internes et externes. Les centaines de propositions des *Éléments* d'Euclide, par exemple, la bible des mathématiques et leur référence première durant vingt siècles, sont constamment citées, uniquement par un numéro, dans tous les travaux mathématiques jusqu'à l'avènement des mathématiques modernes. Ces références constantes sont évidemment extrêmement importantes pour l'historien des mathématiques et l'éditeur peut les répertorier, les classer et donner en extension ces citations. Ensuite, la quantité de figures et de nombres est importante dans les textes scientifiques et elles sont d'ailleurs deux sources d'erreurs supplémentaires et importantes pour les copistes. La question d'une édition critique des figures des textes scientifiques n'a jamais d'ailleurs été posée à notre connaissance. Enfin, les formules, le symbolisme pour des mathématiques plus modernes que celles du XVI^e^ siècle, sont une difficulté supplémentaire.

Le Mauro-*TEX*

Principes Le *Mauro*-TeX est un langage de balises, de *mark-up* permettant de construire pas à pas une édition critique. L'utilisateur transcrit les textes des différents té-

2. http://www.uni-tuebingen.de/zdv/tustep_eng.html
3. http://www.oeaw.ac.at/kvk/cte

moins en les réunissant dans un unique fichier LaTeX. Le langage permet de traiter tous les cas habituels rencontrés dans une édition comprenant un ou plusieurs témoins. Pour chaque situation donnant lieu à une ou des variantes dans les témoins, il signale consécutivement leur contenu dans les différents champs d'une macro dédiée (`\VV{}`). Quelques exemples suffisent pour en assimiler les principes. Soient trois témoins ***A***, ***B*** et *C* ayant comme leçons respectives :

A : Sit data gratia, sit datus cubus …
B : Sit data latio, sit datus cubus …
C : Sit data ratio, sit datus cubus …

La transcription est faite de la façon suivante :

```
Sit data \VV{{A :gratia}{B :latio}
{C :ratio}}, sit datus cubus ...
```

La sortie papier a alors la forme :

Sit data gratia[a], sit datus cubus …

a. gratia *A* latio *B* ratio *C*

Le premier champ de la macro `\VV{}` est par convention celui qui est retenu pour le texte critique ; les autres sont destinés à remplir l'apparat. Une simple permutation des champs permet de modifier le texte critique et l'apparat en conséquence. Ces informations suffisent à séparer le texte critique de l'apparat.

Pour traiter les différentes situations courantes rencontrées dans les textes, l'utilisateur dispose d'un certain nombre de raccourcis permettant de remplir automatiquement l'apparat critique avec les abréviations et arguments standards de la philologie. Supposons que dans le témoin *C*, le mot *gratia* barré vienne après le mot *ratio*. Le transcripteur utilise une macro prédéfinie `\POSTDEL` et écrit :

```
Sit data \VV{{C :\POSTDEL{gratia} :ratio}
{A :gratia}{B :latio}}, sit datus cubus ...
```

Et il est imprimé :

Sit data ratio[a], sit datus cubus …

a. *post* ratio *del*. gratia *C* gratia *A* latio *B*

Si le mot *ratio* apparaît comme un ajout en interligne :

```
Sit data \VV{{C :\INTERL :ratio}
{A :gratia}{B :latio}}, sit datus cubus ...
```

Le texte imprimé est :

Sit data ratio[a], sit datus cubus …

a. ratio *in interl*. *C* gratia *A* latio *B*

Il est bien entendu possible de remplir l'apparat critique avec ce que l'éditeur désire, surtout dans les cas moins courants qui n'ont pas été expressement prévus par le langage. Par exemple, s'il souhaite indiquer que le mot *ratio* est écrit à l'encre rouge dans le témoin *C* ; il fait appel à une macro `\DES{}` dont l'argument est libre :

```
Sit data \VV{{C :\DES{in rubro
atramento} :ratio}{A :gratia}
{B :latio}},sit datus cubus ...
```

Le texte imprimé sera :

Sit data ratio[a], sit datus cubus …

a. ratio *in rubro atramento* *C* gratia *A* latio *B*

Tous les cas courants possèdent une macro raccourci : omissions, lacunes, mots non lus, répétitions, transpositions, intégrations interlinéaires et marginales, corrections du copiste, conjectures et corrections de l'éditeur etc. Une codification des noms des témoins a été prévue pour décrire les différents mains ayant travaillé sur le manuscrit. Dans le cas de variantes longues, ne pouvant apparaître pour des raisons esthétiques *in extenso* dans l'apparat sans lourdeur, un mécanisme a été prévu permettant à l'utilisateur d'indiquer ou se trouvent le début et la fin du texte donnant lieu à une variante, auxquels le logiciel se réferera automatiquement. Enfin, il est possible de demander à ce que ne soient pas traitées certaines variantes *banales*. C'est le cas en particulier pour les fautes d'orthographe et les coquilles rencontrées : l'usage veut qu'on les signale dans l'introduction à l'édition mais qu'elles soient absentes de l'apparat pour ne pas l'alourdir inutilement. Le *mark-up* est identique à ce qui a été expliqué précédemment, sauf que la macro utilisée est `\VB{}` à la place de `\VV{}`. Il s'agit donc d'un moyen très simple pour choisir si la variante est significative ou banale et modifier automatiquement l'apparat en conséquence.

Le *mark-up* semble suffisamment complet pour réaliser une véritable classification des variantes utiles à l'éditeur et à l'historien et d'autres balises permettent de décrire formellement les témoins. Des macros plus élémentaires sont ainsi prévues pour insérer les changements de folios, marquer les dates, les titres, les citations et références, énoncés de propositions, etc.

Conséquences Un des axes de développement du langage est qu'il doit permettre de récupérer le contenu de chacun des témoins afin de parvenir à une présentation mettant en vis-à-vis deux ou plusieurs témoins ou le texte critique. L'intérêt pour le lecteur ou l'historien est immédiat : il peut ainsi visualiser facilement les différences et les correspondances des témoins comparés (ce procédé n'a pas encore été exploité dans le cas de l'édition Maurolico).

La procédure de *mark-up* présentée plus haut est adaptée aux évolutions des connaissances sur la tradition du texte édité. Il est aisé d'y intégrer un nouveau témoin, par exemple si un nouvel exemplaire inconnu venait à être trouvé. Si l'on a déjà fait l'édition de *n* témoins, un

$(n+1)^e$ témoin est facilement intégré et l'édition est automatiquement mise à jour par le déplacement éventuel des champs de la macro `\VV{}`.

L'édition Maurolico est particulière car la tradition des textes qui la composent est relativement pauvre. Les textes sont souvent autographes, souvent des *unicum* et quelques textes ont deux ou trois témoins et un seul quatre. Cette relative pauvreté présente l'avantage que transcription et édition peuvent être faites facilement en travaillant sur un unique fichier. De fait, l'utilisateur du *Mauro*-TEX collationne les textes l'un après l'autre, ajoutant aux balises `\VV{}` du fichier précédent un nouvel argument. Une tradition avec beaucoup de témoins rend la procédure bien plus délicate. Avec quatre témoins, le fichier devientpresque illisible. Quelques essais ont été faits pour modifier la procédure dans le cas d'une tradition avec beaucoup de témoins. Ils s'appuient sur un nouveau programme, un «mélangeur», que l'on fait agir après la collation des témoins l'un après l'autre à un texte de référence. Le mélangeur construit alors le fichier final contenant tous les témoins. La procédure simule de fait celle qu'emploie un philologue dans le cas d'une tradition très riche. Il commence par choisir et transcrire un texte de référence : un des témoins ou une édition déjà existante, selon ce qu'il juge le plus adéquat. Il collationne un premier témoin avec ce texte de référence. Il reprend le fichier contenant uniquement le texte de référence et recommence une collation avec un deuxième témoin. Et ainsi de suite. Chaque fichier a donc un *mark-up* lisible puisqu'il n'inclut que le texte de référence et un seul témoin. Après avoir fait la collation de tous les témoins, le «mélangeur» réunit tous les fichiers produits et construit automatiquement le fichier final qui contient le *mark-up* de tous les témoins.

D'un point de vue pratique, la procédure d'utilisation du *Mauro*-TEX peut être résumée ainsi : l'utilisateur transcrit et collationne les différents témoins. On traite ce fichier avec un programme, *m2lv*, servant de préprocesseur, qui permet d'obtenir un fichier LATEX intermédiaire qui soit compilable à l'aide des macros d'un fichier *mauro.sty* (Fig. 1).

On obtient finalement un résultat visualisable en *dvi*, PostScript, pdf et imprimable (Fig. 2).

Un deuxième programme *m2hv* permet de traduire le fichier source en fichier(s) HTML exploitable(s) sur l'internet (Fig. 3).

Le site `http://www.maurolico.unipi.it` de l'édition des œuvres mathématiques de Francesco Maurolico permet à tous de lire toutes les œuvres qui ont été transcrites à ce jour avec le *Mauro*-TEX. Celles-ci sont accompagnées de nombreuses pages d'introductions et d'autres utiles à l'évaluation des travaux et du personnage : catalogues des manuscrits, des imprimés, iconographie, etc. Le site internet permet d'accroître la visibilité internationale du projet et de montrer l'évolution en temps réel d'une édition ; la durée habituelle pour une telle quantité d'œuvres d'une édition peut être de quelques dizaines d'années. Enfin, il essaie d'exploiter au mieux les possibilités que fournissent les liens hypertextes pour naviguer entre les différentes œuvres.

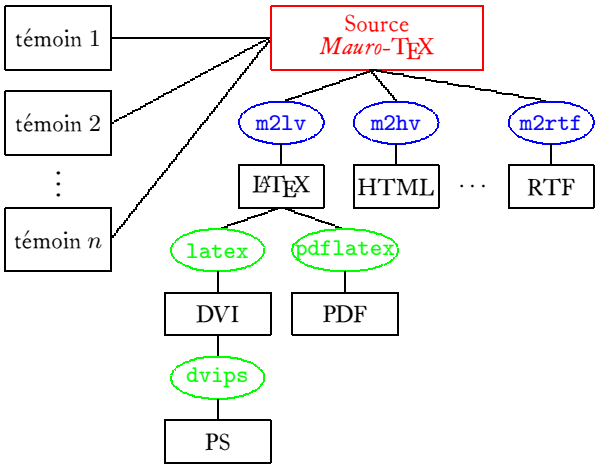


Fig. 1 : Le système *Mauro*-TEX

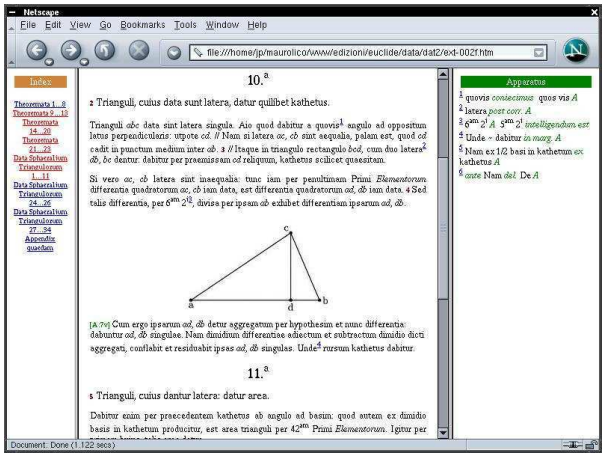
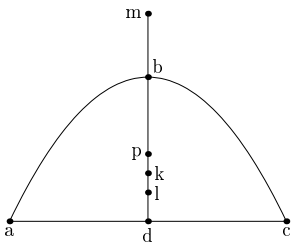


Fig. 3 : Une page HTML issue du système *Mauro*-TEX

Le langage et les outils informatiques construits autour peuvent aussi être considérés comme une aide à l'édition et un outil d'analyse. Pendant la phase de transcription, l'éditeur est guidé par le langage des macros-raccourcis. La classification des variantes induite qu'il suppose est aussi une aide à l'analyse des textes. Il est, de par la conception même du langage, facile de rechercher et de classer les variantes selon leur genre. Le cas le plus typique est la recherche des variantes qui décrivent les omissions. Leur analyse fournit en général la possibilité de construire le *stemma codicum*. Il est par exemple possible de visualiser dans des *frames* `html` à l'aide du programme *m2hv* toutes les omissions d'un témoins par rap-

centra scalae et portionum, quae sunt partes solidum integrantes. **70** Quod est absurdum, per tertiam | praemissarum propositionum. Non igitur cadet [A:13r] centrum solidi infra punctum k. //



71 Cadat, si possibile est, supra punctum k utpote in punctum p. **72** Atque punctum l ponatur centrum scalae cylindricae. Deinde dividatur axis toties, ut dictum est, constructis cylindris; ut cylindrica figura ad relictas portiones maiorem habeat proportionem, quam linea bp — linea pl utpote eam proportionem, quam habet linea mp — lineam pl. Hoc[28] enim possibile est per 11.am. **73** Cumque p sit centrum totius solidi: l vero centrum unius partium, scilicet scalae cylindricae: iam per 4.$^{\text{am}}$ praemissarum m erit centrum reliquae partis, scilicet relictarum portionum. **74** Quod per quintam praemissarum est impossibile. Quandoquidem centrum extra rei gravis ambitum cadere absurdum est. **75** Non igitur cadet solidi centrum[29] supra punctum k. Sed neque infra illud cadere posse ostensum est. Omnino igitur ipsum k punctum erit solidi centrum. **76** Quod demonstrandum proponebatur.

* Corollaria

77 <1.$^{\text{um}}$> Et quoniam per 12.$^{\text{am}}$ centrum scalae est per sextantem unius axium particularium inferius puncto k quod centrum esse totius nunc ostensum est. Sequitur ut centrum relictarum portionum per sextantem totalis axis sit superius centro scalae: sicut in triangulo fiebat. Propter proportionalitatem partium et totius in triangulo et solido sumptarum.

78 <2.$^{\text{um}}$> Sequitur etiam, tam in triangulo, quam in solido parabolico, ut sextans totalis axis ad sextantem unius axium partialium in ea sit proportione, in qua totale triangulum ad relictas portiones sive totale solidum ad relictas portiones. Quod per quartam propositionem et coniunctam proportionalitatem constat.

79 Et haec hactenus.

♄ 5.$^{\text{o}}$[30] maii 1565

[28] Hoc ~ per 11.am *signo posito in marg. A*
[29] *ante* solidi centrum *del.* punctum *A*
[30] *post* 5.° *del.* apr. *A*

Fig. 2 : Une page PostScript issue du système *Mauro*-TEX

port aux autres, ou toutes les omissions communes à plusieurs témoins. À l'aide d'un système de va-et-vient entre les références des omissions et les textes des témoins, il est possible d'analyser *de visu* l'omission et de vérifier la pertinence d'un lien entre les témoins. L'éditeur dispose ainsi d'un système semi-automatique d'aide à la constitution du *stemma codicum*.

Au début de notre travail, les macros LATEX du *mauro.sty* suffisaient à construire l'édition sans le préprocesseur *m2lv*. Au fur et à mesure du développement du projet sont apparus des besoins qui ont modifié la procédure. Le projet avait aussi pour but de construire un site internet contenant l'édition : il a donc été construit un *parser bison* pouvant interpréter la grammaire *Mauro*-TEX et un programme en C++, *m2hv*, pouvant la transformer en `html`. Les outils d'analyse des textes imaginés ont ensuite naturellement utilisé ce *parser*. Les fonctions d'éditions que l'on ajoutait au langage devenaient de plus en plus difficiles à programmer dans le langage propre aux macros LATEX, alors qu'elles devenaient faciles à insérer dans un ou des programmes en C++ utilisant le *parser*. Ainsi le *mauro.sty* a-t-il été supplanté par un autre programme, *m2lv*, et c'est aujourd'hui ce dernier qui est développé en priorité. À terme, le *mauro.sty* pourrait disparaître.

Évolutions

Le *Mauro*-TEX actuel est le résultat de cinq ans d'évolutions et d'adaptations aux besoins de l'édition Maurolico, et il commence à s'adapter à des situations pré-

sentant des caractéristiques philologiques différentes. Il est aujourd'hui à l'écoute des besoins des philologues qui voudraient l'utiliser et collaborer à son amélioration. Le *Mauro*-TEX et les programmes développés autour sont d'ailleurs sous licence libre GPL dans le but de partager et faciliter sa diffusion. Plusieurs évolutions ont déjà été expérimentées.

L'apparat est pour l'instant géré grâce à des appels de notes : un mot du texte critique pour lequel une variante existe dans un témoin ,est suivi d'un appel de note en exposant, et l'apparat a la forme d'une succession de notes. Cette solution est due au fait que le projet s'est pour le moment surtout concentré sur la sortie internet. Pour la sortie imprimée, la meilleure présentation pour une édition critique est sans doute celle qui consiste à référencer dans l'apparat les variantes par numéro de ligne ou par paragraphe. Le projet Maurolico a commencé à développer une gestion par paragraphe pour la sortie imprimée comme pour l'internet.

Les textes scientifiques présentent, comme cela a déjà été mentionné, la particularité d'avoir de nombreuses références internes et externes, la plupart du temps dans les démonstrations faisant appels à d'autres propositions déjà démontrées. Expliciter ces références dans un apparat différent de l'apparat critique fait partie des évolutions futures du *Mauro*-TEX pour l'édition Maurolico. Ainsi chaque référence à, par exemple, *per* 5^{am} 2^{i}, donnera lieu à un apparat fournissant *in extenso* l'énoncé de cette proposition d'Euclide. Cette fonction demande bien sûr que tous les appels de citation soient marqués dans le texte et la constitution d'une base de données des citations des auteurs les plus cités par Maurolico. Le *mark-up* a déjà été réalisé pour tous les textes transcrits jusqu'à présent et la constitution de la base des références est en cours. On pensera ensuite à construire des outils d'analyse des citations ; ne serait-ce que la possibilité de faire le lien inverse, c'est-à-dire étant donnée une citation quelconque, d'Euclide par exemple, trouver tous les endroits de toutes les œuvres de Maurolico dans lesquels cette proposition est utilisée, serait extrêmement utile aux historiens des sciences.

Les moyens électroniques et l'internet permettent maintenant de fournir plusieurs niveaux de lecture des textes. Le site internet de l'édition Maurolico permet pour le moment d'avoir deux niveaux : le premier est une lecture du texte critique seul, le deuxième présente le texte critique avec son apparat et les notes de l'éditeur. Plusieurs autres niveaux seront proposés dans le futur et, à terme, on pourra avoir les choix suivants :

1. texte critique seul ;
2. texte critique avec apparat critique géré par paragraphe ;
3. texte critique avec apparat critique, notes éditoriales, apparat des citations, le tout géré par paragraphe ;
4. *idem* mais géré avec des appels de notes en exposant afin de faciliter les recherches et le va-et-vient entre texte critique et apparats ;
5. textes de témoins mis en confrontation les uns à côté des autres.

En complément, un système de visualisation et mise en correspondance des textes édités et des images des manuscrits a été expérimenté. Ce système, basé sur les liens hypertextes du langage `html` et dont le résultat est visualisé avec un simple navigateur internet, peut ainsi être considéré comme une aide à l'édition, puisque l'éditeur peut vérifier immédiatement et facilement ses choix. On se heurte cependant pour la diffusion aux coûts particulièrement dissuasifs des droits de reproduction demandés par les bibliothèques qui conservent les manuscrits.

Mais la plus grande mutation envisagée est l'abandon de LATEX en tant que langage de *mark-up*, ce pourquoi il n'est pas vraiment fait d'ailleurs. Il a semblé plus logique de muter le *Mauro*-TEX vers un langage du type *xml*. LATEX ne deviendrait plus qu'un intermédiaire permettant la production de fichier PostScript et pdf et l'impression sur support papier.

Conclusion

L'avantage premier du langage et du système *Mauro*-TEX est une certaine simplicité, et l'on retrouve sans surprise les arguments qui ont fait le succès de LATEX : un simple éditeur de texte est suffisant pour travailler ; il est extensible à l'infini ; l'apprentissage en est facile (le fait a pu être vérifié de nombreuses fois dans le cadre du projet Maurolico) ; il répond assez bien à la règle des 80/20 : seulement 20% des commandes sont utiles pour 80% du travail. Le *Mauro*-TEX semble de ce point de vue un langage *sain*, dans le sens où sa complexité d'utilisation est directement proportionnelle à la complexité de la situation : il est simple à utiliser dans des situations simples, plus complexes dans des situations plus complexes.

Le souhait du projet Maurolico est aujourd'hui d'en faire un outil universel de développement, d'analyse et d'édition des textes critiques. Pour qu'il réponde à tous les besoins des éditeurs, il faudra qu'il améliore l'aspect graphique de ses sorties papier, qu'il puisse travailler dans plusieurs langues, quelque soit leur sens d'écriture, avec tout type de document, classique ou scientifique et tout type de tradition. Un éditeur graphique et un navigateur dédiés pourront aider à sa diffusion dans le milieu savant. Une intégration avec les programmes et les machines utilisés pour l'impression pourront aider à sa diffusion dans l'imprimerie. S'il conserve une licence libre, il pourrait alors devenir dans quelques années un standard de l'édition critique.

Inserting External Figures with GraphicP

Péter Szabó
Budapest University of Technology and Ecomomics
Department of Analysis
Műegyetem rakpart 3–9.
Budapest
Hungary H-1111
pts+eurotex@math.bme.hu
http://www.inf.bme.hu/~pts/

Abstract

This paper describes GraphicP, a new, unified LaTeX and plain TeX package that provides a fast and reliable method for including external images into TeX documents. The `\includegraphics` macro of GraphicP is a drop-in replacement of the same command of LaTeX graphics.sty and graphicx.sty, but with many enhancements. Drivers for xdvi, dvips, pdftex and dvipdfm are included. Useful tips are given for converting vector and bitmap images into a format usable for inclusion (typically EPS or single-page PDF).

Résumé

Cet article présente GraphicP, un nouveau paquettage LaTeX/*plain* pour insérer de manière rapide et efficace des images externes dans des documents TeX. La macro `\includegraphics` de GraphicP, remplace au pied levé la commande homonyme de `graphics` et `graphicx`, avec un grand nombre de nouvelles fonctionnalités. Des pilotes pour *xdvi*, *dvips*, *pdftex* et *dvipdfm* sont également prévus. Nous donnons un certain nombre de conseils autour de la conversion d'images vectorielles et bitmap vers un format de fichier utilisable pour l'insertion d'images (typiquement EPS ou PDF mono-page).

Overview of image inclusion

There are many different technologies to embed external images, such as photos, figures, function plots and diagrams into TeX documents. Most methods involve the following steps:

1. designing (drawing) the image in an external program;
2. converting it to a file format recognised by TeX;
3. loading an embedder (a TeX package that can embed images);
4. placing inclusion commands at places of the `.tex` document where the image should appear;

There are additional, less common steps:

5. providing feedback to the image drawing program about the final image size and position;
6. replacing fonts and glyph sizes in the image to match the main text font, compensating for the effect of scaling and allowing TeX math formulas;
7. changing line widths to compensate for scaling.

In some technologies, these steps are integrated, so the non-WYSIWYG image code can be typed directly into the `.tex` document. Examples are the LaTeX portable picture environment and its variants; epic, eepic, gastex, Xy-pic, MFpic, ConTeXt's inline METAPOST environment. This paper discusses only the generic process of embedding external images, not these integrated, specific solutions.

In WYSIWYG word processors, images are usually inserted by opening them in their appropriate application, and using the clipboard to copy them to the document inside the word processor. Scale, rotate, move and resize operations are performed with the mouse, providing instant feedback to the author about the final appearance of the image and how it affects text flowing around it. Compared to this, the steps above seem to be overcomplicated, and a real pain to the author. This is partly because easy handling of images needs a WYSIWYG environment with instant feedback while editing. TeX — by design — lacks both of them. So inserting images to TeX documents is expected to be a tedious process, no matter how sophisticated the tools that are used. Nevertheless, it is worth improving the tools, so TeX can compete with other document preparation methods. Emphasis should be put on quality, stability, compatiblity and output size, not on ease of use.

This paper describes steps 2–4 in detail, using tools traditional for years, including Ghostscript, dvips,

dvipdfm, pdftex, METAPOST, the convert utility of ImageMagick, The GIMP, as well as replacement tools from the new GraphicP package, available from CTAN. The a2ping.pl utility replaces epstopdf and others, img_bbox.pl replaces ebb from dvipdfm, pdfboxes.pl improves PDF files, graphicp.sty replaces traditional LaTeX graphics.sty and plain TeX epsf.tex. These new tools work together with the sam2p [1] raster image converter, which replaces tiff2ps and the EPS and PDF export filters of other image processing software such as convert. This paper compares the new and the replacement tools in detail, and proposes a more reliable and accurate general image embedding technology for both LaTeX and plain TeX, using the proper combination of these programs.

The embedder. TeX doesn't understand image files. In particular, TeX cannot extract an individual pixel, line or label from an image. The only reason why the embedder macros running in TeX read the image file is that they need the bounding box for proper scaling. The bounding box (or *bbox*) is a rectangular area of the image file that contains all visible parts of it. The bbox is of the form ⟨llx⟩ ⟨lly⟩ ⟨urx⟩ ⟨ury⟩, in which the point at (⟨llx⟩, ⟨lly⟩) is the lower-left corner of the image, and the point at (⟨urx⟩, ⟨ury⟩) is the upper-right corner. It is a common tradition to have ⟨llx⟩ = 0, ⟨lly⟩ = 0, ⟨urx⟩ = image width, ⟨ury⟩ = image height. Once the bounding box has been extracted, and the desired width of the included image is known, the embedder can calculate the actual height and leave empty space for the image on the paper.

The embedder inserts the image file name, the computed horizontal and vertical scale factors into the DVI file as a `\special`. The printer driver is responsible for loading the image file and sending it to the printer properly scaled and rotated. The image file format must be compatible with the printer driver. For example, the popular printer driver dvips requires all images to be in the Encapsulated PostScript (EPS) format. pdftex accepts PDF, PNG, JPEG and TIFF images. dvipdfm accepts PDF, PNG, JPEG and METAPOST EPS images. Usually the author of the image prefers a different file format for development, so conversion is necessary. Some converters are safe, efficient and faithful in the sense that they create valid and small output without information loss, but others have to be used with great care.

The most important requirements for an embedder are:

1. ability to specify the image size and/or scaling
2. ability to specify rotation angle and mirroring
3. ability to clip unnecessary parts (crop)
4. extracting the bounding box properly from all file formats
5. full compatibility with dvips, pdfTeX, dvipdfm
6. compatibility with other printer drivers
7. reuse of the same image object if embedded multiple times
8. running text around the image
9. support for floating figures

Requirement 9 has been well supported for a long time in LaTeX, by the `table` and `figure` environments. However, requirement 8 is solved only, and with limitations, in floatflt.sty. The other requirements are correlated more strongly, they are implemented in the de facto standard for an embedder of LaTeX: graphics.sty (includes both `graphics.sty` and `graphicx.sty`). It is well-documented, and it provides a convenient syntax and unified (printer driver and file format-independent) interface for including any kind of rectangular image into LaTeX documents. Using it is rather easy:

```
...
\usepackage[dvips]{graphicx}
... \begin{document} ...
\begin{figure}
  \includegraphics[width=0.9\textwidth]%
    {footown.eps}
  \caption{The map of Footown}
\end{figure}
... \end{document}
```

graphics.sty is not available for plain TeX, but there is a similar but less powerful package (specific to EPS images), epsf.tex. An example:

```
\input epsf  % in plain \TeX
...
\epsfxsize=0.9\hsize \epsfbox{footown.eps}
```

Why GraphicP? graphics.sty contains several inconsistencies and weaknesses, which document authors must care for when embedding images. Some of these problems are just annoying quirks, others affect the image placement and scaling, visible to the reader. GraphicP intends to be a stable and accurate replacement for graphics.sty, fixing many problems, but without changing the syntax of the `\includegraphics` command substantially. It doesn't contain fundamental additions, only small fixes and additions, and increased consistency and portability.

As GraphicP evolved, it has been extended with Perl scripts, for example a2ping.pl and other standalone programs in addition to the TeX macros in graphicp.sty. These external programs are optional, because a Unix system is needed by some of them.

graphics.sty implements a framework, separating the general high-level functionality from the printer-driver specific low-level one. For example, the command `\usepackage[pdftex]{graphicx}` loads the graphicx.sty user interface with the pdftex.def driver. (The

other interface, graphics.sty, differs in the syntax of the `\includegraphics` command.)

This separation is a good design choice in general, but it makes it more difficult to do fundamental changes. Another drawback of graphics.sty is that it is tightly integrated into LaTeX, so it would be quite difficult to add plain TeX compatibility. GraphicP has been implemented from scratch. The name of its embedder, pgraphicp.sty, suggests that it can be used as a replacement of graphics.sty. For example, the LaTeX 'Footown' example above works by replacing the first line with `\usepackage[dvips]{graphicp}`.

Preparing the image for inclusion

PostScript is a two dimensional page description language that can fully describe the visual appearance of printed material. A PostScript document is composed of straight lines, curves, filled regions delimited by these, text, bitmap images and others. PDF is a portable document format mainly for distributing two dimensional, mainly static material in electronic form. The same graphics model is used in PostScript and PDF, so — in the ideal case — there is no loss of information or precision when converting between PostScript and PDF.

Other important differences, such as sheet trimming, colors, slide shows and web hyperlinking are not considered in this document.

Any image (bitmap, vector and combined) can be faithfully represented in PostScript and PDF. The EPS (Encapsulated PostScript) file format is a single-page PostScript document with some other minor restrictions, so it is perfectly suitable for representing inline images in a document. There is no restriction in PDF; any one-page PDF can be treated as an image. Most TeX printer drivers accept the images in either EPS or PDF, so the goal of this section is to convert any image to both of these formats. Note that TeX and dvips cannot embed normal (non-encapsulated) PostScript documents, because the bbox is missing, and PostScript documents may contain device dependent or global state changing operators. Use a2ping.pl to convert PS to EPS.

Most Windows users working with a word processor haven't heard of PostScript, and have never used PDF for embedding. The Windows clipboard and OLE hide the file format; as long as the image can be copy-pasted, its format doesn't matter. This user-friendly approach is not available in TeX, beacause it is technically impossible to copy-paste binary image data into the human-readable `.tex` source. Thus each image has to be saved into a separate file, and the `.tex` file contains only references to these files in the form of `\includegraphics` commands. The file format depends on the printer driver; it should be — in general — EPS for TeX combined with dvips, and PDF for pdftex and dvipdfm.

Conversion to EPS or PDF. EPS and PDF files are not easily editable, so they should be converted or exported by the preferred image drawing appliaction of the author. Most modern vector graphics editing programs, including Illustrator, CorelDRAW, Visio, Acrobat, InDesign and QuarkXPress have a direct EPS export feature; some of them can export PDF.

One has to consider the quality, the compatibility and the size of the exported image. Some programs build up circles using a constant number of straight lines or cannot emit glyphs in vector format (⇒ low quality), some images contain proprietary or legacy junk, or they need new features of PostScript LanguageLevel 2 or 3 not supported by old printers (⇒ low compatibility), some programs emit 100 kB of procedure sets that are not used anyway, or they represent objects inefficiently (⇒ big size). Thus it is worth knowing more than one way to do the same file format conversion. Unfortunately, there is no golden rule: the best EPS and PDF output can be found only by experimenting.

If the original image is in raster format, the sam2p command-line utility can be used to create a small and compatible PDF or EPS file. sam2p — with the help of tif22pnm, png22pnm and djpeg — can read today's most popular raster image formats. The author should save the image in an intermediate format (recommended: PNG or TGA), and feed that to sam2p.

The first page of a normal PostScript document can be converted to EPS or PDF with a2ping.pl, part of GraphicP, for example: `a2ping.pl in.ps out.eps`. For the PS→PDF conversion, a2ping.pl calls Ghostscript with the device `pdfwrite`. Ghostscript 7.00 or later is recommended to avoid missing objects and low-quality glyphs. The PDF output is quite small. As an alternative, Acrobat Distiller can be used for converting PS to PDF, but it is not free, and the settings should be specified properly to create a small and compatible PDF.

There is a general EPS export method for any Windows and Macintosh application that is able to print. It needs the PostScript printer driver freely available from Adobe [10]. The PostScript printer description should be `adist4.ppd` [11]. Select the following features during installation: `adist4.ppd`, print to file, optimize for compatibility (ADSC), PS LanguageLevel 2, embed all the Type 1 (vector) and TrueType fonts as Type 1 fonts, embed even the standard fonts. One may choose between PS (DSC) or EPS output to avoid rotation. Scaling is not important, because the TeX embedder is able to re-scale images. Translation can be compensated by editing the bounding box comment by hand. After setting it up, any document can be printed from any application to a PostScript file — it works similarly to normal printing, but a file will be created on disk. The bounding box can then be modified by hand, and a2ping.pl may be applied if

necessary. If the application can create EPS files itself, it should be compared to the printer-driver-based generic method.

Adobe distributes a utility named PDFWriter that can be used from any Windows application to print to a PDF file. Using it is not recommended, because it completely confuses accented glyphs, and even other glyphs in some fonts.

PostScript has been the traditional page description language on Unix systems for a long time, so most Unix utilities have PostScript output. The most important of these are:

- Acrobat Reader can print to PostScript, so it can be used as a PDF to PS converter. In the File/Print dialog, select *File* and *Level 2 Only*, uncheck *Fit to Page* and *Download Far East fonts*, check *Download Fonts Once* and *Use Printer's Halftone Screen*.

  As an alternative, invoke:
  `acroread -toPostScript -level2 ...`
  from the command line.
- Ghostscript can convert PostScript to PDF. The epstopdf utility by Thomas Esser makes this easier, but a2ping.pl is a better replacement.
- `dvips -E` creates EPS files, but it sometimes computes the bounding box wrong, so a combination of `dvips` and `a2ping.pl` is necessary.
- Figures created by XFig can be converted to EPS with a command like `fig2dev -L eps in.fig > out.eps`. Also XFig can create EPS files, and newer versions can even make TEX typeset some of the labels: The *Special Flag* should be set to *Special* in the *Text Edit panel*, and the file exported as *Combined PS/LATEX*, with name `out.eps`. Then `\input out.eps_t` should be called. Unfortunately it is impossible to post-scale the image this way.
- Both Netscape Navigator and Mozilla can print to PostScript, but their output is rather ugly. (The printed output of Internet Explorer is not so ugly, but tables are often cropped at right.)
- The GIMP can print to both PostScript and EPS, but sam2p is usually a better solution.
- pdftops from xpdf can convert PDF to PS and EPS. Unfortunately it doesn't work with weird fonts or encodings well, and older versions simply discard PK fonts embedded by pdftex.

The output of METAPOST doesn't need conversion, because it's already EPS, and the PDF-specific drivers in GraphicP can embed it in a PDF document without external converters. It is possible to embed TEX output into TEX: the PDF files created by pdftex can be included directly, and the PostScript output should be run through a2ping.pl to create EPS.

Sometimes a multi-step conversion produces the best results. For example, the author of this article often uses the pipeline of PostScript printer driver, Acrobat Distiller, Acrobat (cropping and EPS output), Acrobat Distiller, pdftops to create a small and portable EPS file that can be embedded.

Content management. Additional information has to be remembered for each image: proposed image title (caption), the documents or floating figures containing the image, the file name, the editable source file of the image, how it was converted from its source, is it available in another format (e.g. both EPS and PDF), the `\label` the figure will have, etc.

Keeping this meta information up to date is important if either the images or the document containing them are planned to be reused in the far future. The author of the document should decide how to face this organization task. Only some hints are provided here.

On Unix systems it is traditional to write a `Makefile` for compilation, even document compilation. The enough_tex.pl script is provided in the GraphicP distribution for convenience: it runs TEX enough times to resolve all references and indices. It can be inserted into a `Makefile` instead of bare `latex` invocations. `Makefiles` may also automate image conversion, creating EPS and/or PDF from the source images before compiling the document.

It is wise to retain an EPS, PDF, PNG or JPEG copy of each image, so they can be opened decades later, because tools that can open these formats with the same semantics as today are expected to be available for a long time. On the other hand, proprietary and closed file formats should only be used for temporary storage — if the company stops supporting the file format, it will be impossible to open such images later.

The same is true for TEX texts: one should make a backup of all classes, styles and auxiliary macro files, including those that were used to create the format file, plus all source and image files belonging to the specific document. A full backup eliminates the risk that a `.tex` source doesn't compile anymore, or it compiles with different line breaks. The Linux `strace` utility can list all files opened by `latex` and other programs.

Features of GraphicP

Runs on both plain TEX and LATEX. It contains a compatibility layer (laemu.sty) in plain TEX that provides the LATEX package loading, error and warning indication, and some other simple common macros. Care has been taken to use plain TEX constructs whenever possible, e.g. `\def` instead of `\newcommand`. During development and testing, I have been using plain TEX, because porting it to LATEX is almost trivial compared to the other direction.

More accurate calculations. Knuth has designed TEX not to use floating point numbers. This is an important portability advantage, because different rounding implementations of different CPUs don't affect the positions of line and page breaks calculated by TEX. It would be a painful headache, for example, if the same TEX document compiled differently on the author's Linux system and the publisher's Solaris system.

Real numbers in TEX are represented in 15.16 signed fixed point notation. That is, 15 bits are reserved for the integer part and the sign, and the precision is 2^{-16}, so a rounding of up to 2^{-17} pt may occur after each operation. Fixed point arithmetic has the important advantage that additions and subtractions are always accurate.

But what about multiplication and division? When scaling an image of size $wd \times ht$ to the desired width dwd, the actual height is calculated as $ht \cdot dwd/wd$. Multiplication and division are equally important operations in image scaling, and often the result of the multiplication exceeds the maximum TEX dimension of about 16000 pt.

TEX provides real number multiplication with the following trick:

```
\dimen0=42pt
\dimen1=0.333333333\dimen0
\showthe\dimen1
```

The calculated result (13.99979pt) is not accurate. Another weakness of the built-in multiplication is that it introduces errors larger than the minimum 2^{-17} pt. For example $0.00001\,\text{in} = 0.0011\,\text{pt}$, but $0.000005\,\text{in} = 0.0\,\text{pt}$.

The TEX primitives are not suitable for scaling; because multiplication is not accurate enough, it results in overflow, and there is no built-in real number division at all. So a high precision, non-overflowable scaling operation has been implemented from scratch in div16b.sty. Its internal number representation is 30.32 signed fixed point, and it stores a number in two TEX `\count` registers. Input and output values are still 15.16 real numbers. Due to the increased precision, overflow can never occur, and the multiplication is always accurate, even $1\,\text{sp} \cdot 1\,\text{sp}$ isn't truncated. The division routine does repeated subtraction, possibly doubling or halving the divisor after each subtraction. Halving may introduce small internal rounding errors, but fortunately no error occurs when the scaling ratio is $a \cdot 2^b$, where a and b are integers.

Empirically, the error of the scaling algorithm in practical image sizes (100 .. 3000 pt) is 0 .. 2 sp, while the result of `\Gscale@div` in graphics.sty deviates sometimes as much as 1 pt, which is noticeable. Another drawback of graphics.sty is that it calculates different widths inside normal latex and pdflatex.

Enforced dimensions. When the user specifies `[width=` or `[height=`, these will be enforced (using `\hss`), irrespective of what the driver generates. graphics.sty doesn't have this feature, and considering the less accurate image scaling, differences up to a few pt can occur, which can seriously affect further line and page breaks in the document.

Gives bbox hints. It is possible to convert a TEX page to EPS with `dvips -E`. dvips calculates the bounding box of the page automatically, taking into account the glyphs and rules on the page. Unfortunately, version 5.86e still detects the image bounding box wrong, especially with images descending below the baseline. GraphicP works around the problem by forcing dvips to exclude the image from the calculation, and adds two small invisible (white) rules at the corners. This feature can be disabled.

A similar problem occurs in xdvi, which sometimes crops too much from the edges of the image. This is solved by forcing it not to crop at all. Cropping can be controlled from `\special{PSfile=...}`, but printer drivers interpret it differently, so the specification emitted by graphicp.sty disables cropping completely. This also solves a similar problem with dvipdfm, which would otherwise crop EPS images below the baseline.

File format detection. As opposed to graphics.sty, GraphicP doesn't rely on the the file name to determine the file format. Files with bogus or invalid extensions are treated properly, and EPS files created by METAPOST (`img.1`, `img.2` etc.) are also embedded correctly. The annoying bug of graphics.sty of failing to recognise `an.image.pdf` as `.pdf` is also eliminated.

File format decisions (implemented in pts_bbox.sty) are based on the first four bytes read from the file. GraphicP can distinguish between PNG, TIFF, JPEG, MPS (EPS created by METAPOST), EPS, DOS EPSF and PDF properly.

External bbox parsing. graphics.sty can read tiny `.bbx` files that contain only a `%%BoundingBox` comment. This is faster and more accurate than full image parsing, because `.bbx` files have extremely simple syntax. GraphicP accepts `\graphicPmeta` commands instead of `.bbx` files. Each of these commands describes a single image file, for example:

```
\graphicPmeta{i.1}{EPS.MPS}{0}{0}{99}{534}
\graphicPmeta{i.2}{EPS.MPS}{8}{9}{10}{76}
```

A list of these lines can be inserted right into the `.tex` document before typesetting the images, or it can be `\input` from a separate file (proposed extension: `.gpm`).

A Perl script called img_bbox.pl is included in GraphicP to generate these lines. For example, use the command `img_bbox.pl -tex *.eps *.pdf *.jpg *.tiff *.png >all.gpm`. The script supports

more than 42 image formats, included all formats embeddable by graphicp.sty. File names may contain TEX control characters; they are properly escaped.

Getting the bbox of a PDF (the /MediaBox) is rather complicated, because it is deeply hidden somewhere in the page tree of the binary PDF file. Neither GraphicP, nor graphics.sty can do this reliably when running inside tex; both of them expect the /MediaBox to be in a line on its own, and they can be confused when there are multiple such lines. The solution is to run GraphicP inside pdftex, or — to get all four bbox coordinates — to use img_bbox.pl, which can navigate the PDF page tree properly.

As an alternative, GraphicP contains another Perl script, pdfboxes.pl, which modifies an existing PDF file so that the bounding box will be available right in the beginning. This is essentially a proof-of-concept implementation: it proves that it is possible to insert a new object into a PDF file and modify all offset references to other objects, without the need to parse and regenerate the full PDF. sam2p 0.43 and above emits the bounding box early enough, so pdfboxes.pl is not needed.

a2ping.pl can detect all three bounding box types, the setpagedevice and other PostScript operators.

Internal bbox parsing. The bbox extraction capability of graphicp.sty is limited by the fact that TEX reads files line-by-line, thus it is hardly possible to parse a binary file properly. \catcodes are used extensively to ignore most of the binary "junk", so the dimensions of PNG, TIFF and JPEG files cannot be extracted, and PDF parsing is very limited. Fortunately, pdftex provides primitives to extract the bounding boxes of these binary files, and dvips doesn't support these file formats anyway.

All three types of EPS bounding boxes are supported; the user can choose between the exact and the rounded bbox, if both of them are present. The default is to choose Exact, then Hires, then normal (rounded to integer) bounding box. This can be overridden by [hiresbb] and [exactbb].

Nonzero depth. Images can be lowered below the baseline, for example:

```
\includegraphics[lower=20]{t.eps}
```

moves the image down by 20 bp. Also GraphicP can recognise negative lower-left in the bbox, and move the image below the baseline automatically (only with [below]). Horizontal movement is not possible, but the user can insert the appropriate \kern commands before and after the image.

Raster images are always aligned onto the baseline, so an explicit [lower=] or [raise=] should be used instead of [below] to move them vertically. Alternatively, sam2p has the -m:lower:⟨dimen⟩ option that creates a pre-lowered EPS or PDF from the raster image.

a2ping.pl automatically raises images up to the baseline, unless the --below option is given.

Avoids duplication. dvipdfm and newer versions of pdftex both support Form XObjects, a means for reusing material already typeset. GraphicP uses Form XObjects to embed an image file only once, no matter how many times it appears in the document. This optimization is impossible in PostScript documents, because they are read sequentially, and caching images already read imposes a high risk of memory shortage.

METAPOST with all drivers. Although METAPOST generates EPS, these text files follow such a simple structure that they can be converted to PDF drawing operators (\pdfliteral) within TEX, as done in ConTEXt's supp-pdf.tex, written by Hans Hagen. GraphicP loads this routine in case such an MPS image is to be loaded. However, these macros consume a lot of TEX memory, so they can be disabled by saying

```
\usepackage[nopdftexmpost]{graphicp}
```

In fact, METAPOST output is the only image format that is supported by all drivers of GraphicP. It is recognised by an ADSC comment in the EPS header; the file name doesn't matter — it can be t.1, t.eps, t.ps or anything else.

METAPOST doesn't emit a high resolution bbox by default, but context/mp-tool.mp adds the appropriate code to extra_endfig. Alternatively, one can type this into the beginning to the .mp file:

```
extra_endfig := extra_endfig & "special ("
& ditto & "%%HiResBoundingBox: " & ditto
&"&decimal xpart llcorner currentpicture&"
                    & ditto & " " & ditto
&"&decimal ypart llcorner currentpicture&"
                    & ditto & " " & ditto
&"&decimal xpart urcorner currentpicture&"
                    & ditto & " " & ditto
&"&decimal ypart urcorner currentpicture"
& ");";
```

METAPOST creates EPS files with the wrong extension. When reading ajob.mp, the figure under the scope of beginfig(42) will have the file named ajob.42. Passing a negative number of beginfig will create ajob.ps. These settings are hard-wired into METAPOST, thus the files have to be renamed before inclusion with graphics.sty. GraphicP doesn't have this limitation.

Better Babel compatibility. Many Babel languages make the characters " and ‘ active. This conflicts with the METAPOST loader and other PDF-specific code graphics.sty borrows from ConTEXt, so

```
\usepackage[pdftex]{graphicx}
\usepackage[magyar]{babel}
```

is the correct loading order. Users of GraphicP don't have to care, because it uses external code only for METAPOST EPS to PDF conversion, wrapped by proper `\catcode` resets, so no error occurs.

Backward compatibility. Although the TEX and LATEX interfaces used haven't changed much in the last few years, pdftex is under development. GraphicP adjusts itself to the version of pdftex running it. GraphicP has been tested with pdftex 0.12r (Debian Slink), 0.14 and 1.00a (Debian Woody).

graphics.sty is not Unix-specific; it should work in any architecture or OS to which TEX has been ported, but the Perl scripts provided in the GraphicP distribution currently require a Unix system. They can be ported to other platforms easily if there is considerable interest.

For educational purposes, graphicp.sty (which provides `\includegraphicP`) and graphicx.sty (which provides `\includegraphics`) can be loaded in this order.

Draft support. It is often desirable to omit images from the document, especially in the development phase where fast compile–redisplay cycles are of primary importance. Of course, the space for the images still has to be reserved. Contrary to graphics.sty, GraphicP supports this kind of draft mode by specifying the appropriate driver, e.g.:

```
\usepackage[driver=invisible]{graphicp}
```

The supported draft drivers are:

invisible a transparent rectangle is displayed without the image

blackbox a solid black box is displayed, showing the bounding box — wastes a lot of ink

frame a black rectangular frame shows the bounding box

namedframe the image file name is displayed in the black rectangular frame. Uses `\textan` (in asciiall.sty) if available to display weird characters in file names.

The real drivers are:

pdftex the default driver when pdfTEX is detected. Cannot display EPS images.

dvi a common subset of *dvips* and *dvipdfm*. This is the default when running under normal (non-pdf-)TEX. Can display EPS and MPS only.

dvips cannot display PDF and raster images.

dvipdfm calls Ghostscript to convert EPS to PDF automatically.

The file `texmf/dvipdfm/config` should be updated to improve bounding box calculation and others during EPS to PDF conversion (don't forget to enter the actual path to a2ping.pl):

```
D "zcat -f %i | ./a2ping.pl --below - %o"
```

EPS fixups. a2ping.pl detects and emits all three bbox types with proper rounding, removes DOS EPSF binary junk, removes HP UEL header, adds some ADSC comments, can read and write from a pipe, converts PS to EPS, calls sam2p to convert raster images to EPS, removes form feed from end of EPS.

For the PDF→EPS direction, a2ping.pl invokes pdftops. a2ping.pl can output multiple-page PS, PDF and HP PCL5 documents, with corrected paper size (forced), resolution, duplex and tumble settings. The ouput of Ghostscript is post-processed if necessary.

a2ping.pl can shift images, so the lower-left corner is in the origin, but it can also retain the original bbox. This works for both EPS and PDF.

a2ping.pl is not only a converter, but it contains many fixup routines, so it can be used to fix EPS files from a source incompatible with dvips.

Work pending

cropping Now `\includegraphics` always embeds the whole image. It should have a `clip=` option, just as in graphics.sty.

transformations all 8 combinations of flipping and rotation by 90° should be added.

imtrix a unified way for replacing labels in EPS and PDF images by those generated by TEX. Would be similar to psfrag.sty.

compatible options `\includegraphics` should have the following options, with the same meanings as in graphics.sty: `bb=`, `totalheight=`, `keepaspectratio=`, `type=`, `ext=`, `read=`, `viewport=`.

optional bbox dots the 1 sp wide white dots that forcibly mark the bounding box of the image should be made optional.

arithmetic more complex arithmetic expressions than `width=0.9\textwidth` should be allowed in the `width=` and similar options.

testing with images originating from various programs.

Conclusion

The most important benefit of TEX is that it helps authors and typesetters to produce beautiful printed documents. Although inserting figures in TEX documents isn't easy, TEX helps us to make the images consistent and pretty. There are serious quirks and limitations during production, conversion and inclusion, but once the image has been included properly, it remains there without accident: it won't overlap the bottom margin (unless explicitly requested), it will be numbered and floated properly, it will never be torn from its caption, etc. The total size of the images doesn't affect TEX: it runs happily (albeit slowly) on a book with thousands of large images, and never crashes. Beyond creativity, the authors must

have the technical knowledge to create documents with images, but they can also enjoy many benefits unique in computer typography.

The author uses several tools when dealing with images: image editors, converters, the embedder and the printer driver. It is essential that these tools work properly and they can communicate with each other. GraphicP provides an embedder implemented from scratch that gives more flexibility to the author and instructs the printer drivers in a more compatible way than the traditional LaTeX embedder, graphics.sty. GraphicP also contains many converters that fill the gap between the various output file formats of the powerful image editors and the formats the printer drivers work well with. The converters do not enhance the visual appearance of the image, but they ensure that bounding box and orientation information is emitted properly, and they also do some syntactical changes. They try to work smartly, without image-specific instructions from the caller.

Existing printer drivers are fairly good, provided that the embedder gives them specific and correct instructions what to do. However, existing WYSIWYG image editors cannot cooperate well with TeX. Their output often has to be adjusted by hand, or by using specific converters.

The aim of graphicp.sty is not to compete with or replace graphics.sty, but to provide a proof-of-concept alternative showing that some of its functionality can be implemented better. The key, unmatched benefits of graphics.sty are the framework approach, support for many printer drivers, and the extent of testing. Extending graphics.sty with the features of GraphicP while retaining these benefits would be a glorious, but enormous, work.

The scripts and other programs of GraphicP are, as far as its author knows, unique. They can be used together with both graphics.sty and graphicx.sty, and even for generic image processing purposes unrelated to TeX. GraphicP is hoped to increase the efficiency of everyday image processing tasks done by TeX authors and publishers.

References

[1] Péter Szabó. *Inserting figures into TeX documents*. In proceedings to EuroBachoTeX 2003.

[2] Hàn Thế Thánh, Sebastian Rahtz and Hans Hagen. *The pdfTeX user manual*. teTeX:doc/pdftex/base/pdftexman.pdf.gz, 1999.

[3] Mark A. Wicks. *Dvipdfm User's Manual*. teTeX:doc/programs/dvipdfm.dvi.gz, 1999.

[4] Tomas Rokicki. *Dvips: A DVI-to-PostScript Translator*. teTeX:doc/programs/dvips.dvi.gz, 1997.

[5] D.P. Carlisle. *Packages in the 'graphics' bundle*. teTeX:doc/programs/grfguide.ps.gz, 1999.

[6] Keith Reckdahl. *Using Imported Graphics in LaTeX 2ε*. teTeX:doc/programs/epslatex.ps.gz, 1997.

[7] Adobe Developer Support. *PostScript Language Document Structuring Conventions Specification, Version* 3.0. Adobe Developer Technologies. 1992.

[8] Ed Taft, Steve Chernicoff and Carline Rose. *PostScript Language Reference*. Addison-Wesley, 1999.

[9] Jim Meehan, Ed Taft, Steve Chernicoff and Carline Rose. *PDF Reference. Second edition*. Addison-Wesley, 2000.

[10] http://www.adobe.com/support/downloads/main.html, select *PostScript printer drivers*.

[11] PPD for a generic printer. http://www.uniprint.ee/Web/OpenResource.aspx?ResFile=47 and http://www.rgraphics.com/downloads/ADIST4.PPD

Geometric Diversions with TEX, METAFONT and METAPOST

Karel Horák
Mathematical Institute of the Academy of Sciences
Žitná 25, 115 67 Praha 1,
Czech Republic
horakk@math.cas.cz
http://www.math.cas.cz

Abstract

Recently METAPOST has been preferred to METAFONT, although this need not be the general case. Each can be used, with its own specific advantages, and readable source text can be written for both great compilers! We provide some advice on how to do it in such a way that practically no change in incorporating corresponding figures into a document is necessary. This includes typesetting labels using Alan Hoenig's labtex macros, which can be done independently even if graphics produced by METAPOST instead of METAFONT are then used in the document. Some examples of geometric diversions are given.

Résumé

Ce dernier temps METAPOST est de plus en plus utilisé, vis-à-vis de METAFONT, ce qui n'est pas indispensable. En effet, on peut utiliser chacun des deux dans des cas spécifiques et profiter de leurs avantages respectifs ; d'autre part, on peut écrire du code lisible par les deux compilateurs ! Nous proposons un certain nombre de conseils pour ce faire, de manière à ne pas changer la manière dont une figure est insérée dans un document. Cela comprend également la composition de légendes aux figures en utilisant les macros labtex d'Alan Hoenig, ce qui peut être fait indépendamment du fait si les graphiques ont été produits par METAPOST au lieu de METAFONT. Nous proposons également un certain nombre de divertissements géométriques.

Introduction

There are many ways to assign labels to selected points of a picture. One easy example is given in the *TEXbook* (p. 389), which introduces the macro `\point`. Using variable `\unit` (or possibly `\xunit` and `\yunit`, why not?) one can label any inserted graphics and change the position of its labels dynamically, should the picture be rescaled, by changing `\unit`. But if we are concerned with pictures prepared by METAFONT or METAPOST, there are more convenient possibilities as well.

A better way is to use communication between METAFONT and TEX via a *tfm* file. There are two basic possibilities on how to store coordinates of selected points: via fontdimens or via kerning. Both of these ideas belong, as far as I know, to Alan Hoenig (see [2,3]).

The first way has one important drawback: the number of fontdimens is very limited. The other way (`labtex.mf` and `labtex.tex` macros) gives the user much more freedom to store coordinates. This very short macro file uses for that task kerning with the first three characters (it can be easily modified to use more if necessary, as METAFONT's limit on the number of kerning pairs is now more than 32,000).

Inspired by [3], Alan Jeffrey proposed in [5] another way to manipulate labels of METAFONT diagrams. Because of deficiencies in the file-handling capabilities of METAFONT he used the *log* file for communicating between METAFONT and TEX, extracting the desired information with the *grep* utility.

Clearly, METAPOST can solve all these problems easily and smoothly, as it has own labelling mechanism which uses TEX for typesetting labels, which are then converted into low-level METAPOST commands. Does this mean that we should rewrite all our old programs written for METAFONT, as we did when we changed from the old METAFONT 78 to the new one? Not at all! John Hobby, author of METAPOST, has arranged for all METAFONT sources (with some exceptions) to compile with METAPOST using a special format file `mfplain.mp`.

While there are many good reasons to prefer METAPOST to METAFONT (use of colour, no problems with higher resolution, etc.), I am still preparing many black and white documents with geometric pictures printed in normal resolution and I am accustomed to use both: METAFONT, when proofing, and only then (if necessary) METAPOST for the final compilation. For such a reason I rather often prefer to use labtex (after ten years of

excellent experience) for labels as it works in both cases without any substantial change. The aim of this short contribution is to stress that labtex can also be used with METAPOST without any serious problem.

It could be done directly: METAPOST stores the metric information into a tfm file exactly as METAFONT does; the only drawback is then the great number of trivial files generated. So it seems better to produce pictures by METAPOST *without* labtex, while the appropriate metric file with related kerning information given by labtex's pointing commands will be regenerated by METAFONT without any garbage later. As we want to use the same source file twice, we presuppose using `mfplain.mp` as the format. It can be done by the following fork (testing if colour *red* is known):

```
mode_setup;
u#:=1mm#;
define_pixels(u);
if known red:
    % labtex information is not used
def labtex (text t)= enddef;
warningcheck:=0;
else:
input labtex
fi
```

A similar fork can be used for any case where METAFONT and METAPOST behave differently (culling of some region could be alternatively changed to clipping, etc.).

The pros and the cons

One big advantage was mentioned before: one can easily change old sources from bitmap to outline without too much work.

For me, another advantage of this attempt is that I can easily preview typeset pictures (without possible colours) with the emTEX previewer, while previewing PostScript with Ghostscript is substantially slower (even if much faster now than some years ago).

Another advantage of this concept is that one can easily combine METAPOST output with other *eps* pictures, a problem that was solved only by some special tricks thanks to Bogusław Jackowski and his colleagues.

One possible disadvantage (mentioned in [5]) is that this concept does not involve communication from TEX to METAFONT (e.g. size of the label to leave some void space in colored region). Such a feature could of course be added using ideas from [2], if one wished to typeset many more such labels.

One of the advantages of METAPOST's own labelling mechanism is that one can easily transform the labels (rotate, scale, skew...), use color, erase appropriate surrounding space of label, etc. These of course could be done with labtex using TEX `\special` commands and PostScript macros (PSTricks can do all of this).

Using labtex with METAPOST

What is the main difference between a character generated by METAFONT and the corresponding PostScript picture produced by METAPOST(&mfplain)? Their dimensions are known to TEX! Dimensions (width, height, and depth of each character are given by parameters of `beginchar` in the METAFONT source file and one must take some care to guess these dimensions as best as possible, or even better to construct the picture depending on them (i.e. using variables w, h, d for defining basic points of any figure). On the other hand, dimensions in eps files output by METAPOST are read by TEX from the corresponding bounding box, which is computed by METAPOST according to the real dimensions of the resulting picture. For alternative use in TEX it is in most cases sufficient to correctly position the reference point of the picture. So we must first change the lower left corner of the bounding box to be at the origin.

The labtex macros are designed to typeset the picture first, then labels. This is even more important for the PostScript variant of the `\fig` macro, as PostScript draws the picture sequentially layer after layer, so labels typeset in advance could disappear completely under some coloured part of the picture.

The following adaptation of Hoenig's `\fig` macro can do the right job, changing the reference point of the resulting picture.

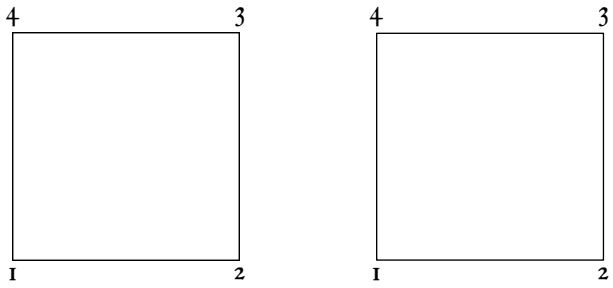
```
\input labtex
\input epsf
\newcount\epsfurX   \newcount\epsfurY
\def\MPfig#1{% usage:
    %%%%\MPfig{figure number}{labels}
    %%%%\endfig
 \figcount=#1 \catcode`\^^M=5
          %%%%to be sure in obeylines
 \setbox\figbox=
    \hbox{\labtex\char\figcount }%
 \global\figwd=\wd\figbox
 \global\fight=\ht\figbox
 \global\figdp=\dp\figbox
 \setbox\figbox=
    \hbox{\epsffile{\fontname\labtex.#1}}%
    \epsfurX\epsfurx\epsfurY\epsfury
 \setbox\figbox=
    \hbox{\epsffile[0 0 {\the\epsfurX} %
    {\the\epsfurY}]{\fontname\labtex.#1}}%
 \zerobox\figbox%
 \hbox\bgroup\box\figbox
       %%%%% now add \point'ing commands
       %%%%% and labels
 \vrule width0pt height\fight depth\figdp}
\def\zerobox #1{\ht#1=0pt \dp#1=0pt \wd#1=0pt }
```

Of course, one should never forget to compile the source in the right order! Using labtex, the order of labels is also important, as we have mentioned earlier (i.e. we must typeset labels in the same order as the labtex macro stores coordinates of their anchor points).

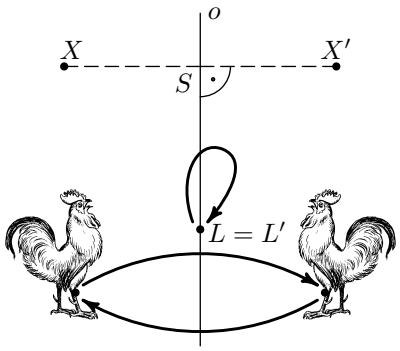
Once the picture is ready, one can easily correct the labels only in the TEX file without recompiling the METAFONT source.

Examples and comments

Next you can see two identical trivial pictures of a square with its vertices labeled by 1, 2, 3, and 4. The first is typeset with a bitmap font, the other with an eps file produced with METAPOST from the same source.



Another example shows an illustration from [6] using two instances of a scanned picture converted to eps:



When testing figures using Alan's original macros, I rediscovered (after some delay, of course) a somewhat strange feature: figures must be numbered sequentially starting from zero, which actually I am not accustomed to. A small change in the original code removes this necessity. Instead of

```
if not known chars[i]:
   beginchar(i,0,0,0); endchar;
   if not known fpf: fpf=i; fi
fi
```

one can find the number of the last known figure without any such assumption:

```
if not known chars[i]:
   beginchar(i,0,0,0); endchar;
   else: fpf:=i;  % last known figure
fi
```

On the other hand, if I am preparing pictures for anybody who or does not use TEX at all, or does not want to manipulate METAFONT sources, the most secure way is to prepare pictures, typeset them and then convert to EPS using another useful utility of Gdańsk provenance (the `ps_conv` package of P. Pianowski & B. Jackowski).

To my own sorrow I have not succeeded in learning more about ConTEXt and its way of manipulating METAPOST graphics. Then I could probably forget about PSTricks and others... Of course it surely cannot happen without making progress to use a real operating system (here I do not think of Windows of any kind). I am going to be twice as old as TEX is, perhaps this is the right time to make substantial progress...

TEX and METAFONT are genuinely designed to cooperate and there are many ways to enjoy it.

The last example (on the following page) is known to all my friends who, for the last six years, have received every New Year a calendar to be cut out and glued together to form a polytope.

References

[1] John D. Hobby: A User's Manual for METAPOST. `http://ctan.org/tex-archive/graphics/metapost`.

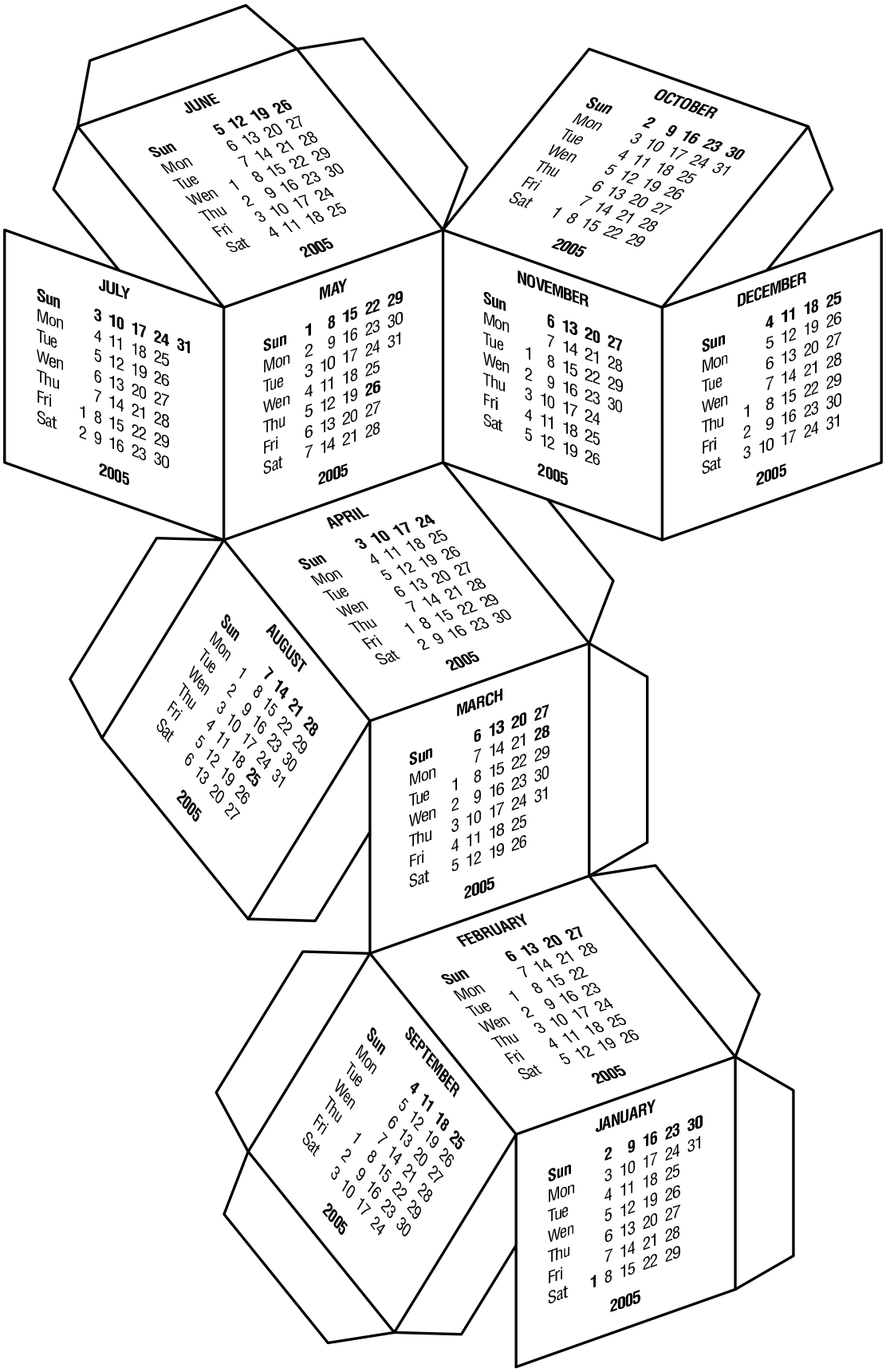
[2] Alan Hoenig: When TEX and METAFONT work together. Proceedings of the 7th European TEX Conference, Prague, 1992.

[3] Alan Hoenig: Labelling figures in TEX documents. *TUGboat* 12(1), 1991, pp. 125–128.

[4] Alan Hoenig: TEX Unbound. LATEX & TEX Strategies for Fonts, Graphics, & More. Oxford University Press, Oxford-New York, 1998.

[5] Alan Jeffrey: Labelled diagrams in METAFONT. *TUGboat* 12(2), 1991, pp. 227–229.

[6] František Kuřina: 10 geometrických transformací (10 geometric transformations; in Czech). Prometheus, Praha, 2002.



Printing Digital Photographs with LaTeX

Frédéric Boulanger
Supélec — Service Informatique
Plateau de Moulon, 3 rue Joliot-Curie
F-91192 Gif-sur-Yvette cedex
Frederic.Boulanger@supelec.fr
http://wwwsi.supelec.fr/fb/

Abstract

Affordable digital cameras and photo inkjet printers make it possible to obtain digital prints with a quality which is generally as good as, and often better than, a compact film camera and standard processing. However, printing many photographs in various numbers of copies on standard-size paper may be cumbersome. There are programs to build albums and print them, but they generally force you to lay out photographs by hand on each page. The photoprint LaTeX document class is an attempt to automate the printing of digital photographs on standard-size paper. It uses a list of photographs tagged with the number of copies to print, divides the paper into "print slots", and takes care of rotating, cropping or scaling the photographs to make them fit into the slots. It can also build an index of the photographs to help people choose the prints they want, add crop marks to the printed pages to help cut the prints, or overprint the date on the prints.

Résumé

Des appareils photographiques numériques et des imprimantes à jets d'encre de prix abordables permettent aujourd'hui d'obtenir des tirages papier d'une qualité équivalente, sinon meilleure, à celle obtenue avec un appareil argentique compact et un tirage standard. Toutefois, l'impression en quantité de photographies en nombre de copies variable sur du papier de taille standard devient rapidement malaisée. Il existe des programmes permettant de placer des photos dans un album pour l'imprimer ensuite, mais ils exigent généralement un placement manuel de chaque photographie. La classe de documents LaTeX photoprint est une tentative d'automatisation de l'impression de photographies sur du papier de taille standard. Elle s'appuie sur une liste de photographies associées au nombre de copies désirées et sur un découpage de chaque feuille en « emplacements d'impression ». Elle gère la rotation, la mise à l'échelle et le rognage des photographies pour les faire tenir au mieux dans les emplacements. Elle permet aussi d'obtenir un index des photographies qui facilite le choix des retirages, peut imprimer des marques repères pour le découpage, et afficher la date en surimpression.

Introduction

Printing photographs from a family wish list can keep you busy for a whole weekend: you must find the photographs to print, lay them out on pages, taking the total number of copies of each into account, rotate them if they do not match the page orientation, resize and crop them so they match the paper size, wait for the printer, and finally cut the pages into prints. Of course, you can use special photo-paper with pre-cut margins, but it is more expensive than standard-size paper, yields not so clean edges, works only for a given brand of printers because its margins are adapted to the print area of the printer, and limits you to one print size.

Using standard-size paper (for instance A4 paper) gives you more freedom since you can use it for prints from 10×14 cm up to 20×28 cm. You can even use smaller prints, for instance to make stamps printed on self-adhesive paper. But this freedom comes at the cost of doing the layout of the photographs on the paper.

Regular digital photographs have a $\frac{4}{3}$ aspect ratio, while A4 paper has a $\sqrt{2}$ aspect ratio. So, to use as much as possible of the expensive sheets, the photos must be cropped. Some photos are in portrait orientation (their longer side is vertical), while others are in landscape orientation (their longer side is horizontal), so the photos which don't have the same orientation as the paper must be rotated.

The photoprint LaTeX document class divides pages into "print slots" and lays out photographs on these pages. It reads the names of the photographs and the number of copies of each from a file. This file can be written by hand, created by a shell script, or generated by a helper Java application which provides a contact-sheet-like view of the photographs from which you can select pictures

and choose the number of copies.

Since most digital cameras yield JPEG files, it is much easier to use photoprint with pdflatex. However, nothing in photoprint is hardcoded for pdflatex, so if you provide bounding box files for your JPEG pictures so that graphicx can determine their size, you can use photoprint with latex.

Basic setup

photoprint is a regular LaTeX document class, so you just write:

```
\documentclass[a4paper,landscape]{photoprint}
```

to use it with A4 paper in landscape orientation. Then, you have to specify the size of the printable area of your printer. Since this does not depend on the orientation of the paper, photoprint uses the top, left, bottom and right margins from the edges of the paper in portrait orientation to describe the print area. The command used to set the print area is:

```
\photopaper{⟨top⟩}{⟨left⟩}{⟨bottom⟩}{⟨right⟩}
```

For instance:

```
\photopaper{0.5cm}{0.5cm}{1cm}{0.5cm}
```

says that your printer can print as close as 0.5 cm from the top edge, 0.5 cm from the left edge, 1 cm from the bottom edge, and 0.5 cm from the right edge of a sheet of paper. These are the default settings if you don't use the \photopaper command.

The bottom margin is generally larger than the other margins because the printer must still be able to hold the paper between its rolls while printing the bottom of the page.

If you print in landscape orientation, the bottom margin will be on the left of the printed page. You can consider that \photopaper takes a physical description of the print area in which top is the side which comes out of the printer first, bottom is the side which comes out last, and left and right are defined relative to top and bottom, without consideration of the printing orientation.

Next, you have to decide how you want to divide the sheet of paper. For this, you give the number of print slots per line (so it is the number of columns), and the number of print slots per column (the number of lines). You set the number of photographs on a line with the:

```
\photosperline{⟨number⟩}
```

command, and the number of photographs in a column with the:

```
\photospercol{⟨number⟩}
```

command. For instance:

```
\photosperline{2}
\photospercol{2}
```

sets four print slots. This is the default setting which yields about 10×14 cm prints on A4 paper.

To print a contact sheet, in landscape orientation, you could use:

```
\photosperline{5}
\photospercol{4}
```

which makes for 4 rows of 5 pictures on each page. However, since printing contact sheets is a common operation, photoprint has a command for this which does more than printing the pictures.

Last, in the body of your document, you will lay out the pictures, using as many pages as necessary, with the command:

```
\printphotos[⟨filename⟩]
```

⟨*filename*⟩ is the name of the file which contains the list and number of copies of the pictures to print. It defaults to photoprint.job. To use the list photosfordad.job, write:

```
\printphotos[photosfordad.job]
```

The format of the photoprint.job file is very simple: each line contains the name of a picture file, optionally followed by a * and the number of copies to print. For instance, the following asks for one copy of 001-23-IMG_0375.jpg, two copies of 004-20-IMG_0949.jpg, one copy of 006-22-IMG_1126.jpg, and no copy of the other pictures:

```
001-23-IMG_0375.jpg*1
002-28-IMG_0425.jpg*0
003-27-IMG_0707.jpg*0
004-20-IMG_0949.jpg*2
005-20-IMG_1091.jpg*0
006-22-IMG_1126.jpg*1
007-25-IMG_1313.JPG*0
008-05-IMG_1549.jpg*0
```

If you want contact sheets, use \contactsheet instead of \printphotos. \contactsheet will ignore any \photosperline and \photospercol settings and print four rows of 5 pictures per page, with a header, and the name and index of each photo printed below it. You may obtain strange results if you don't use landscape orientation when printing contact sheets. When printing contact sheets, photoprint ignores the number of copies requested for each photo, and puts each photo once on the contact sheet.

Convenience files

Some settings are so common that two files are defined to allow printing photos without writing any LaTeX commands at all. The file photos.tex prints four pictures per page in landscape orientation:

```
\documentclass[%
  a4paper,
  landscape,
  cropmarks
]{photoprint}
```

```
\photosperline{2}
\photospercol{2}
\begin{document}
  \printphotos%[name-of-photo-list]
\end{document}
```

With the contents of the `photoprint.job` file as shown above, running

```
pdflatex photos.tex
```

yields the result shown in figure 1.

The file `largephotos.tex` prints two pictures per page and yields 14 × 20 cm prints on A4 paper:

```
\documentclass[%
  a4paper,
  landscape,
  cropmarks
]{photoprint}
\photosperline{2}
\photospercol{1}
\begin{document}
  \printphotos%[name-of-photo-list]
\end{document}
```

With the contents of the `photoprint.job` file as shown above, running

```
pdflatex largephotos.tex
```

yields the result shown in figure 2.

The file `contactsheet.tex` prints contact sheets for the pictures listed in `photoprint.job`. It ignores the number of copies requested and makes each photo appear once on the contact sheet. With the contents of the `photoprint.job` file as shown above, running

```
pdflatex contactsheet.tex
```

yields the result shown in figure 3.

For Unix systems (Linux, Mac OS X, and others), photoprint is distributed with two shell scripts. The `mkphoto` script builds a `photoprint.job` file containing the list of all JPEG files in the current directory, with the number of copies set to 0. This script considers files to be JPEG if their extension is `.jpg`, in either lower or upper case. The `mkcs` script builds a contact sheet from the `photoprint.job` file in the current directory. It uses the name of the current directory as the title of the constact sheet.

When installing photoprint, since docstrip adds a dot at the end of the name of both scripts when they are extracted, you should rename these scripts and make them executable with the `chmod` Unix command. For instance `chmod a+x mkcs` will make `mkcs` executable by any user.

Options

In the examples of the previous section, you can see black lines in the margins of the pages. This is the effect of the `cropmarks` option. These marks help adjust the position of the page on the paper cutter. The default is not to print crop marks.

If you want white margins around your prints, use the `whitemargins` option. The length of the smallest margin of the printable area is used as the white margin, making for an even white border around the prints when they are cut. Figure 4 illustrates the effect of the `whitemargins` option.

Some photographs may contain important details even at their edges, and cropping them to adapt their aspect ratio may lead to bad results, with important details removed. The `fullpictures` option prevents photoprint from cropping pictures. This makes less optimal use of the paper since the aspect ratio of the pictures can no longer be adapted to the print slots, as shown in figure 5.

The `landscape` option makes photoprint use the paper in landscape orientation (largest edge is horizontal). It should always be used for contact sheets. When printing photographs, the `landscape` option does not change the aspect of the prints. It only changes the way they are laid out on the pages.

The `showdate` option prints the date in the lower right corner of the pictures, which may be useful later to remember on which occasion a picture was taken. The date is read from a file which should be named `!Name_Map.can` or `!Name_Map.mav` and contains information about the photographs. The name and format of this file are set for historical reasons from the tools I use to manage my digital photographs. Each line of the file describes a photograph with four items delimited by braces:

1. The number of seconds since January 1, 1904 (the epoch on the Macintosh). This item is not used by photoprint and may be left empty (but must be present).
2. The file name of the photograph.
3. The file name of the original picture as taken by the camera. This item is not used by photoprint and may also be left empty.
4. The date and time when the photograph was taken, in ISO format:
   `YYYY-MM-DDThh:mm:ssZ`
   where `YYYY` is the year, `MM` the month (`01` for January, `12` for December), `DD` the day of month, `T` the separator between date and time in the ISO format, `hh` the hour (`00`–`23`), `mm` the minutes (`00`–`59`), `ss` the seconds (`00`–`59`), and `Z` the time zone. The letter `Z` indicates UTC time. It may be replaced by the signed offset of the time with regard to Universal Time (for instance `+01:00` is the offset of French local time in winter). Everything after the `T` is ignored by photoprint.

For instance, the file used to store information about the example pictures has the following contents (because of

Fig. 1: Result of `pdflatex photos.tex`

Fig. 2: Result (first page only) of `pdflatex largephotos.tex`

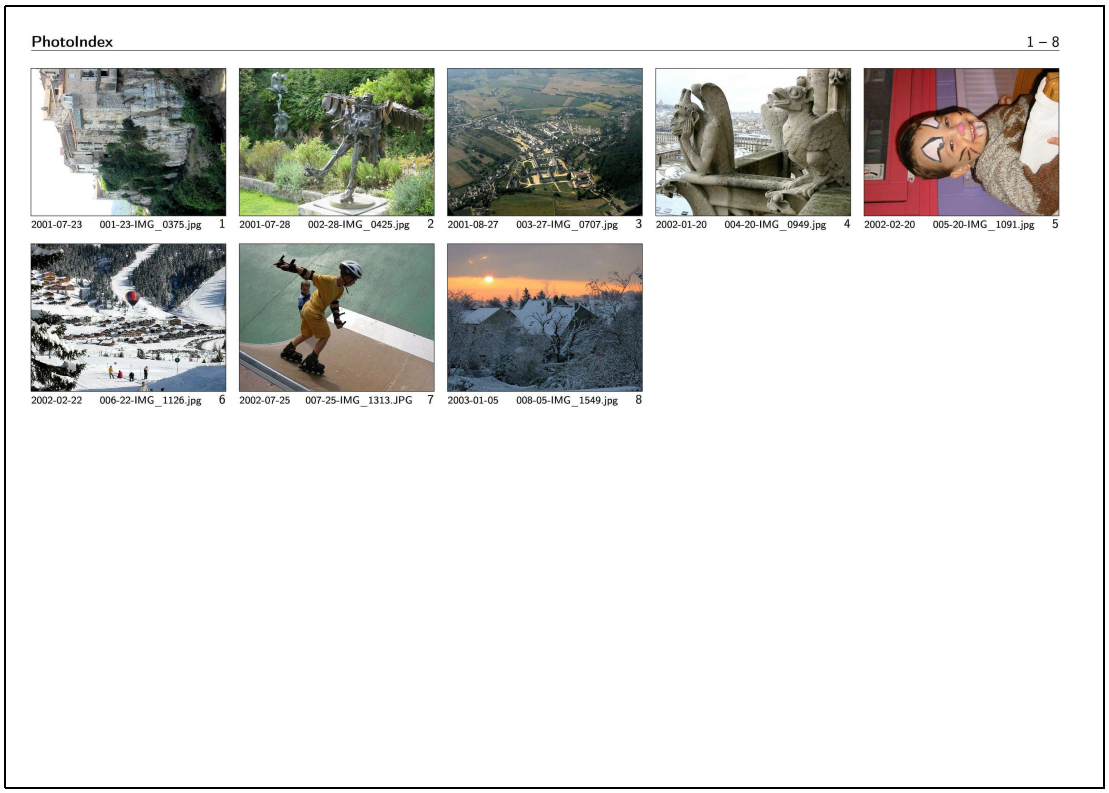


Fig. 3: Result of `pdflatex contactsheet.tex`

Fig. 4: Effect of `whitemargins` option

narrow width of this column, we have broken lines with a \):

```
{3078749666} {001-23-IMG_0375.jpg} {} \
    {2001-07-23T16:14:26+01:00}
{3079183448} {002-28-IMG_0425.jpg} {} \
    {2001-07-28T16:44:08+01:00}
{3081773514} {003-27-IMG_0707.jpg} {} \
    {2001-08-27T16:11:54+01:00}
{3094393350} {004-20-IMG_0949.jpg} {} \
    {2002-01-20T17:42:30+01:00}
{3097077318} {005-20-IMG_1091.jpg} {} \
    {2002-02-20T19:15:18+01:00}
{3097226436} {006-22-IMG_1126.jpg} {} \
    {2002-02-22T12:40:36+01:00}
{3110471962} {007-25-IMG_1313.JPG} {} \
    {2002-07-25T19:59:22+01:00}
{3124602926} {008-05-IMG_1549.jpg} {} \
    {2003-01-05T09:15:26+01:00}
```

The effect of the `showdate` option is shown in figure 6 where unusual settings have been chosen to make the date very visible (`\datecolor{red}` and `\datestyle{\Huge \sffamily\bfseries}`).

Other options (for instance `a4paper`) are passed to the article document class.

Finer tuning

Several commands allow for changing the default behavior of photoprint:

`\datecolor{`⟨color⟩`}` sets the color used to print the date on pictures when the `showdate` option is active. The default is `black`.

`\datestyle{`⟨style⟩`}` sets the text style used to print the date on pictures when the `showdate` option is active. ⟨style⟩ should be a series of text style declarations. The default is `\sffamily\small`.

`\infostyle{`⟨style⟩`}` sets the text style used to print the information about the pictures on contact sheets (date and name). ⟨style⟩ should be a series of text style declarations. The default is:

```
\sffamily\footnotesize
```

`\cstitle` If this command is defined, its value will be used as the title of the contact sheet by the `\contactsheet` command. For instance:

```
\newcommand{\cstitle}{Summer 2002}
\contactsheet
```

will produce a contact sheet with the title "Summer 2002".

`\photohmargin{`⟨length⟩`}` sets the amount of horizontal white space left around pictures. Consecutive pictures will be separated horizontally by twice this length. The default value is 0 pt (or the smallest margin of the print area when the `whitemargins` option is active).

`\photovmargin{`⟨length⟩`}` sets the amount of vertical white space left around pictures. Consecutive pictures will be separated vertically by twice this length. The default value is 0 pt (or the smallest margin of the print area when the `whitemargins` option is active).

`\photomargins{`⟨length⟩`}` sets the amount of white space left around pictures, both vertically and horizontally. Consecutive pictures will be separated by twice this length. The default value is 0 pt (or the smallest margin of the print area when the `whitemargins` option is active).

Requirements

The photoprint document class relies on the standard LaTeX document class article, on the standard LaTeX packages calc, ifthen, graphicx, color and fontenc. It also needs the geometry package.

Although photoprint does not make any assumption on the TeX engine used, and uses the graphicx package to include pictures, it is much easier to use with `pdflatex` since digital photographs are generally in JPEG format.

The tasks of photoprint

photoprint performs several tasks to prepare pages of prints from the list of photographs and selected layout:

1. From the dimensions of the print area, the orientation of the paper, and the number of photographs to print per page, photoprint computes the size and position of the "print slots" (the areas where pictures will be placed).
2. From the list of photographs and number of copies to print, photoprint builds a list of pictures to lay out.
3. For each picture in this list, photoprint compares the orientation of the picture and the orientation of the current print slot. If they differ, the picture is rotated by 90°.
4. The picture is then scaled to fit into the print slot.
5. When the last print slot on a page has been used, a new page is started.

Determining print slots photoprint considers only the print area of a page, and divides it according to the required numbers of photos per line and per column. The horizontal and vertical margins around prints are also taken into account to determine the size of the print slots. Figure 7 shows how print slots are determined in landscape orientation, with two photos per line and per column.

Reading the list of photos The default name for the file which contains the list of photographs is `photoprint.job`. In this file, each line should be formatted as:

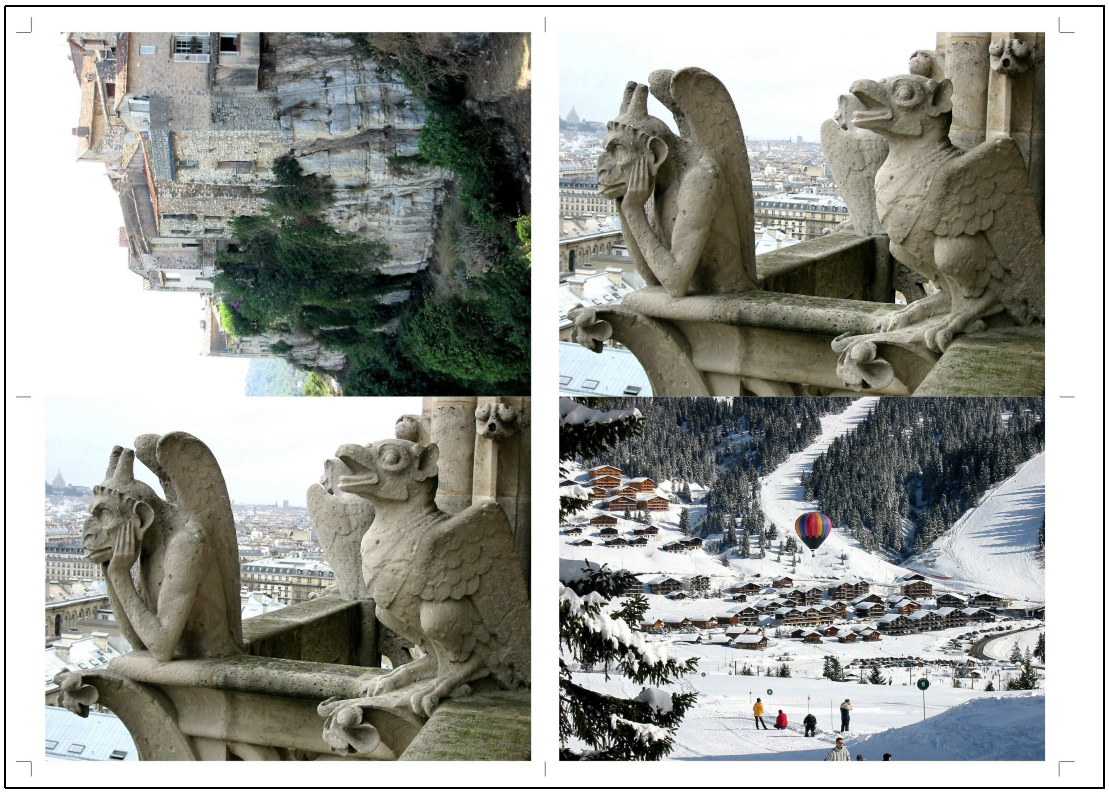


Fig. 5: Effect of the `fullpictures` option

Fig. 6: Effect (exaggerated) of the `showdate` option

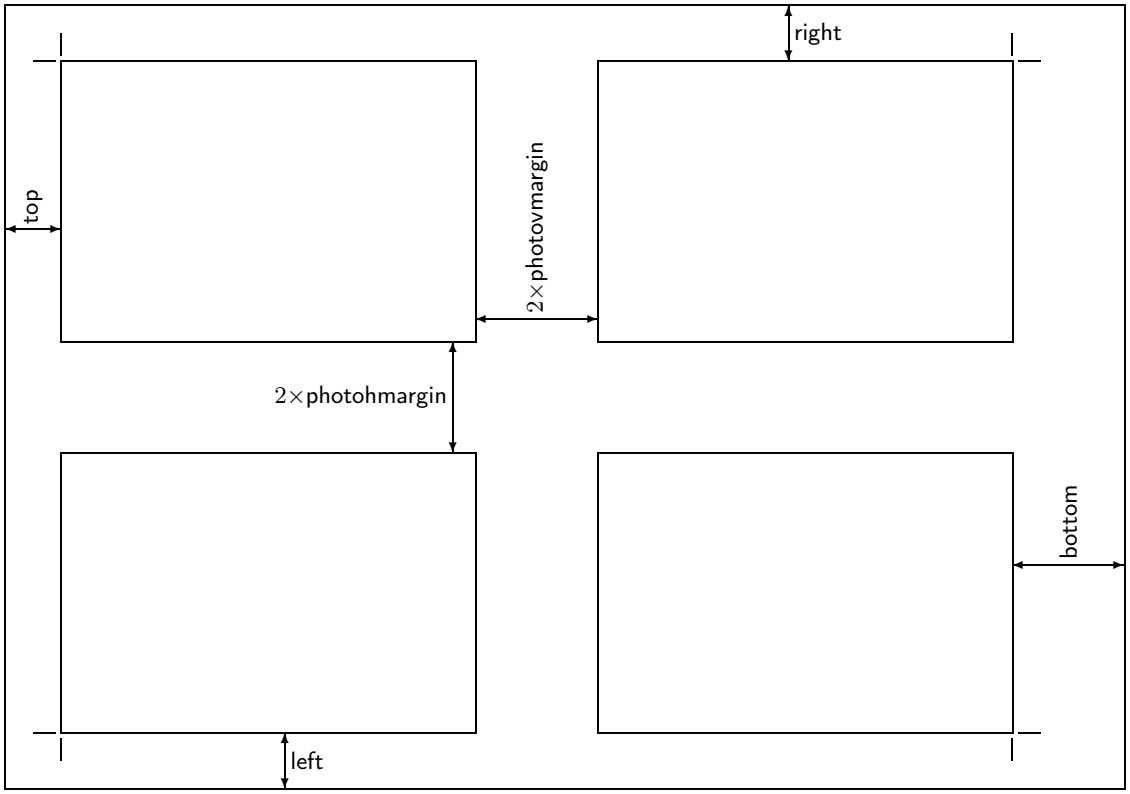


Fig. 7: Dividing the print area

Print slot is too narrow

Print slot is too shallow

Fig. 8: Fitting a picture into a print slot

⟨*filename*⟩*⟨*num*⟩

or as:

⟨*filename*⟩

where ⟨*filename*⟩ is the name of a picture file, and ⟨*num*⟩ is the requested number of copies for this photograph. If the "*⟨*num*⟩" part is omitted, a value of one is assumed.

photoprint stops reading the file as soon as it reads an empty string, so the file should not contain empty lines except at its end. There is no special character to mark a line as a comment.

Putting the photos in the print slots The print slots are used starting from the upper left, and ending with the lower right, from top to bottom and from left to right. Each picture is put in the first unused slot on the page.

For each picture, photoprint compares the orientation (landscape or portrait) of the print slot and the ori-

entation of the picture. If they differ, the picture is rotated by 90° counter-clockwise.

Then, photoprint computes the width the picture would have if it was scaled by a factor s_h so that its height were exactly the height of the print slot. This width is called the *scaled width* of the picture. photoprint also computes the height the picture would have if it was scaled by a factor s_w so that its width becomes exactly the width of the print slot. This height is called the *scaled height* of the picture.

If the scaled width of the picture is larger than the width of the print slot, photoprint scales the picture by s_h and trims its left and right edges by half the difference between the scaled width and the width of the slot.

If the scaled height of the picture is larger than the height of the print slot, photoprint scales the picture by s_w and trims its top and bottom edges by half the difference between the scaled height and the height of the slot.

In both cases, the picture fills the print slot, but it is cropped, as shown in figure 8.

However, when the `fullpictures` option is on, photoprint uses the smaller of s_h and s_w so that the picture fits entirely inside the print slot, between equal white bands to fill the slot.

Contact sheets Contact sheets are a special case in which photoprint reserves space at the top of the page to display the title of the contact sheet and the picture range (there are 20 pictures on each page of a contact sheet). It also reserves space between the rows of pictures to display their names, indices and date if available.

Related software

Photoprint started as a Java application, but I never managed to make it print my pictures the right way. This is why I started writing the photoprint document class (everybody knows that it is much easier to develop in LaTeX than in Java ...)

However, the Java Photoprint application is still here and allows choosing pictures to print and to set the number of copies of each. So I added a menu item to save the current selection of pictures to a file whose default name is ... `photoprint.job`.

This Java application has never been released, but I plan to make it available as a helper to photoprint. It may even launch `pdflatex` and open the result in Acrobat so that people who don't like command-line tools feel more at ease. Keep an eye on

`http://wwwsi.supelec.fr/fb/Development.html`

for a release.

Output Routine Requirements for Advanced Typesetting Tasks

David Kastrup
Kriemhildstr. 15
44793 Bochum
Germany
dak@gnu.org

Abstract

LaTeX's output routine is designed with fixed layout and functionality in mind. This is even true for the current state of the new output routine work by the LaTeX3 team. When typesetting things like critical editions, one may have to cater for multiple levels of marginal notes (like verse/line numbers and additional cross references), multiple and nested levels of footnotes, footnotes also carrying marginal notes, multicolumn material with synchronized page breaks and so forth and so on. It is not feasible to anticipate every possible perversion of typesetting tasks in advance.

For this reason, flexible methods of allocating the various special penalties for communicating the relevant information for such material, and for interpreting, collecting and assembling the relevant material within the context of the output routine and minipage-like environments are required. The implementor of such contorted layouts should then have the possibility to allocate the required levels and layers of specialities and plug them together in comparative ease.

A suitably generic core could be employed by vastly different and far more diverse typesetting tasks than just the LaTeX base classes. With suitable generality, it might even become feasible to root different systems with different user interfaces (like ConTeXt) on common core mechanisms, making it easier to cross-port functionality over the various systems.

Résumé

La routine de sortie de LaTeX a été conçue en vue d'une utilisation avec des mises en page et des fonctionnalités fixes. Ceci s'applique aussi à l'état actuel de la nouvelle routine de sortie développée par l'équipe LaTeX3. Quand on compose des documents tels que des éditions critiques, on nécessite parfois plusieurs niveaux de notes marginales, plusieurs niveaux de notes de bas de page, des notes de bas de page munies de notes marginales, du multicolonnage avec fins de page synchronisées, etc. Il est impossible de prévoir toutes les perversions typographiques possibles et imaginables, à l'avance.

Pour cette raison, on nécessite des méthodes flexibles, pour allouer les pénalités spéciales, pour communiquer l'information *ad hoc*, pour interpréter, collectionner et assembler le matériel dans le contexte de la routine de sortie et des environnements de mini-page. L'implémentateur d'une telle mise en page doit être capable d'allouer les niveaux et couches requis nécessaires et des les joindre au logiciel relativement facilement.

Un noyau raisonnablement générique peut être utilisé par un grand nombre de tâches de composition diverses et variées, bien au-delà des classes LaTeX de base. En gardant un niveau de généralité assez haut, on pourrait même tenter d'utiliser des interfaces différentes (comme ConTeXt) et obtenir ainsi une compatibilité inter-systèmes.

Introduction to output routines

The output routine is one of the more mysterious pieces of TeX. The chapter of the TeXbook discussing output routines claims that designing output routines makes one achieve the level of a "TeX Grandmaster".

As is so often the case, mastery of the concept of an output routine in plain TeX will only barely prepare you for the complexities awaiting you with LaTeX's variant of an output routine.

Basic operation So what is an output routine? When TeX is typesetting pages of continuous text, it will gather material until it can find a least-cost page break intended to make the gathered material fit the `\pagegoal` size. The gathered material will then be placed into `\box255` and the output routine stored in the token register `\output` will be processed in a group of its own. Usually it will arrange the gathered material in some way, add headers, footlines and page numbers, and ship the gathered results out in typeset form with the `\shipout` command. At the time of the `\shipout` command all `\open` and `\write` commands stored in the box shipped out are ex-

panded and written out. This is what makes it possible to have page labels corresponding to the actual page numbers at the time of shipout: the corresponding info is written to the `.aux` file at that time.

The output routine may decide to place material back on the main vertical list instead of shipping it out.

Marks One feature connected to the output routine is marks: marks can contain arbitrary tokens and are placed in the vertical list with `\mark` commands. The first and last mark on any page can be accessed as `\firstmark` and `\botmark`, respectively, when the output routine gets called. ε-TEX provides for more than one mark at a time if necessary. Marks can be used to gather information about what range of material has made it to the page.

Special penalties and 'here' points Penalties govern how infeasible to consider a page break. Negative penalties indicate good break points. Penalties of −10000 and below force an immediate page break. Those penalties don't occur naturally, so one can use various values for signalling conditions to the output routine in so-called 'here' points. One possibility is to use TEX's `\vadjust` command to insert such a special penalty right after the current line. The page will then be broken at that point and the output routine called. The output routine will then notice that it was invoked by a special penalty and will cause a corresponding special action, like placing a figure or margin note, or adjusting to a grid, or placing a line number (for that purpose, the normal interline penalty would be set to an appropriate special penalty) or a number of other possible actions. Typically, `\box255` will be just contributed back to the current vertical list in the output routine after the necessary special action has been called.

Insertions TEX has the concept of insertions that are used for floating material and things like footnotes. An insertion is a mechanism by which material associated with the main vertical list, such as footnotes or floating figures, is collected.

Insertions consist of the following items:

A box register is where the collected insertions appear when the output routine is called. Insertions may be split or float completely to the next page. In that case the remaining material is not in the box but will be inserted for the main vertical list immediately *before* any material contributed from the output routine.

A skip register specifies how much vertical space on the main page is consumed if at least one item from the insertion appears.

A count register is a scale factor telling TEX how much space in the main vertical list gets eaten up by material in the insertion. For example, single column footnotes below double column text might take twice the vertical space from the galley, margin notes might take up no space at all, and so on.

A dimension register can be set to the maximum amount of material that can be permitted from insertions of that type on a single page.

There are also parameters recorded with every single insertion, like its floating penalty (how bad is it if the insertion material does not at least start on the same page where it is referenced).

Insertions like footnotes may be split across pages, in which case the material before the split appears in the corresponding box register while the material following the split is placed into an insertion before the material that the output routine may recontribute to the main vertical list.

Examples for output routine requirements

We start off with two rather innocuous extracts from a two-language version of the Iliad (figure 1). Such two-language versions of a text will usually be set in two separate passes where each pass concerns itself with typesetting one language and after completing both passes, the results get joined into one common document by alternating the produced pages. This particular variant looks like one of the easiest ones: since we have an original text organized by verse lines, and a translation organized similarly, synchronization happens automatically.

Almost: what if we have overlong lines and need to wrap them? There are two approaches: the first one is to just put the line end somewhere else, usually in the line below. The version in figure 1(a) has to use the line above because of a verse number, so the placement also depends on the contents of the following lines. However, it can also depend on the position of page breaks: if the following line happens to be on the next page, we get more space for tacking on some line stubs and may not need to move stuff around too much. There are advantages to trying to fix stuff such things, as well as line numbers, within the scope of the output routine, but in this case it would probably be overkill.

The second approach is to synchronize the extra line needed with the translation, as seen in figure 1(b). Of course, this synchronization needs to be established in a multipass scheme: the blank lines can only be inserted after the other language has had its page set. It would actually be more convenient to have two copies of TEX running in lockstep, each reading from its own input file. Then we could save ourselves the trouble of a multipass algorithm.

Unique arrangements of standard elements The next interesting example is an excerpt (figure 2) from James Joyce's "Finnegans Wake". We notice a few difficulties with regard to LATEX's usual operations: the footnote area has

κεῖνος δ' αὖ περὶ κῆρι μακάρτατος ἔξοχον ἄλλων
ὅς κέ σ' ἐέδνοισι βρίσας οἶκόνδ' ἀγάγηται.
οὐ γάρ πω τοιοῦτον ἴδον βροτὸν ὀφθαλμοῖσιν,
οὔτ' ἄνδρ' οὔτε γυναῖκα· σέβας μ' ἔχει εἰσορόωντα.

Aber keiner ermißt die Wonne des seligen Jünglings, [Hause!
Der dich gewinnt mit den reichsten Geschenken und führt dich nach
Denn ich sah noch nie solch einen sterblichen Menschen,
Weder Mann noch Weib! Mit Staunen erfüllt mich der Anblick!

(a) Synchronicity by arrangement

ἔχθεσθ', ἀλλ' ἔτι πού τις ἐπέσσεται ὅς κεν ἔχῃσι
δώματά θ' ὑψερεφέα καὶ ἀπόπροθι πίονας ἀγρούς.«
ὣς φάτο, τῆς δ' εὖνησε γόον, σχέθε δ' ὄσσε ῥόοιο.

ἣ δ' ὑδρηναμένη, καθαρὰ χροὶ εἵματ' ἑλοῦσα,
εἰς ὑπερῷ' ἀνέβαινε σὺν ἀμφιπόλοισι γυναιξίν,

Ganz verhaßt; es bleibt ihm noch einer, daß er beherrsche
Dieses hohe Haus und die weiten gesegneten Felder.
Also sprach sie und stillt' ihr den Gram und hemmte die Tränen.

Und sie badete sich und legt' ein reines Gewand an,
Ging hinauf in den Söller, von ihren Mägden begleitet,

(b) Synchronicity by spacing

FIG. 1: Two excerpts from the Iliad

As we there are where are we are we there from tomtittot to teetootomtotalitarian. Tea tea too oo. UNDE ET UBI.

With his broad and hairy face, to Ireland a disgrace.

Whom will comes over. Who to caps ever. And howelse do we hook our hike to find that pint of porter place? Am shot, says the bigguard.[1] SIC.

Menly about peebles.

Dont retch meat fat salt lard sinks down (and out).

Whence. Quick lunch by our left, wheel, to where. Long Livius Lane, mid Mezzofanti Mall, diagonising Lavatery Square, up Tycho Brache Crescent,[2] shouldering Berkeley Alley, querfixing Gainsborough Carfax, under Guido d'Arezzo's Gadeway, by New Livius Lane till where we whiled while we whithered. Old Vico Roundpoint. But fahr, be fear! And natural, simple, slavish, filial. The marriage of Montan wetting his moll we know, like any enthewsyass cuckling a hoyden[3] in her rougey IMAGINABLE ITINERARY THROUGH THE PARTICULAR UNIVERSAL.

[1] Rawmeash, quoshe with her girlic teangue. If old Herod with the Cormwell's eczema was to go for me like he does Snuffler whatever about his blue canaries I'd do nine months for his beaver beard.
[2] Mater Mary Mercerycordial of the Dripping Nipples, milk's a queer arrangement.
[3] Real life behind the floodlights as shown by the best exponents of a royal divorce.

260

FIG. 2: An excerpt from "Finnegans Wake"

a different text width from the main text, and we have margin notes on both borders of the text. LaTeX's margin notes are a pain to switch between the margins (using `\reversemarginpar`), and it happens that we only have the possibility to specify outer and inner margin notes, whereas this chapter of "Finnegans Wake" would rather require left and right notes since the notes do not switch sides on opposing pages. This kind of layout should be easier to achieve than it is currently with LaTeX 2ε. In addition, we find that in "Finnegans Wake" several chapters have separate layouts, so it would be desirable to be able to switch between layouts from chapter to chapter, changing page formats and general layouts on the fly in the middle of a document.

Critical edition In figure 3, we have an example for a critical edition. We happen to have three footnote apparatus here. The first one is from the original commentator. The second one gives text variants of the original commentary. The third apparatus gives comments of the current editor. The interesting thing is that footnote anchors of the lower footnote apparatus can occur anywhere in the material on the page before. Note that the order of footnote numbers depends on the page breaks: if a footnote is continued to the next page, and a lower level footnote is anchored within the part broken to the next page, the number and position of the lower level footnote

und müssen sie einerseits gegen die Kulturkrankheiten der Skepsis und der Anarchie sowie des naturalistischen Atheismus verteidigen, andererseits sie in die neuen intellektuellen Horizonte einstellen, den jeweils neuen ethisch-sozialen Aufgaben anpassen[c1)]. |

Zu solchen Betrachtungen leitet uns der Befund der Historie. Sie sind B 78
freilich Geschichtsphilosophie und insofern keine strenge Wissenschaft. Aber die Wissenschaft ist nicht bloß exakte Wissenschaft, sonst müßte sie auf Mathematik und Naturwissenschaft[a<], etwa auch eine streng motiva-

1) Hiermit sind nur meine früheren Ausführungen über „Die Selbständigkeit der Reli- A 70, B 77
gion" und „Metaphysik u. Geschichte" (Z. f. Th. u. K. VIII 1897) fortgeführt und teilweise genauer bestimmt, insofern mit dem Ideal eines Begriffes der Religion noch stärker gebrochen ist, als es dort bereits der Fall war. Im übrigen darf ich auf das neue Buch Euckens verweisen „Der Wahrheitsgehalt der Religion" Leipzig 1901, das meinen Anschauungen überaus nahe steht. Nur[b] erledigt auch Eucken m. E. zu[c] rasch das Absolut-

---

a *In A, folgt:* auf Anthropologie u Psychologie

b *A:* Nur über die Begriffe der „universalen" und der „charakteristischen Religion", namentlich über das Verhältnis beider würde ich eine etwas andere Darstellung wünschen. Die „universale" Religion ähnelt zu stark dem alten Wesen und Begriff der Religion und die „charakteristische" zu stark der Realisation des Begriffes der Religion.[148] Eben damit

c–c *A:* ziemlich

---

148 Religion muß nach Eucken einen „universalen Charakter" tragen. Rudolf Eucken: Der Wahrheitsgehalt der Religion (1901), S. 209. Sie sei „aus dem Ganzen des Lebens" erwachsen, „es handelt sich bei ihr um den Gewinn einer neuen Welt, der Welt selbständigen, sowohl der Natur als dem menschlichen Getriebe überlegenenen Geisteslebens" (S. 209). Religion sei damit „der Höhepunkt, an dem sich eine Umwandlung des ganzen Lebens vollzieht"; sie sei die „Konzentration des Ganzen" (S. 210). Innerhalb der universalen Religion habe sich zwar eine „weltüberlegene Innerlichkeit" (S. 331) angekündigt, da aber das „Göttliche" sich „einstweilen nur als Gesetz und Gericht" (S. 303) erschlossen habe, sei gleichzeitig „die eigne Erhabenheit und unsere Nichtigkeit zu stärkster Empfindung" (S. 303) gekommen: „So droht die ganze unermeßliche Lebensflut ins Leere zu verrinnen" (S. 303). Eine Rettung komme aus der „Kraft göttlichen Lebens", aus einer „weitere[n] Erschließung weltüberlegener Vernunft", wie diese „Wendung" in den „sog. historischen oder positiven Religionen" uns entgegengetreten sei (S. 303). Die universale Religion entwickelt sich zur charakteristischen. Eucken setzt bei dieser eine „Thathandlung der Gottheit selbst" an, „in der [...] eine neue Art der Gemeinschaft, ein neuer Lebenskreis, eine neue Wirklichkeit gebildet wird" (S. 332). „Daß eine solche neue Welt in unser Dasein hineinwirkt, daß ihm ein überweltliches Leben gegenwärtig ist, das bildet die Voraussetzung einer neuen Gestalt der Religion, die uns die

FIG. 3: Critical Edition with multiple footnotes

will be *after* a same level footnote anchored into the main text.

The footnote number reordering can be solved by using a `\label`-like mechanism that will get the numbers right in a multiple pass method, so we won't delve deeper here.

What is more interesting here is that we have margin notes that are used for indicating the original pagination. Margin notes like that can occur in the main text, but they can also occur in the footnotes. Here the standard mechanisms of LaTeX break down: as insertions, LaTeX treats margin notes only in the main text. So what do we have to do in order to get the same treatment in the footnotes also?

There are several possibilities. One is to use `\vsplit` in order to achieve similar functionality to the output routine, in figuring out 'here' points placed into the footnote. So it is apparent that we would really want to have the sort of 'here' point processing from the output routine available also as a separate operation.

Another possibility would be to place insertion material into the footnote boxes, then rerun the output routine with the material that has been inserted from within the footnotes. This could make for some problems in a case where we have to deal with multiple columns, though: if footnotes are to be attached to each column, the amount of material for a column might change when insertions migrate out, leading to different column breaks.

So no generally useful strategy appears to suggest itself, making it obvious that any particular format should be as free as possible to choose its own order of trying to deal with that problem.

Of course, this does not address the main problem: we have here several layers of page breaking decisions, and complicated rules about just when something might be carried over or placed on the next page. There are some easily understandable rules: a footnote must begin on the same page as its reference point, all footnotes in a block must be numbered consecutively in order of vertical appearance of their reference points on the page, regardless of the logical order, and only the last footnote on one footnote block may be broken over to the next page. If a footnote is broken over to the next page, the part carried over must constitute the top part of the footnote block on the next page, even if footnotes in a level ordinarily placed before it (such as footnotes from the main text) appear in the same block.

If we have a footnote to a footnote that gets split across a page, then its top part must come at the bottom of its footnote block, whereas its bottom part must continue on the top of the footnote block of the next page. This means that subordinate footnotes must be

- kept in separate insertions from other footnotes in the block
- kept in separate insertions even from other footnotes of the same level, since their reference might fall on a different page, making other footnotes from, say, the main matter intercede.

Why do we need insertions at all, rather than handling this just with a queue of boxes, like LaTeX's usual floats do? Because then LaTeX will not attempt to split footnotes...

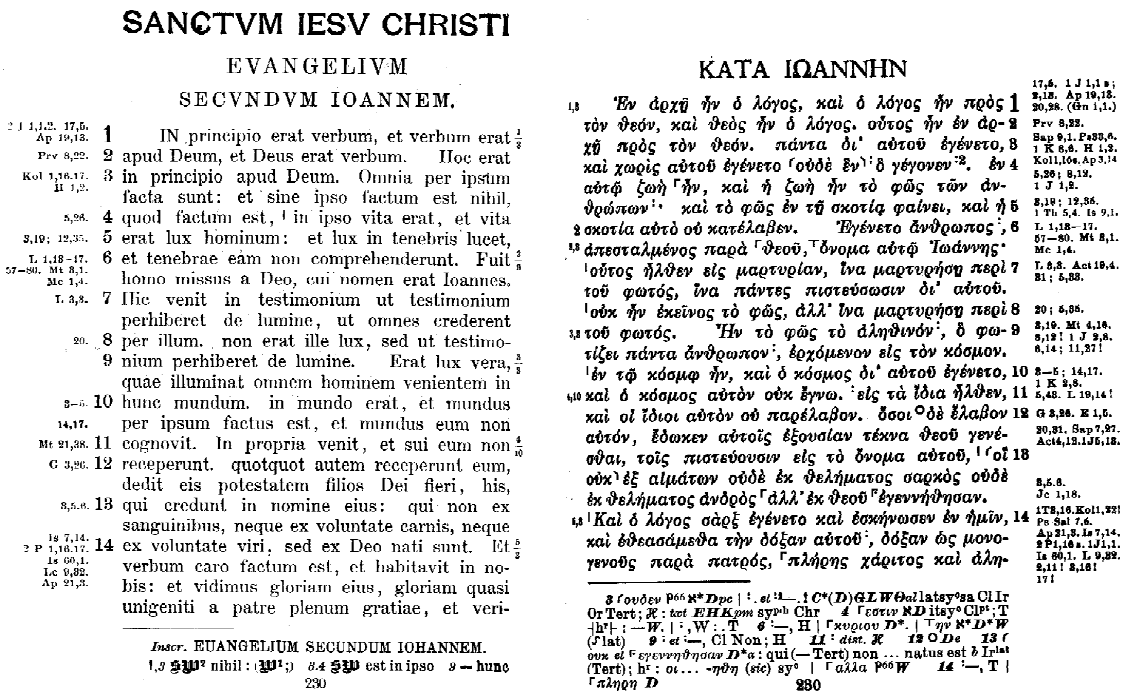
The details are pretty messy. Let it suffice to say that we have a complicated mess of decisions to make. It turns out that we can do this reasonably only with a multipass approach again where footnotes get shortened systematically until the material fits the page.

It becomes apparent that we want a mechanism with which we can easily specify that we want a particular box treated for 'here' points both for subordinate footnotes as well as margin notes, while we would not want to have to code things like attachment of margin notes more than once.

Another critical edition In figure 4, we have a typical passage from a biblical critical edition. We see footnotes in run-in paragraph form, we have verse numbers (one sort of margin notes), we have *another* level of margin notes on the outside for cross references, we have margin notes for indexing purposes on the inner side. All in all, three levels of margin notes, and we have different languages on opposite pages that have synchronized page breaks. Now the synchronization of page breaks is not possible to achieve sensibly apart from typesetting even and odd pages separately and writing information into external files for synchronization: TeX does not offer a way for interchanging between alternating document sources. One could possibly try to make the text of one language insertions in a main text created by the other language, but this would be pretty awkward and would certainly require that the texts get interspersed. Since the texts will usually be prepared by people proficient in different languages, this would be distinctly impractical.

Synchronization, however, could be left to a separate package. It is easy enough to achieve with insertions: synchronization points could be implemented simply by creating an insertion with a fixed height. The output routine can then measure the height of the respective insertion box, corresponding to the number of synchronization points, and write it out to an external file. The number of synchronization points on a page can be strictly limited by setting the `\dimen` register of the appropriate insertion to a maximum height value.

A magazine Figure 5 provides us with a view to more complicated figure placements. Placement requirements like this are commonplace. Let us suffice here by saying that current experimental output routine work from LaTeX3 can accommodate most of the difficulties of placing material like that, but the work is not yet complete, and several restrictions exist that make working with this cumbersome, and the interfaces are still in a state of flux.



SANCTVM IESV CHRISTI

EVANGELIVM

SECVNDVM IOANNEM.

IN principio erat verbum, et verbum erat apud Deum, et Deus erat verbum. Hoc erat in principio apud Deum. Omnia per ipsum facta sunt: et sine ipso factum est nihil, quod factum est, in ipso vita erat, et vita erat lux hominum: et lux in tenebris lucet, et tenebrae eam non comprehenderunt. Fuit homo missus a Deo, cui nomen erat Ioannes. Hic venit in testimonium ut testimonium perhiberet de lumine, ut omnes crederent per illum. non erat ille lux, sed ut testimonium perhiberet de lumine. Erat lux vera, quae illuminat omnem hominem venientem in hunc mundum. in mundo erat, et mundus per ipsum factus est, et mundus eum non cognovit. In propria venit, et sui eum non receperunt. quotquot autem receperunt eum, dedit eis potestatem filios Dei fieri, his, qui credunt in nomine eius: qui non ex sanguinibus, neque ex voluntate carnis, neque ex voluntate viri, sed ex Deo nati sunt. Et verbum caro factum est, et habitavit in nobis: et vidimus gloriam eius, gloriam quasi unigeniti a patre plenum gratiae, et veri-

Fig. 4: John 1

Conclusions

It is immediately apparent from a cursory scan of a typical bookcase that a variety of requirements for the arrangement of graphical elements on a page exist that are not catered for by LaTeX's standard classes and typesetting mechanisms. Separate tasks provide for separate challenges, and rather than trying to cater for all of them with an overly generic format, it will be necessary to provide convenient access to standard graphical and algorithmic elements, by the use of templates and generic definitions.

Currently, document design necessitates output routine programming and a vast amount of wizardry. But most tasks could be accomplished by rather working from standard building blocks like insertion lists, here points, default mechanisms for margin notes and so on. This would facilitate a considerably larger variety of serious document design tasks with tolerable effort and could help to spread the use of TeX in publishing houses.

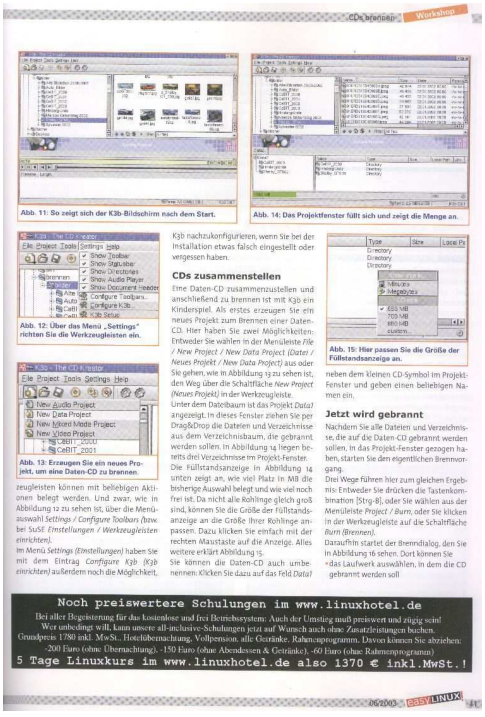


Fig. 5: A magazine

Bibulus — a Perl/XML replacement for BibTeX

Thomas Widmann
3/2, 54 Mavisbank Gardens
Glasgow G51 1HL
Scotland, EU
thomas@widmann.uklinux.net
http://www.widmann.uklinux.net

Abstract

In this paper Bibulus will be introduced. Bibulus is a very flexible, multilingual system for formatting bibliographies in several formats, including LaTeX. Bibliographies are stored in XML, but Bibulus includes a program to convert old BibTeX databases.

Bibulus is being developed as open-source software at http://savannah.nongnu.org/projects/bibulus/.

This paper will first summarise the current BibTeX situation. Then, the main features of Bibulus will be introduced, followed by a guide to switching to Bibulus from BibTeX.

Résumé

Dans cet article nous présentons Bibulus, un système multilingue et très flexible pour composer des bibliographies dans plusieurs formats, y compris LaTeX. Les bibliographies sont stockées en XML, mais Bibulus propose également un programme pour convertir des vieilles banques de données BibTeX.

Bibulus est développé comme logiciel *Open Source* dans http://savannah.nongnu.org/projects/bibulus/.

Dans la première partie de l'article nous présentons l'existant. Puis, les caractéristiques principales de Bibulus seront présentées, suivies d'un guide du passage de BibTeX à Bibulus.

Background

BibTeX is a great tool — something which is demonstrated by the fact that it is still being used today, almost twenty years after it was created — but it has some problems. Some of these have been solved by additional packages:

Natbib and other LaTeX packages add real support for author–date citations.

Custom-bib makes it possible for ordinary human beings to construct `.bst` files according to their needs.

Jurabib and other packages add support for citations in footnotes and for use of *ibid.* citations.

Chapterbib and bibunits allow the use of multiple bibliographies in a single document.

amsrefs among other things allows the definition of bibliographies within the LaTeX file and adds many more entry types.

However, to the best of my knowledge, there are still problems which are not satisfactorily handled by any existing programs:

- Sorting can be very difficult for non-English bibliographies.
- Sharing a single bibliography file between documents in several languages is very difficult, because notes, dates and types are stored as ordinary text in the bibliography file.
- Creating bibliographies in multiple writing systems is not handled at all (except for solutions that allow just one additional alphabet, such as `hellas.bst` for Greek).
- Interacting with programs other than LaTeX is problematic, because many features are hard-coded in BibTeX.
- Although it is not too difficult to define a `.bst` file thanks to Custom-bib, it would be nice to be able to define the appearance of the bibliography within the LaTeX file.[1]
- Like TeX, BibTeX is compiled with a certain memory limit, which means that an expanded version is sometimes needed, something which is likely to cause a newcomer immense problems.

1. To be fair, this is handled by `amsrefs`, but this is not a part of BibTeX.

Enter Bibulus

Bibulus is an attempt to address the problems listed above. In the following some of its main features will be introduced.

XML Bibulus requires its databases to be in Bibulus XML (specified in a DTD[2] bundled with Bibulus). This is not as problematic as it sounds for two reasons.

First, Bibulus comes with a program which will convert old BibTeX databases to XML.

Second, data in XML is generally very easy to convert to another kind of XML, for instance by means of an XSLT script. Although no such scripts are included in the Bibulus module at the moment, it could easily be done if there was a need for them, for instance in order to use XML bibliographies conforming to other bibliographic DTDs with Bibulus.

Unicode Bibulus is truly multilingual. It uses Unicode internally, but it can both read and write other character sets.

Perl Bibulus is written in pure Perl. This means it is very portable, as Perl is available on most platforms.

Not just LaTeX To Bibulus, LaTeX is just yet another input/output format. If you want, you can also get your bibliography in pure ASCII or in HTML, and other formats are easily added.[3]

Specifying the style in your document Bibliography styles can be defined in the LaTeX file. While definitions via `\bibliographystyle` are still supported, one can also use this new format:

```
\bibulus{citationstyle=numerical,
         surname=comes-first,
         givennames=initials,
         blockpunctuation=.}
```

Of course, many more options are possible.

Entries can also be specified or modified within the LaTeX document. For instance, the following command will add a note with the text "Great!" to the entry called `sample`.

```
\bibulusadd{sample}{note}{Great!}
```

Open Source Bibulus is released under the GNU Public Licence which among other things means you get the source code and are free to make any changes you want. Bibulus is very much work in progress, both in the sense that many features have not been implemented yet and that there is a good chance your requests will be implemented.

Name "Bibulus" means *fond of drink*, *thirsty* in Latin. Furthermore, M. Calpurnius Bibulus was consul in Rome together with C. Iulius Cæsar in the year 59 BC.

The name was given in the hope that the program will be fond of "drinking" many books, and that it will rule together with the best typesetting systems.[4]

Tour of Bibulus

In the following we shall have a closer look at various aspects of Bibulus.

Converting BibTeX databases Bibulus comes with a conversion program called `bib2xml` which will convert a BibTeX database to Bibulus XML. As an example, let us convert the file `xampl.bib` which comes with BibTeX.

```
> bib2xml xampl
This is BibTeX, Version 0.99c (Web2C 7.3.1)
The top-level auxiliary file: tmp6600.aux
The style file: bib2xml.bst
Database file #1: xampl.bib
Odd edition number: Silver
```

It should be clear from the above that `bib2xml` calls BibTeX to do its job, thus ensuring it can parse all documents that BibTeX can. It produced one warning, since Bibulus assumes that editions can only be numbers, but `xampl.bib` contains a "silver edition".

The result of this is a file, `xampl.xml`, which conforms to the Bibulus DTD.

For instance, in the original file there is an entry which looks like this:

```
@PHDTHESIS{phdthesis-full,
   author = "F. Phidias Phony-Baloney",
   title = "Fighting Fire with Fire:
            Festooning {F}rench Phrases",
   school = "Fanstord University",
   type = "{PhD} Dissertation",
   address = "Department of French",
   month = jun # "-" # aug,
   year = 1988,
   note = "This is a full PHDTHESIS entry",
}
```

In Bibulus XML, this has become:

```
<thesis id="phdthesis-full" type="phd">
  <author>
    <name gender="unknown"
          nametype="familylast">
      <given>F. Phidias</given>
      <family>Phony-Baloney</family>
    </name>
  </author>
  <title>Fighting fire with fire:
         Festooning French
         phrases</title>
  <institution>Fanstord
               University</institution>
```

2. The Bibulus DTD has been inspired by the one described in *The LaTeX Web Companion* [1] and by various bibliographic DTDs that can be found on the Internet.

3. For instance, as suggested by a reviewer, it would be easy to make Bibulus output BibTeX databases.

4. Hopefully more happily than its ancient namesake.

```
  <place>Department of French</place>
  <year month="8">1988</year>
  <note>This is a full
        PHDTHESIS entry</note>
</thesis>
```

Some notes:

- The BibTeX entry types `@MASTERSTHESIS` and `@PHDTHESIS` have been unified.
- The `type` field is gone; instead, there is a `type` attribute which can contain certain predefined types. (Allowing free-form text makes it impossible to output the entry in another language.)
- The `month` field has become an attribute to `year`; furthermore, it now only allows a single month, not a range of months.
- The title has been down-cased. The BibTeX practice of down-casing on the fly is problematic, not least because it requires an escape mechanism for words that should remain in upper case. Instead, Bibulus follows `amsrefs` in storing the lower-case version in the file and up-casing on the fly instead (this applies only to English, of course).
- Two attributes to `name` have been added. The first, `nametype`, allows the handling of names in Chinese and other languages where the family names comes before the given name, even when used in Western languages (e. g., "Deng Xiaopeng" would normally be written in this way, not as "Deng, Xiaopeng" or "X. Deng").[5] The second attribute, `gender`, is necessary to certain languages where other words in the sentence inflect according to the gender of the author of a reference.

Let us regard a further example from the same file.

```
@ARTICLE{article-full,
   author = {L[eslie] A. Aamport},
   title = {The Gnats and Gnus Document
            Preparation System},
   journal = {\mbox{G-Animal's} Journal},
   year = 1986,
   volume = 41,
   number = 7,
   pages = "73+",
   month = jul,
   note = "This is a full ARTICLE entry",
}
```

This is transformed by `bib2xml` to the following:

```
<article id="article-full">
  <crossref id="article-full-PART2"/>
  <author>
    <name gender="unknown"
          nametype="familylast">
      <given>L[eslie] A.</given>
      <family>Aamport</family>
    </name>
  </author>
  <title>The gnats and gnus document
         preparation system</title>
  <pages>73+</pages>
  <note>This is a full ARTICLE
        entry</note>
</article>
<magazine id="article-full-PART2">
  <journal>G-Animal's Journal</journal>
  <volume>41</volume>
  <number>7</number>
  <year month="7">1986</year>
</magazine>
```

Most of this is hardly surprising by now, except for the fact that the entry has been split into two. This does not affect the output since Bibulus (like BibTeX) will inline cross-references that are only used a limited number of times (specified by the user). This allows for a significant simplification of the DTD.

Editing There is no Bibulus editor (for the time being), but there exist many XML editors, all of which ought to work well with Bibulus XML. However, Bibulus XML is really not any more complicated than BibTeX databases, so it is also quite feasible to edit the files in a plain text editor.

The same situation holds for validation, i. e., checking that an XML file conforms to the definitions in the DTD: There is no Bibulus validator, but many standard tools can be used, and it is highly recommended to validate Bibulus bibliographic databases in this way instead of relying on built-in error handling.

Notes and annotations in the text BibTeX requires us to write notes and annotations in the bibliographic database, but there are problems with this approach. Annotations are typically unique to each bibliography (this is often true for notes, too). The bibliographic database is therefore the wrong place to specify them — it should be done in the main text instead. Furthermore, these fields require translation when the document is translated, something which is much easier if they are kept together with the main text. Bibulus allows both for backwards compatibility.

Transliterations and translations One of the most important *raisons d'être* for a bibliography formatting system is to make it possible to define an entry once and then extract it in many different formats. To achieve this, Bibulus is able to transliterate names and titles automatically, and it is possible to add translations of titles, either in the XML database or in the LaTeX source.

5. It is thus not to be used for, e. g., Hungarian names that behave as other Western names when used in English.

Migrating to Bibulus

In the following we shall see how a LaTeX user can move from BibTeX to Bibulus.

Getting started The very first step is to convert the BibTeX databases to Bibulus XML, as described on page 469.

Without making any changes to the LaTeX document, one can then start to use `bibulustex` instead of `bibtex`. If a standard bibliography style is used (e.g., `plain`), this should produce equivalent output.

However, only a few bibliography styles are defined, so this is likely to be less than needed.

Farewell to `\bibliographystyle` The second step is to use the `\bibulus` command in LaTeX to define the style of the bibliography.

The default is a style close to BibTeX `plain`, so only options that differ need to be defined. For instance, if one wants `alpha` labels (first letters of the last name + last two digits of the year) instead of numerical labels and furthermore wants author names to be written in small caps, one can just write the following in the LaTeX document:

```
\bibulus{citationstyle=alpha,
         namefont=sc}
```

More Bibulus commands in LaTeX The next step is to start to use the LaTeX commands for manipulating the bibliography, e. g., by adding notes, annotations or translations of titles.

It is also possible to create an alias for a title if one is not happy with its label in the XML file. For instance, if *The TeXbook* is stored with the ID `knuth86`, one might want to issue the following command:

```
\bibalias{knuth86}{texbook}
```

After this, citing `knuth86` and `texbook` will be fully equivalent.

Goodbye to `bibulustex` All that `bibulustex` really does is to get the filename from the command line and then do the following:

```
my $bib = new Bibulus::LaTeX;
$bib->procaux($filename);
open (BBL, ">$filename.bbl")
 or die "Could not write $filename.bbl.\n";
print BBL $bib->getbib;
close BBL;
```

If more functionality is needed, one can thus make a personalised version of `bibulustex` with extra functionality.

For instance, to output all years *ab urbe condita* (after the founding of Rome), add the following code after the first line:

```
$bib->whenparsing('year',
                  sub {
                    return $_[0] + 754;
                  });
```

Most of this is just Perl code. What matters is the following: The code within the `sub` is executed when a `<year>` is encountered; the contents of this XML chunk is passed to the `sub` in `$_[0]`; and, whatever the `sub` returns replaces the old contents of the chunk.

This is a very simple and silly example, but the possibilities are endless, especially as the full power of Perl is available.

Extending Bibulus If the built-in extension hooks do not provide enough freedom, one can extend Bibulus quite easily.

Create a file called, say, `myBibulus.pm` and put the following into it:

```
package myBibulus;
use Bibulus::LaTeX;
our @ISA = qw(Bibulus::LaTeX);

sub newblock {
  return "\\par ";
}
```

Now replace the line

```
 my $bib = new Bibulus::LaTeX;
```

with

```
 my $bib = new myBibulus;
```

in your personalised version of `bibulustex`, and the bibliographies produced will now have the blocks separated by `\par` instead of `\newblock`.

Any internal Bibulus function can be overridden in this way.

Final words

This has been a brief introduction to the main features of Bibulus. As the program is being developed continuously, there may be features available now that have not been described in this paper, and the syntax of certain commands might have changed slightly.

For more information, please visit the project's website, `http://savannah.nongnu.org/projects/bibulus/`. There are also two mailing lists, one for developers and one for users — please consider joining one of them.

Bibulus still has a some way to go, but with the help of the user community, we can do it!

References

[1] Michel Goossens and Sebastian Rahtz. *The LaTeX Web Companion*. Addison-Wesley, Reading, Massachusetts, 1999.

BibTeX++: Toward Higher-order BibTeXing

Fabien Dagnat
Computer Science Lab, ENST Bretagne
CS 83818, F-29238 PLOUZANÉ CEDEX, France
Fabien.Dagnat@enst-bretagne.fr
http://perso-info.enst-bretagne.fr/~fdagnat/index.php

Ronan Keryell
Computer Science Lab, ENST Bretagne
CS 83818, F-29238 PLOUZANÉ CEDEX, France
rk@enstb.org
http://www.lit.enstb.org/~keryell

Laura Barrero Sastre, Emmanuel Donin de Rosière, Nicolas Torneri
(Emmanuel.DoninDeRosiere|Nicolas.Torneri)@enst-bretagne.fr

Abstract

In the LaTeX world, BibTeX is a very widely used tool dealing with bibliographies. Unfortunately, this tool has not evolved for the last 10 years and even if a few other bibliographical tools exist, it seems interesting to develop a new tool with new features using modern programming standards.

The BibTeX++ project began in 1999 and is written in Java for the portability and object oriented aspects and offers new functionalities: for the expressiveness, the security model and native Unicode character set support, BibTeX++ styles are directly Java classes; for compatibility, it uses advanced compiler techniques to translate plain old BibTeX styles to new Java styles. Such a translated style can even set the development basis of a new native BibTeX++ style to ease the style programmer's life. The architecture is designed for extensibility and split into different components: core, parsers (to be compatible with other bibliographical styles, sources or encoding schemes), pretty printers (to generate bibliographies for other tools than LaTeX), plugins. The plugin concept is used to dynamically extend BibTeX++ functionalities. For example, since (meta-)plugins can load new plugins, new styles can be directly downloaded from the Internet.

Résumé

Dans le monde LaTeX, BibTeX est un outil répandu pour gérer les notices bibliographiques. Malheureusement, cet outil n'a pas évolué ces dix dernières années. Même si d'autres outils bibliographiques existent, il semble intéressant de développer un nouvel outil, offrant de nouvelles fonctionnalités et utilisant de nouveaux standards de programmation.

Le projet BibTeX ++ a démarré en 1999 et il est écrit en Java pour des raisons de portabilité et de fonctionnalités accrues dues aux aspects de programmation orientée objet. Il offre de nouvelles fonctionnalités : BibTeX ++ est une classe Java, pour son expressivité, le modèle de sécurité et le support natif d'Unicode ; il contient un compilateur pour traduire les anciens styles BibTeX en Java, et ce code Java peut être à la base d'un style bibliographique BibTeX ++ natif de manière à faciliter la vie du programmeur ; l'architecture a été conçue en vue de l'extensibilité : le noyau, les parseurs (pour être compatible avec d'autres styles bibliographiques, d'autres sources ou codages), les enjoliveurs de code (pour générer des bibliographies pour d'autres outils que LaTeX), les plug-ins ; le concept de plug-in est utilisé pour étendre BibTeX ++ de manière dynamique. Ainsi, puisque les (meta-)plug-ins peut à leur tour charger d'autres plug-ins, des nouveaux styles peuvent être téléchargés sur Internet.

Introduction

A bibliography or list of references is a listing of all sources from which you have taken information directly (by literal quotation) or indirectly (through paraphrase), or where you have used information or reproduced material. These references are used to:

- clearly identify each document. So it provides some identification elements to allow the reader to look for these documents in library catalogues or anywhere else. In most cases, these identification elements are normalized [17] (we often use the name of the book, the author, the editor, ...), but because of the digital revolution, there are more and more document types (web pages for example) and identification elements (e-mail, URL, ...). So the management of the references has become more and more difficult;
- enable the reader to consult the sources you have used with a minimum of effort. Thus we have to indicate precisely where (on which page, the electronic location) or under which circumstances (personal interview, e-mail) you obtained the information.

Unfortunately, creating a bibliography has always been a headache for typographers: if the article talks about "Larousse (2002)" as a book introducing metamiddleware [20] and in the bibliography "Larousse 2002" is described instead as a French cookbook, there has clearly been a problem somewhere. Another problem is that each journal has its own bibliography style, so bibliographical management software has to allow the user to create his own style and send it to other people.

Fortunately, BibTeX [23] is a popular tool used by the LaTeX [21] community to generate bibliographical notices in publications. If there are also many other tools available [9], this one suits very well the LaTeX philosophy and is well integrated with it: the user describes what she wants with a mark-up language without having to dive into the deep layout details: the LaTeX infrastructure will use some styles to typeset the presentation from the high level content description.

The citation matter is stored in a database (a file with a `.bib` extension) and BibTeX picks from the `.aux` file the needed information according to the citation marks placed in the LaTeX document (`.tex` file) by the author. A typeset version of the bibliography is made by BibTeX to the `.bbl` file by using a `.bst` bibliography style file. The work-flow is summed up in figure 1 [13].

There are a lot of bibliographical styles available for BibTeX targeted at many different scientific journals, book styles, etc. The user needs only to select the desired style and the bibliography notice is generated from her common bibliographical reference database. There are also a lot of such bibliographical reference databases available on the Internet that can be used directly, such as [22].

There are many tools targeted at easing the management of all these BibTeX files: database tools, editors, ... For example to write this article we used the grand Emacs multi-platform text editor that has various

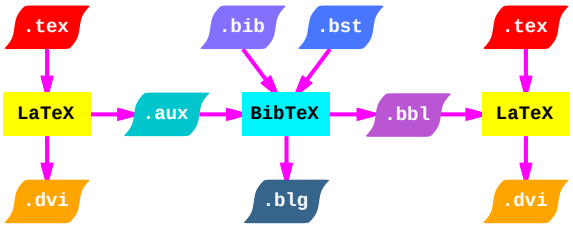


Fig. 1: BibTeX work flow.

modes to deal with a LaTeX document in various coding systems: the great AUCTeX mode [1] to deal with many things in the LaTeX source file, refTeX [7] to deal with citations and cross-referencing, such as a bibliography browse-and-pick mode, and some BibTeX editing modes.

A lot of BibTeX record databases are also widely available on the Internet through bibliographical servers such as CiteSeer [22] and it is often easy to get some BibTeX records from few keywords.

Even though there is so much material available for BibTeX, BibTeX is old, does not evolve anymore and has many shortcomings according to modern tool standards: the character encoding is constrained to an 8-bit ASCII variant, it is difficult to mix different languages on the database side or on the document side, memory usage is selected at compile time, style programming is done in a language that is rather optimized for the implementation efficiency and not for the style programmer's brain. The tool lacks extensibility and such modern features as network awareness.

This is why we began the BibTeX++ project in 1999 as a way to extend the BibTeX functionalities without throwing away the BibTeX compatibility, databases and styles.

In this article we present first the basic functionalities in BibTeX++ (§ 'BibTeX++ functionalities'), then the software architecture in § 'Software architecture of BibTeX++' with some emphasis on our BISTRO compiler (§ 'Software architecture of the BISTRO BST to JAVA compiler') and the plugin architecture (§ 'Plugins and metaplugins').

BibTeX++ functionalities

The BibTeX++ name has been chosen to assert a BibTeX compatibility with an improvement comparable to the object-oriented add-on from the C to C++ languages. The ordinary BibTeX++ user does not have to embrace the object oriented architecture of BibTeX++, but this is not the case for the style programmer (and of course the BibTeX++ programmers).

BibTeX compatibility Since BibTeX is a very widely used tool, compatibility with the old BibTeX is a requirement.

BibTeX usage is simple; the first thing to do is to create the database file (a file with a `.bib` extension) or better to use an old one with at least the bibliographical references that are used in the article being produced with LaTeX. Each reference record has a type chosen from the 13 possible default types (article, book, conference, thesis, ...) and is composed from various fields such as its key (used to cite this reference from the LaTeX file), its title, its author(s), the publication year and so on such as:

```
@BOOK{LaTeX:companion,
      Author = {Michel Goossens
                and Frank Mittelbach
                and Alexander Samarin},
      Publisher = {Addison-Wesley},
      Title = {The \LaTeX{} Companion},
      Year = 1994
}
```

This reference describes a book and can be cited with the key `LaTeX:companion`. If there are several authors, they are separated by an `and`. Plain LaTeX can often be put in most of the fields such as the `\LaTeX{}` in the previous `Title`. There are numerous kinds of fields (23) available to be used to define a reference. According to the reference type, a field can be mandatory or optional. Other fields can be used for extensions or information since they will be ignored by BibTeX itself.

Once the reference file is created, the reference must be inserted in the LaTeX document. For this purpose a `\cite{`⟨*key*⟩`}` command is put where a reference to ⟨*key*⟩ is to be cited. Thus with the previous example inserting in the text "`\cite{LaTeX:companion}`" will appear as "[13]".

One instructs LaTeX to use a bibliography ⟨*style*⟩ with `\bibliographystyle{`⟨*style*⟩`}` and to insert the bibliography somewhere in the document from the database ⟨*bib-base*⟩`.bib` with a `\bibliography{`⟨*bib-base*⟩`}`

When run, BibTeX picks in the ⟨*document*⟩`.aux` file the needed information given by the citation marks placed in the LaTeX document (⟨*document*⟩`.tex` file) by the author. A ready-to-typeset version of the bibliography is made by BibTeX to the ⟨*document*⟩`.bbl` file by using a ⟨*style*⟩`.bst` bibliography style file and the citation matter found in the `.bib` file(s). Error and information output goes to the ⟨*document*⟩`.blg` file. The work-flow is summed up in figure 1 [13].

BibTeX comes with some predefined styles but it is also possible to program other styles, for example to accept a `url` field to cite Internet information. All these styles are defined in `.bst` files that are used by BibTeX

```
FUNCTION {sort.format.names}
{ 's :=
  #1 'nameptr :=
  ""
  s num.names$ 'numnames :=
  numnames 'namesleft :=
    { namesleft #0 > }
    { nameptr #1 >
        { "   " * }
        'skip$
      if$
      s
      nameptr
      "{vv{ } }{ll{ }}{  ff{ }}{  jj{ }}"
      format.name$ 't :=
      nameptr numnames = t "others" = and
        { "et al" * }
        { t sortify * }
      if$
      nameptr #1 + 'nameptr :=
      namesleft #1 - 'namesleft :=
    }
  while$
}
```

FIG. 2: BST sample code.

to generate the bibliography according to the work flow described in figure 1.

From the user point of view, these concepts are also used in BibTeX++ *texto* with all the old BibTeX styles available too.

The style files are written in the BST language.[1] This is indeed an awful stack-based language rooted in the sixties that was chosen for an easy implementation of BibTeX concepts: it is simple to parse and to execute on a computer. The dark side is that the programming burden is put afterwards on the style programmer. BST code looks like that in figure 2 and there are 1258 such lines in the classical `alpha.bst`...

It is clear that building a new style from scratch is a *tour de force* and often basic programmers will change only few details. Fortunately there are many styles available already that can fit most needs, and there are also some custom style generators [5].

Extensions to BibTeX A basic extension is to be able to deal with encodings other than plain ASCII and more generally to be multilingual. Since BibTeX can sort the bibliography, a localized sort according to the document language is to be introduced.

1. According to its author, this language has no name in fact. But for the clearness of this article we call it "BST".

A tool designed today could hardly escape the networked world, so BibTeX++ must have a way to access world-wide on-line bibliographical databases through the Internet.

The bst language in BibTeX lacks some expressiveness and a new language should be designed instead of trying to extend the old one. Some modern features should be added to allow scalability and extensions, for example by using an object oriented language. But we also want compatibility with the old BibTeX styles that are written in bst, so this is rather tricky.

Besides, BibTeX targets only LaTeX; BibTeX++ could also target other typesetting tools or other bibliographical database concepts.

But since BibTeX++ is still a work in progress as a general bibliographical workbench, there are many new functionalities that are not yet implemented or even not envisioned yet.

Style programming in BibTeX++ If we have to define a new language for BibTeX++ it should be a clearer language than bst.

A good candidate is to choose a domain specific language (dsl). From a programmer point of view it is yet another (less) cryptic language to learn that is still cumbersome. The expressiveness may not suit everyone's needs, and we may extend the language later to fit some usages.

On the other side, if we want a language as expressive as any another computer language, why not use such a language? If we choose an object oriented language, all the domain specific aspects could be seamlessly hidden in objects dealing with all the bibliographical stuff.

Since in BibTeX++ there is no particular performance requirement, this solution is acceptable.

The next question is to select an implementation language. We want BibTeX++ to be portable, programmed in a clean language from a syntax and object point of view that can deal with big programs. The language should come with a lot of standard libraries to deal with all the modern programming ways (Unicode, Internet, ...), and widespread enough to avoid the *yet another weird-language to learn* syndrome.

Of course there is no one-stop answer and we cannot escape some trade-offs. From our point of view, Java has been considered as a good candidate.

The BibTeX++ core is thus directly written in Java for expressiveness and simplicity. All the generic library functions and classes to deal with bibliographies in BibTeX++ have been rewritten in Java too. BibTeX++ can run on every machine for which we have a run-time and a compiler.

For internationalization, Unicode is natively accepted in Java, we inherit all the Java locale stuff and even specialized collators for international sorting so important in BibTeX.

Of course choosing a language different than the original bst language does not solve at all the BibTeX compatibility issue. This one can be solved by implementing BibTeX as an add-on in BibTeX++, which would remove a lot of interest in BibTeX++, or add a translation process from bst to the new programming style in BibTeX++.

This last approach is more challenging but far more interesting since the translated old bst styles can be used as the basis of new style developments. This translation process is more deeply described in § 'Software architecture of the bistro bst to Java compiler'.

Extending BibTeX++ further: plug-ins and meta-plug-ins Extensions in the old BibTeX to deal with new concepts are quite difficult to develop since code must be added directly in the BibTeX source.

In a modern tool, it is mandatory to use a more incremental and tractable approach even for an inexperienced user by allowing loadable add-ons or plugins instead of needing to dig into the code to change its behavior. A plugin is a piece of code (a module) that can be added to BibTeX++ to increase its functionality without having to change the native BibTeX++ infrastructure.

Plugins should be able to modify any BibTeX++ behavior without corrupting the original design. Of course there is compromise to be found between expressiveness and complexity.

We present further the kind of extension one can expect with the plugin concept applied to different parts of BibTeX++ (described later in § 'Overview of the BibTeX++ architecture').

Parsers A natural extension needs to regard the input syntax that BibTeX++ can accept. New parsers can be written and used to change any information source into some internal abstract representations of the tool.

The bibliographical source could be changed to directly use the CiteSeer database [22] through the Internet, use some xml database or any other bibliographical database tool instead or in addition to the old `.bib` syntax database.

The input selection, normally read from the `.aux` file to pick `\cite` information, could be extended to deal with OpenOffice, DocBook or Word™ documents.

Prettyprinters Since BibTeX++ can be seen as a parameterized compiler that deals with some internal bibliographical representations, it could be nice to target other output formats than LaTeX such as an OpenOffice, DocBook or Word™ document syntax or to automatically internationalize a style from one language to another.

In an easier way, BibTeX++ could be used to translate a `.bib` file to another bibliography database format or apply some basic manipulations on bibliographical databases (sorting, merging, ...).

Style transformation More generally, adding some features to already existing styles is useful to revive some of the old-fashioned styles; for instance, adding new `url` records to old styles written before the Internet wave.

To make a bibliographical database or a researcher publication list [19] available on the Internet it could be useful to be able to fetch, close to the typeset bibliographical notice, the `.bib` record itself that was used to generate the notice if someone wants to cite it in her article. This could be done in any style by using an appropriate plugin.

Metaplugins Whereas BibTeX did not evolve for many years, BibTeX++ should rapidly evolve according present programming and Internet standards. A natural way is to use mobile code concepts in BibTeX++, code that can also be downloaded by a plugin itself. Since it is a plugin that can fetch from the network other plugins it has been nicknamed 'meta-plugin' in BibTeX++.

One can embrace various future uses of this concept.

First, to write this paper we could have picked all the BibTeX++ bibliographical style from the *TUGboat* web site.

On the BibTeX++ web site we could have a page referencing all the bibliographical databases available on the Internet. A plugin associated with this page could download other plugins to deal with each database we are interested in.

If a new typesetting tool is introduced, its author could provide on her site some plugin material for BibTeX++ to be compatible with that new tool. The BibTeX++ user would only provide to the meta-plugin the plugin address.

We will see in § 'Security model' how to circumvent the obvious security issues related to this kind of mobile code in the wild.

Plugin syntax To remain close to the existing LaTeX and BibTeX interaction syntax we do not add new LaTeX macros but rely on the current ones, merely adding special keywords inside the `\bibliography` and `\bibliographystyle` macros.

Of course plugins can also be included from the BibTeX++ invocation line or with another mechanism to suit other input formats than LaTeX.

To avoid conflicts with other tools that could also use this kind of extensions, a naming space beginning with `:bibtexpp` is used.

For example if a user from the computer science community wants to directly use citations from the CiteSeer database [22], this will be chosen with

```
\bibliography{:bibtexpp:plugin:citeseer}
```

If a user wants to add a plugin style to automatically add the output of a `url` attribute if any from the database at the end of the typeset bibliography item, the following will have to be added

```
\bibliographystyle{:bibtexpp:plugin:
  add-attribute:url}
```

or more generally to add the output of some specific attributes `description` and `summary`

```
\bibliographystyle{:bibtexpp:plugin:
  add-attribute:list:description,summary}
```

or all the attributes with

```
\bibliographystyle{:bibtexpp:plugin:
  add-attribute:all}
```

Plugins themselves can be downloaded from the default BibTeX++ network repository defined in its code with, for example

```
\bibliographystyle{:bibtexpp:plugin:meta:
  beautiful-bib}
```

of from any other place by defining a URL such as

```
\bibliographystyle{:bibtexpp:plugin:meta:
  URI:http://nice.bib.org/beautiful-bib}
```

If the BibTeX++ installation is a little bit old, some plugins may be absent from the local distribution but present in the BibTeX++ Internet repository. To be able to compile documents without choking, BibTeX++ can be used in an automatic metaplugin way where any lacking plugin will be retrieved from the repository.

If other plugins have been registered on the naming global BibTeX++ directory but reside on other servers, a proxy-plugin will be downloaded to download later the real plugin code from its server.

Naming clashes should be avoided either with the current LaTeX best practice for package names or by using a hierarchical naming space mimicking Java class naming space or URL names.

Indeed a plugin call can be defined equally well in either the `\bibliography` or `\bibliographystyle` macros, but according to the plugin role, one may be more logical than the other.

The basic compatibility with LaTeX and BibTeX is based on the fact that some macros with this syntax will only generate warning in BibTeX without stopping the LaTeX compilation. Of course, the bibliography of the document will be lacking or at least incorrect, a user without BibTeX++ will be able to have an approximation of the document.

More expressiveness can be used with new macros, defined in a new package `bibtexpp`, if this compatibility is not mandatory.

Of course, a parser plugin can itself define a new plugin definition or parameter syntax to deal with other

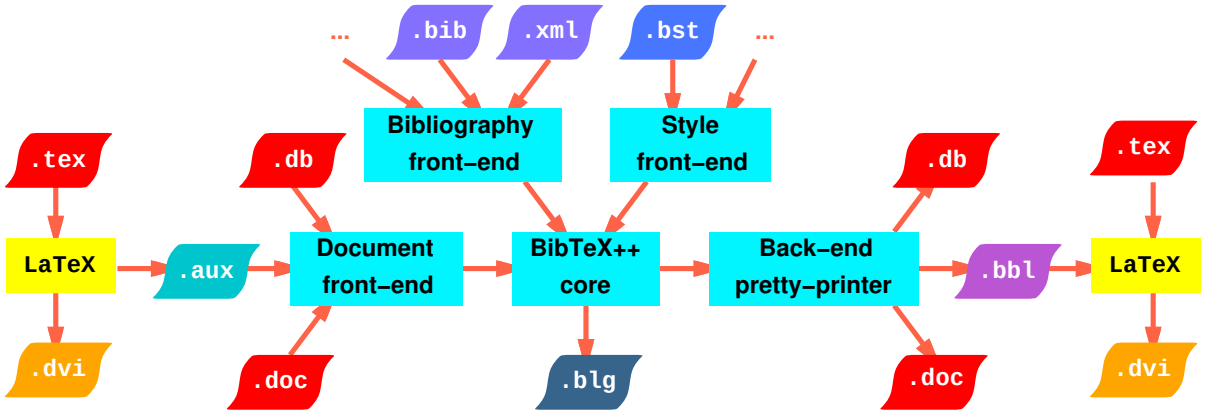


Fig. 3: BibTeX++ work-flow.

input languages such as with DocBook or Word™ documents.

Software architecture of BibTeX++

Overview of the BibTeX++ architecture In BibTeX++ we are specially interested in the software architecture point of view since the project began first with the challenging idea of compiling the bst language into something newer and more expressive.

The principal elements of this architecture are closely related to the BibTeX++ data-flow, and are summed up in figure 3.

To be more generic and more scalable, output is handled by prettyprinters of internal representations, and input is read by parsers that build a common internal representation. In this way, we have only to define a new prettyprinter or parser to deal with a new format, without having to change the BibTeX++ internals.

The parsers are made with Sablecc [12], a compiler compiler for Java written in Java. Unlike other compiler compilers (such as Java Cup, for example [14]), we do not need a specific library to use the generated compiler. So users only need Java to execute BibTeX++.

- The aux parser which tries to obtain the name of the style, database filenames, and information about languages used in the LaTeX file.
- The bib parser which converts databases to a list of Java objects.
- The bst parser which transforms a style file into a syntax tree.
- The bst compiler. It takes this syntax tree and makes a Java style file from it. It also tries to correct some common errors in bst files and optimize the output code.

BibTeX++ core The core deals with all the basic BibTeX++ infrastructure, from bibliographical concepts to house-keeping and the data structures used by various abstract internal representations.

All the BibTeX functionalities that can be used by the various bibliographical style are implemented in a core library.

Bibliography back-end This relatively simple part outputs the internal style execution into a file usable by the typesetting tool to be used. Right now, a `.bbl` file to be used by LaTeX is generated but by changing this prettyprinter other output could be generated for another tool.

Document front-end It is organized according to a strategy pattern to be able to deal with various document formats.

Right now a Sablecc parser has been written to deal with LaTeX only.

Bibliographical database front-end This front-end also follows a strategy pattern to cope with various database formats.

A Sablecc parser has been written now to read only BibTeX bibliographical database in the `.bib` format.

Caching BibTeX++ styles BibTeX++ styles can be stored on the computer running it, as with BibTeX, but can also be retrieved from the network or translated from an old BibTeX bst style.

If the first access method is quite fast, the two other ways may be deadly slow compared to it. This is why a cache architecture has been added in the BibTeX++ style pipe to avoid fetching again and again a remote BibTeX++ style or translating a style from BibTeX format on every BibTeX++ run.

Software architecture of the bistro bst to Java compiler Now we consider the bistro compiler for translating bst to Java. See figure 4.

The BibTeX stack-based input language BibTeX++ `.bst` files use a stack-based language: it is a type of lan-

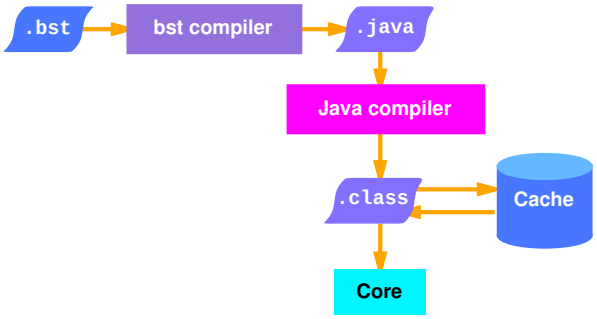


Fig. 4: Compilation work-flow.

guage where you push the data to manipulate on a stack. Also, functions store their results on this stack. The syntax uses a reverse polish notation (also called postfix notation). For example, to compute `2 + 3`, you first push `2` and `3` on the stack and then call the `add` operator, so you have to write something like this: `2 3 +`.

Some stack-based languages We may cite as stack-based programming languages:

- Forth, used in many fields, especially embedded control applications;
- PostScript, used primarily in typesetting and other display purposes. Because the majority of PostScript code is written by programs, it can also be regarded as an intermediate language;
- RPL (reverse polish LISP), the language of the HP-48 calculator. It is a run-time type-checked language with mathematical data types (it is on a calculator) and has a Forth-like syntax;
- the BibTeX style (BST) language, the stack-based language used for processing BibTeX databases.

Although stack-based languages are sometimes used as programming languages, they are more popular as intermediate languages for compilers and as machine-independent executable program representations. As intermediate languages may be listed:

- the UCSD P-code used in the UCSD system, an operating system well-known for its Pascal compiler. P-code was either interpreted or compiled to native code. We can find today P-code compilers for more recent systems;
- the Smalltalk-80 bytecode is the intermediate language of the Smalltalk-80 system.

Some other stack-based languages target several machines as machine-independent languages, notably:

- the Java bytecode output from compiling a Java file. It can be used on every system supported by Java because it will be interpreted by the JVM (Java Virtual Machine).

Unfortunately computers are mainly register machines[2] and it is not easy to implement directly and efficiently a stack-based language. That is another reason why stack-based languages are not frequently used. Today, there are 3 ways to execute a stack-based language on register machines, via:

- an interpreter. The execution is dynamic but very slow;
- a compiler that statically transforms the code into target one;
- a source-to-source translator to convert stack-based code into a high-level language that is then compiled for the target. It allows the code to be executed on many different systems if the high-level language is well-known and widely used.

BST language A style file for BibTeX is a program that formats the reference list in a certain way. For example, a style file can sort the reference list in alphabetical order using the author names, and italicize titles.

The BST language [23] is a domain-specific language using ten commands to manipulate language objects (constants, variables, functions, the stack and the entry reference list). A string constant is between double-quotes like `"abcd efgh"` and an integer constant is preceded by an `#` like `#23`. There are also three different types of variables:

- global variables, declared by `INTEGERS` or `STRINGS` commands;
- entry variables, which can be strings or integers, with a value assigned for each entry of the list;
- fields, which are read-only strings. They represent information from the curent reference item, so each one has a value for every entry.

Among the 10 BST commands available, here are some of the more interesting ones:

- `ENTRY` declares the fields (in the bibliography databases) and the entry variables. `crossref` is a field which is automatically declared (used for cross referencing) and `sort.key$` is an entry variable (used for sorting references), also automatically declared;
- `ITERATE` executes a single function for each entry in the reference list. These calls are made in the list's current order;
- `READ` reads the database file and assigns to fields their value for each entry;
- `REVERSE` performs the same action as `ITERATE` but in reverse order;
- `SORT` sorts the reference list in alphabetical order according to `sort.key$`.

2. There is probably no other popular architecture since the Transputer [16].

All these commands support defining the structure of a style file. But with them, we can not manipulate variables. This is why 37 built-in functions have been declared in BibTeX, from integer and string computations to control-flow operation (`if$`, `while$`, ...) and the `write$` which writes the top string item into the `.bbl` file (the BibTeX output). With all these built-in functions and commands some other new useful and more complex functions can be designed such as:

```
FUNCTION {and}
{   'skip$
    { pop$ #0 }
  if$
}
```

This function calculates the "logical and" between two numbers: if the first element on the stack is greater than 0 (meaning "true"), `skip$` is executed so this function returns the second element on the stack. Else, `pop$ #0` is executed which puts 0 on the stack. We can see that, even with this over-simplified example, it is not very easy to understand the BST language:

- we are not accustomed to postfix stack notation;
- the number and the type of input and output variables are implicit;
- we have to read all the control structure in reverse order.

This explains why only a few people are able to program a new style in this language. So for BibTeX++, we will have to create a more expressive style language. But because of the need for compatiblity with BibTeX, we'll have to transform this stack-based language into a standard one. So we will see how to remove the stack in a stack-based language.

Stack removing in stack-based languages A number of techniques have been proposed in the literature for this type of translation.

Source-to-source translator There are only a few source to source translators for stack-based language. The most famous research on this was done in [10, 11] where Forth code was translated into C in order to increase the portability of Forth applications. Ideally, to use a Forth application on a special system, one should develop a special interpreter. However, if one transforms the Forth into C first, the program can be used on every system where a C compiler is available. Furthermore, no deep optimization of the translator is needed since the C compiler will optimize the output code.

Practically all the other source to source translators for stack-based language are Java decompilers like *krakatoa* [24] or *mocha* [25] which try to transform Java bytecode back into Java. With this, the idea is to get back the program sources from compiled files. Nevertheless, all these translators use the same algorithm and special optimizations for Java and JVM (Java Virtual Machine) bytecode.

C variables	Stack	
x_1		↖
x_0		Top of the stack
p_0		↖
p_1		Top of the stack
p_2		upon function entry

FIG. 5: Stack mapping to C variables.

In [11] the *f2c* Forth to C translator described uses several steps to convert Forth code to C language. The first is to split the code into its functions which will be processed independently. Then, *f2c* counts the number of input and output parameters for each function. The next step is to convert the elements on the stack into C's local variables, as shown in figure 5.

In that figure, $p_0, p_1, \ldots$ represent the entry variables of each function and $x_0, x_1, \ldots$ are used like local variables. This scheme ensures that stack items that are not affected by an operation do not have to be copied around between local variables.

Then *f2c* converts each Forth primitive into a C sequence. For example, if the top of the stack resides in x_1, the translation of + will look like:

```
{
  Cell n1=x1;
  Cell n2=x0;
  Cell n;
  n = n1+n2;
  x0=n;
}
/* top of the stack now: x0 */
```

This sequence is very long, but a good C compiler can compile it to only one instruction (sometimes, it can convert several sequences into one instruction). So the translation process always works like this:

- all the useful elements are declared as C local variables and are initialized;
- the C code for the Forth primitive is generated;
- the result variables are copied back to the stack.

f2c has also to convert all the control structures. Since Forth allows the creation of arbitrary control structures, it is easiest to convert them into C `goto` instructions and labels.

This translation mechanism must know the height of the stack everywhere in the Forth code, but it is not always possible. Sometimes the stack depth is unknown. For example, the instruction `?DUP 0= IF` means that if the top of the stack is 0, replace it with the previous element on the stack, else we delete the element. So in this

case, *f2c* has to create a stack in C and use it throughout the whole function.

Compiler Source-to-source translators are not the only software which remove the stack in a stack-based language. Stack-based compilers do the same thing, but they convert this language into a low level one (often into mnemonic instructions). So their algorithms may be useful for BibTeX++.

RAFTS [10] is a framework for compiling Forth code. It tries to produce fast and efficient code, so it needs to use some optimization techniques and interprocedural register allocation to eliminate nearly all stack accesses because they slow down the execution of the program. RAFTS compiles all of Forth, including unknown stack heights.

RAFTS uses several steps to compile Forth code. The first step is to split the code into basic blocks. A basic block is a set of instructions which contains only simple primitives like literals, constants, variables, operators and stack manipulation instructions. So a basic block does not contain any branch or jump: all primitives are executed sequentially. Then RAFTS builds a data flow graph of this basic block.

After that, it converts the Forth primitives into mnemonic instructions and transforms all stack items into unlimited pseudo-registers. So all stack accesses within a basic block have been eliminated and the DAG (Directed Acyclic Graph) is now an instruction DAG. Then an instruction scheduler orders the nodes of the instruction DAG, i.e., it transforms the DAG into a list. This list is optimized to reduce register dependencies between instructions.

Now, we have a set of mnemonic blocks, but we have to connect them with control structures. Control flow splits (`IF`, `WHILE` and `UNTIL`) are easy to transform but control flow joins (`ENDIF` and `BEGIN`) are a little harder because the corresponding stack items of the joining basic blocks usually do not reside in the same register. So RAFTS needs to move some values around to have the same structure.

In order to have faster output code, three good register allocation algorithms are proposed: graph coloring register allocation [2], hierarchical graph coloring [3], and interprocedural allocators [4].

Another stack elimination in a compiler can be found in Java compilers. Today, faster and faster execution is needed for Java applications. Better Java performance can be achieved by *Just-In-Time* (JIT) compilers which translate the stack-based JVM bytecode into register-based machine code. One crucial problem in Java JIT compilation is how to map and allocate stack entries and local variables into registers efficiently and quickly so as to improve the Java performance.

LaTTe [28] is a Java JIT compiler that performs fast and efficient register mapping and allocation for SPARC machines. LaTTe converts Java bytecode (a stack-based language) to SPARC assembler. It uses severals steps for this:

- first, LaTTe identifies all control join points and subroutines in the Java bytecode, via a depth-first traversal, in order to build a control flow graph (CFG);
- then, it converts this bytecode into a CFG of pseudo SPARC instructions with symbolic registers;
- optionally, some traditional optimizations are performed;
- in the fourth step, LaTTe performs a fast register allocation, generating a CFG of real SPARC instructions.
- finally, the graph is converted into a list of SPARC instructions.

To transform the stack into registers, LaTTe uses symbolic pseudo-SPARC registers whose names are composed of three parts:

- the first character indicates the type: `a` for an address (or object reference), `i` for an integer, `f` for a float, `l` for a long and `d` for a double;
- the second character indicates the location: `s` for operand stack, `l` for local variable and `t` for temporary variables used by LaTTe ;
- the remaining number distinguishes the symbolic registers.

For example, `il2` represents the second local integer register. At the end of the algorithm, LaTTe transforms these pseudo-registers into real ones with two passes for each extended basic block:

- the backward sweep algorithm is a post-order traversal which collects information on the preferred destination registers for instructions;
- the forward sweep algorithm is a depth-first traversal which performs the real register allocation using that information.

Sometimes, we need to move some registers in order to reconcile register allocation at region join points because LaTTe uses these two algorithms on each extended basic block independently. So two blocks may not use the same register for the same item on the stack.

Globally, this method is very efficient: the output code of LaTTe is on average two times faster than the Sun JIT, and this speed comes particularly from the register allocation algorithm.

Compiling BibTeX BST styles to BibTeX++ Java styles The transformation of a typeless stack-based language into an object-based one is something quite unusual, and a bit complex. We planned a classical compiler architecture divided into several steps as shown in figure 6:

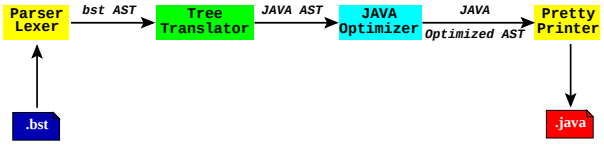


Fig. 6: bistro architecture.

- first, we use the bst parser to convert a bst file into a bst abstract syntax tree (ast) with a Sablecc grammar;
- then, in the tree translator the bst ast is transformed to a Java-like ast :
  - useful information from the tree is gathered, like function names, global variables, ... Functions are analyzed to decide if the stack can be removed and if so what is the number of input-output parameters;
  - unlike bst, Java is a type-based language, so we need to know the type of every variable in the non-stack-based session. But in some cases, it is very difficult to find this type, so if we cannot determine it, we use instead a *Cell* object: an object that can store both a integer and a string;
  - with this data, the bst ast is translated into a Java ast ;
- later, the Java ast tree is optimized with some classical transformations such as constant propagation, dependencies reduction, transformation of integer into boolean in the condition block, dead code elimination, peephole optimizations, ...;
- at last, a Java file is written by the prettyprinter and the new Java BibTeX++ style can be compiled and used.

Globally, some bst code like

```
FUNCTION {or}
{   { pop$ #1 }
    'skip$
  if$
}
```

will be converted into Java as

```
public int or( int i0 , int i1 )
{
   if( i1 > 0 )
   {
      i0 = 1;
   }
   return( i0 );
}
```

For most users and style designers, since the second block of code has been optimized for human comprehension, it should be easier to modify an existing style as a development basis of a new native BibTeX++ style.

Furthermore, because there are more Java programmers than bst ones, new arbitrary and complex styles will be easier to create with BibTeX++ than with BibTeX. If a simple interpreter had been designed instead of a translator, this would not have been possible.

More information on the bst to Java compilation can be found in [8] but 2 phases are detailed here.

Java translation First, information about functions is searched for in the bst ast. For each function, we try to obtain the name of the function, the number of input and output parameters and the possibility of removing the stack in this function (unfortunately this is not always possible).

Then the type of the arguments of all functions are inferred with type propagation from hints found in the program outside the stack, such as typed constants (a string or an integer), typed global variables, and use of BibTeX functions with well-known entry or return types. The propagation is recursively done for all the home-made functions. Propagation is done in both directions in a use-def or def-use way to deal with typed entries or output. Some further abstractions are used to follow the dependency graph even if there are stack manipulation operations in the code such as `duplicate$`, `pop$` or `swap$`.

When it is not possible to infer the type, it indicates that we will have to use a polymorphic *Cell* object which can store either a string or an integer.

Next the body of the functions and their stacks are analyzed to determine how many Java local variables we will have to use, and their types.

With all this information, the bst code is translated to Java functions where we can remove the stack by using local variables instead of stack items when possible, or generate Java functions with a Java stack when stack removal is not possible. Variables are named accordingly to figure 7. If the type has been inferred, the native Java types `String` or `int` are used instead of our polymorphic type *Cell*.

If stack removal is not possible in a given function, the stack architecture is kept, but with a Java api. The generated code looks like this:

```
public void format_bdate()
{
  stack.push(year);
  stack.push(BuiltIn.empty(
         stack.pop().getString()));
  if( stack.pop().getInt() > 0)
  {
    stack.push("there's no year in ");
    stack.push(BuiltIn.cite( bib ));
    stack.push(stack.pop().getString()
```

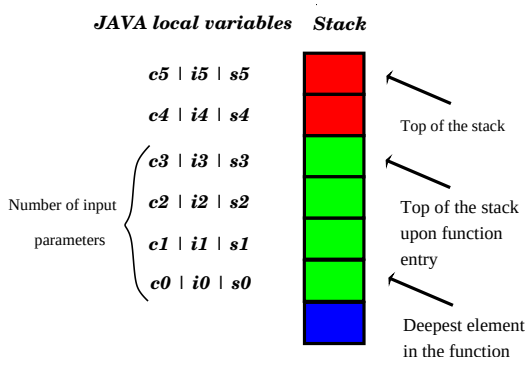


FIG. 7: Variable notations in BISTRO.

```
            +stack.pop().getString());
    System.err.println("Warning : "
                       +stack.pop()
                             .getString()
                       +".");
  }
  else
  {
    stack.push(year);
  }
}
```

If a function with a stack is called by another function, the latter will have a stack too.

Calls to any of the 37 built-in functions of BIBTEX are translated to direct JAVA code when possible (such as for +) or to calls to equivalent functions in the BIBTEX++ library.

For example, the translation of this BST function:

```
FUNCTION {format.lastchecked}
{ lastchecked empty$
    { "" }
    { inbrackets "cited " lastchecked * }
  if$
}
```

will be (without optimizations)

```
public String format_lastchecked( )
{
   String s0 , s1;
   int i0;
   s0 = lastchecked;
   i0 = BuiltIn.empty( s0 );
   if( i0 > 0 )
   {
      s0 = "";
   }
   else
   {
      inbrackets( );
      s0 = "cited ";
      s1 = lastchecked;
      s0 = s0 + s1;
   }
   return( s0 );
}
```

We can see the call to the `empty` function and the translation of the `* BST` concatenation operator to the `+` JAVA concatenation.

The BST control flow operators `if$` and `while$` are replaced by their JAVA counterparts.

After these four passes, we have a JAVA AST but this code is not optimized at all, so we need to clarify it a little.

JAVA optimization We decided to use several types of independent optimizations in order to provide a final optimization which is fully customizable by the user. We do not need very complex optimizations, because the aim of this is to increase legibility, rather than speed execution. Another reason for using simple optimizations is that the input code was written by human programmers in a not very understandable language, so they tried to write this code cleanly.

We use eight different functions for this:

- an `if` optimizer simply removes all the empty `then` or `else` blocks found in a plain BST code;
- a copy and constant propagation function removes most of the variables generated, instead of stack usage. This increases the quality of the next phase, dead code elimination;
- a dead code elimination function is associated with the propagation optimization to remove many useless definitions, because it deletes all *write after write* dependencies;
- a boolean translator: since in the BST language there is no boolean type, this optimization tries to transform integer to boolean in `if` and `while` conditions. For example, it will transform `if( BuiltIn.equal( i1 , i2 ) > 0 )` to `if( i1 == i2 );`
- other small optimizations such as peep-hole optimization and poor-man partial evaluation: for example, transforming `a1=a0+0;` into `a1=a0;` or `"some"+"thing"` into `"something"`, and so on. These optimizations only try to increase the readability of the JAVA code.

After these optimizations we get somewhat cleaner code, such as this from a previous example:

```
public String format_lastchecked( )
{
   String s0;
   if( BuiltIn.empty( lastchecked ) > 0 )
   {
```

```
      s0 = "";
   }
   else
   {
      inbrackets( );
      s0 = "cited " + lastchecked;
   }
   return( s0 );
}
```

If we look at another function from `plain.bst`:

```
FUNCTION {new.sentence}
{ output.state after.block =
   'skip$
   { output.state before.all =
      'skip$
      { after.sentence 'output.state := }
     if$
   }
  if$
}
```

This is translated to rather long Java code:

```
public void new_sentence( )
{
   int i0 , i1;
   i0 = output_state;
   i1 = after_block;
   i0 = BuiltIn.equal( i1 , i0 );
   if( i0 > 0 )
   {
   }
   else
   {
      i0 = output_state;
      i1 = before_all;
      i0 = BuiltIn.equal( i1 , i0 );
      if( i0 > 0 )
      {
      }
      else
      {
         i0 = after_sentence;
         output_state = i0;
      }
   }
}
```

but the optimizations downsize it to a more understandable function:

```
public void new_sentence( )
{
   if( output_state != after_block )
   {
      if( output_state != before_all )
      {
         output_state = after_sentence;
      }
   }
}
```

We can see in this example many different optimizations at work:

- all empty *then* blocks have been removed;
- all global variables have been propagated: the code `i0 = after_sentence; output_state = i0;` has been transformed into `output_state = after_sentence;` We no longer use any local variables;
- some built-in functions have been converted to boolean operators: the `BuiltIn.equal( i1 , i0 ) > 0` is now a simple `i1 > i0`.

So we can see here that all these transformations are pretty efficient. Indeed, the optimized function is much more readable that the non-optimized one.

Plugins and meta-plugins

Plugins architecture The strategy pattern used is based on hook mechanisms such as that used in the Emacs editor. In the original design, many hook points are chosen to enable users to insert calls to their own functions.

From a software engineering point of view, it is close to aspect programming, but in a restrictive way since all the points that can be modified are defined in advance. We think this approach is more tractable but if some features are not easy to implement with the existing hooks, more can be added since BibTeX++ is also an evolving open source program.

Lots of hook objects can be used to replace objects in the current architecture, thus modifying the global behavior, as in the following examples.

Some other specialization frameworks remain to be studied further in this context to fit future extensions, such as direct subclassing of BibTeX++ classes, aspect programming, and reflection and introspection on BibTeX++ classes. The main issue is that code complexity remains manageable and security is not endangered.

Parsers The management of the various inputs is dealt with by parsers crafted to each input format. They rely heavily on the Sablecc parser generator [12] to speed up retargeting of BibTeX++ to a new data format.

New parsers can be loaded as plugins in BibTeX++ when requested by the user.

Prettyprinters Writing plugins to output new `.bib` database files does not cause any trouble since it is a simple prettyprinter that outputs the internal database representation.

But automatically translating the BibTeX++ output into another language or targeting a new typesetting system is far more challenging. Basically, BibTeX++ is a tool able to run BibTeX++ native code or BibTeX code in

an improved way, but is not able to abstract the semantics of what a piece of BibTeX code itself really does (of course; in fact, this is an intractable issue from theoretical computer science). BibTeX++ has no idea that it is outputting an author name or a conference name: it is executing a Java procedure with a side effect that outputs a string.

Thus, we can only rely on some heuristics to deal with the prettyprinter parameterization, such as recognizing a `.bib` text output and translating it.

Since it is not possible to understand the BibTeX++ style it is not possible to modify it for retargeting the LaTeX output to another typesetting format. Instead, one can translate the LaTeX output item to another format. This is simple to do since generally bibliography styles do not generate complex TeX programs, but only text with some simple LaTeX mark-up tags. On the other hand, the *key* information, being directly managed by BibTeX and BibTeX++, can be dealt with by the plugin and output in the correct format.

Modifying the output of the bibliography style with a plugin is harder, since one should access the type of the data. For example if one wants to write a plugin to modify any existing style to display the dates in a numerical form instead of a textual one, say by replacing via regular expression mechanisms all occurrences of "November" with "11", one needs to apply this translation process only on the date field which we are ... not aware of! And what would happen if an author is named "November"?

This kind of problem is similar to automatic translation of buggy pre-year-2000 programs that cannot deal with years after 1999 to cleaner programs that can deal with them. Automatic transformations must be applied only on code that certainly deals with dates and not other numerical computations.

An interesting approach could be to type the output text with the data attribute used to build this text through BibTeX++, in order to approximate the data dependence graph with a slicing approach [26] (that is, to extract from the style code only the minimal code needed to generate the value of a given variable) statically during the BibTeX compilation process or by statically analyzing the Java BibTeX++ style. In this way it could be easy to determine that a given part of the text is a textual representation of the year, the author names, or the title, and use this information to translate these texts in a representation without `\bibitem` and so on, suitable for other typesetting tools. The typeset output for LaTeX could thus be retargeted easily since we would now have in the output what is an author name, what is a surname, what is a conference title, etc.

Since we only want to know what input fields are involved on a given output field, we can use dynamic dependence graph reconstruction. Since BibTeX++ is programmed in an object oriented language, overloading of the data type class to embed this on-the-fly dependence graph construction could be easy. At the BibTeX++ output procedure, for each character, the input fields used to compute its value is known.

This is similar for example to the tainting concept of variables used in the Perl language for security reasons, to know if a variable value has been computed by using a value given by the user, or not. If yes, and that variable is used to execute a privileged operation, the programmer may refuse to execute such a dangerous thing by using tainted mode.

The automatic internationalization process uses the same approach to translate the output text from one language to another. But if we have existing bibliography styles, say, for l languages and we want to be able to translate every language to any other, we need to write $l(l-1)$ translators, which is cumbersome. Instead, if we introduce a kind of "*esperanto*" intermediate abstract representation, we only need to write l translators to this intermediate form and then l prettyprinters to each language.

Another way for classical bst crude code translation would be to use pattern matching to translate common piece of code. Of course, since no semantics recognition can be used, this method is not very adaptable.

Code transformation It is interesting to have a code transformation engine in BibTeX++ to allow plugins to modify the behavior of other styles, and implement other features.

Transformations could be done at different levels in BibTeX++, on the bst internal representation or the Java BibTeX++ style more generally, or more conservatively on the input data structures.

The code transformation itself could use the hook mechanisms already present in the code, or an aspect programming tier in the BibTeX++ infrastructure. A low level approach could be to allow plugins to modify the code (for example, adding at the end of the output routine a call to a new procedure that will output a new field) by using Java reflection and introspection.

Of course, from the security point of view this should be very carefully controlled to avoid plugin malware rewriting security checking in BibTeX++. But this can also be enforced by the underlying Java security model, which we turn to now.

Security model In a tool such as this, where code can be downloaded by foreign documents automatically and transparently from alien servers, security sounds a little scary. It could be easy to design BibTeX++ viruses.

Hopefully, BibTeX++ is written in Java, which implements a sensible security model controlling in a centralized way the execution of arbitrary code [18] at a

very fine level.

We use this mechanism that has been tailored to allow secure execution of programs retrieved from the Internet (applets) in web browsers in a similar way.

It is used in BibTeX++ to avoid plugin code accessing sensitive local resources, modifying the security infrastructure of BibTeX++, and other obvious concerns.

At installation, the `policy` file contains

```
grant {
permission java.util.PropertyPermission
  "bib_max", "read,write";
};

grant codeBase "file:${bib_lib}" {
permission java.io.FilePermission
  "<<ALL FILES>>", "read,write,execute";
permission java.util.PropertyPermission
  "bib_cache", "read";
permission java.util.PropertyPermission
  "bib_lib", "read";
permission java.util.PropertyPermission
  "java.class.path", "read";
};
```

All classes except those inside the BibTeX++ library (thus, plugins and styles) can only access one environment variable: `bib_max`. It is the maximum length of a string in a style.

Performance results With the power of modern computers compared with the older computers at the beginning of BibTeX, we may think that compiling a bibliography must be quite fast and not significant in the composing time.

But although the original BibTeX is quite simple and fast, BibTeX++ is noticeably more complex with its compiler engine and various optimization phases. One could ask if it is still fast enough.

Some tests were made on a Athlon XP 2000+ PC with 512 MB of RAM with *j2re* (Java 2 Runtime Environment) version 1.4.1 for Linux. The BibTeX++ programs were compiled by the Sun `javac` with the `-O` option to optimize the code.

Besides the performance evaluation, the compatibility of bistro with BibTeX styles has been investigated by compiling all 152 styles in the MiKTeX distribution. This allowed us to find and correct some bugs in our software, as well as some bugs in old BibTeX styles.

The execution time and the size of the styles between the optimized version of bistro and BibTeX++ and the normal one is compared in figure 8. We can see that it is two times slower to optimize the Java style file. Nevertheless, the compilation of a `.bst` style file to a Java class file remains quite fast: only 3.2 seconds without any optimization and 4.4 seconds with all of them.

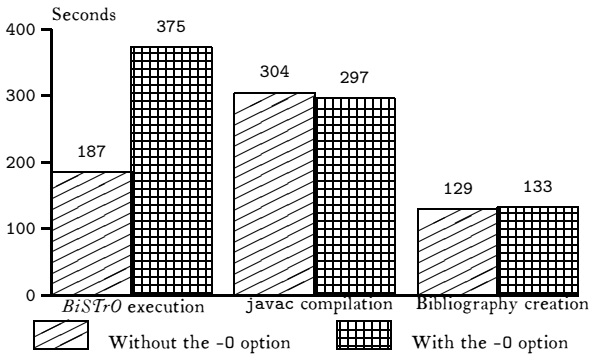


Fig. 8: Total execution time of the BibTeX++ components on 152 styles.

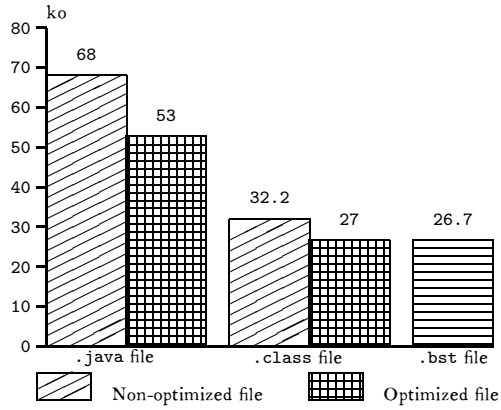


Fig. 9: Mean size of some style formats.

Furthermore, since we use a cache mechanism to compile a new bst file to Java only once, this compilation time is spent only the first time we use a bst style: all subsequent executions of BibTeX++ with the old BibTeX style will skip the compilation phase and thus run faster.

The execution of BibTeX++ is far slower than the original BibTeX (that runs in about 0.04 second on a small example), but this is not disturbing for a normal user because he just has to compile the `.bst` file once (the 3.2 seconds above), and next time the creation of the `.bbl` file will only take 0.8 second.

Finally, the optimizations are not very useful for a standard BibTeX++ user: it increases the compilation time but does not decrease the execution time.

Nevertheless they are useful for style designers. Indeed we have already seen that there are two ways of creating a new style, by modifying an existing one or by creating a new one from scratch. For both, it is easier with BibTeX++ than with BibTeX because the Java language is far more expressive, higher level and comprehensible than the bst language.

In order to help developers who want to reuse an old style, they can ask bistro to optimize its output code.

Nevertheless, as we have just seen, these optimizations do not decrease the execution time of the bibliography generation, mainly because the Java compiler also optimizes the code when we compile it. So these optimizations are mainly useful for a style designer as a reverse-engineering framework.

We can see in the histogram in figure 9 that these optimizations decrease the size of the `.java` style file by 22% and the size of the `.class` file by 16%. So they reduce the length of the code at least (it is usually easier to understand shorter code), but also increases its legibility as we have seen in the optimization code examples.

We can also note that the optimized `.class` files has almost the same size as the original `.bst` files. But since these class files were automatically generated we can imagine that hand-made BibTeX++ style files will be smaller than BibTeX ones, so they will be more easily downloadable and sharable.

Related work

There are many other open tools available to deal with bibliographies [6], but to our knowledge none tackles both extensibility and BibTeX compatibility.

Nevertheless some are closely related to our project, such as MlBibTeX and Bibulus.

MlBibTeX MlBibTeX [15] is a multilingual version of BibTeX rewritten in C. The database files and behavior remain mostly compatible with BibTeX with small extensions.

The cited article introduces in a well documented way the domain and issues related with bibliography internationalization and typography.

The main point of MlBibTeX is the introduction of language switches inside the bibliography items to be able to choose the most correct translation given by the author according to the current language, such as different notes or different transliterations of an author name.

Some other fields, such as the dates and so on, are also naturally translated.

Some extensions are planned, such as using Unicode and a localized sort, but extensibility relies on adding features in the code.

Bibulus Another tool tackles multilingual bibliographies with an extensible framework, but without a BibTeX compatibility for styles: Bibulus [27].

As with BibTeX++ written in Java, Bibulus is written in a language that can deal natively with Unicode: Perl. This allows dealing with all the world's languages. Since collators are also available, sorting can be done according to the requested language.

The input database format uses XML but a tool has been written to translate BibTeX's native `.bib` files to the Bibulus format. The format is typed more strongly, to ease further internationalization. For example the gender of the author is defined to allow for grammatical variation in some languages.

Although Bibulus is right now targeted at the LaTeX environment, other input or output formats could easily be added.

Some style parameters can be written directly in the source document to change the behavior and new items can be added to a citation to override or specialize some points of the bibliography for this particular document, or to add an annotation in the current context and language.[3]

The extensibility is based on two methods. First, as in BibTeX++, there are many hooks to enable the style programmer to modify the standard behavior of Bibulus, such as rewriting things in the parser and so on. Next, since Perl is also an object oriented language, the style programmer can override some methods of Bibulus objects.

But since there is no security model in Perl beyond tainted mode, it seems difficult to allow for secure execution of styles from the hostile world.

Conclusion

BibTeX++ is an extendable tool dealing with the bibliographical area of electronic documents. It aims at extending the well known BibTeX in the LaTeX world by adding modern features such as Unicode document encoding, Internet capabilities, scalability to future usages, and future tools through plugin mechanisms and at the same time to remain compatible with plain old BibTeX.

BibTeX++ is a free software program written in Java, a clean portable object oriented language that natively handles Unicode. Since it is written in Java, BibTeX++ is ready to run on every Java-enabled computer, although the installation phase is still to be streamlined.

BibTeX++ uses advanced compiler techniques in a compiler (bistro) for recycling "dusty deck" bst and bib files. It translates a native BibTeX style written in the bst stack language to a new BibTeX++ style written in Java that can be extended further as a basis of a class of new styles. Right now, BibTeX++ has been tested on Linux and Windows™ on all the BibTeX styles found in the teTeX and MiKTeX distribution. Indeed it allowed us to find that some of these styles are incorrect.

The plugin architecture is still to be developed in the current version with a general extension framework.

Other input front-ends and output back-ends are still to be written for tools other than LaTeX, such as OpenOffice and DocBook. But once these plugins are

3. This interesting idea could be realized in BibTeX++ by writing a plugin.

written we can reach the great bibliographical unification: for example having a Word™ document with an XML bibliography database fetched from the Internet using a `.bst` BibTeX style for a journal found on the Internet.

The BISTRO compiler that currently generates JAVA code for a JAVA BibTeX library could be retargeted to other bibliographical tools such as Bibulus in Perl with its own library.

Another usage of BISTRO is to ease the development of plain BibTeX files. Since it can translate BST code into cleaner JAVA code it can be seen as a reverse-engineering tool for people more comfortable with JAVA than BST.

A code transformation framework for automatic localization of an existing BibTeX style is still to be studied, to understand the output from a given language to another one.

From a computer science point of view, it is fascinating to see how many interesting research questions are to be solved in a tool as simple at first glance as a bibliographical management system. But this must not move us away from the typography domain with some issues such as how to deal with complete mix of Latin, Arabic, Chinese, ... entries in the *same* bibliography, and so on.

Further information on BibTeX++ with its code can be found at `http://bibtex.enstb.org`.

Thanks

The authors want to thank all the students that have worked with them during their studies on the BibTeX++ project through various internships in the Computer Science Laboratory at ENSTBr: Laurent CORDIVAL, Guillaume FERRIER, and Emmanuel VALLIET who programmed the first lines and the infrastructure with Nicolas TORNERI during their first year internship (PAP 2P in 2000), Étienne DE BENOIST, Martin BRISBARRE, Aude JACQUOT, Olivier MULLER, Mathieu SERVILLAT and Mohamed Firass SQUALLI HOUSSAINI who extended it (PAP 5J in 2001), and Sergio GRAU PUERTO for the first review of stack removal.

Bibliography

[1] Per Abrahamsen and David Kastrup. AUCTeX: An integrated TeX/LaTeX environment, 2004. `http://www.gnu.org/software/auctex`.

[2] Preston Briggs. Register allocation via graph coloring. Technical Report TR92-183, Rice University, 24, 1992.

[3] David Callahan and Brian Koblenz. Register allocation via hierarchical graph coloring. In *SIGPLAN 91 : Conference on Programming Language Design and Implementation*, pages 192–203, 1991.

[4] Fred C. Chow. Minimizing register usage penalty at procedure calls. In *SIGPLAN '88 : Conference on Programming Language Design and Implementation*, pages 45–58, 1988.

[5] Patrick W. Daly. The custom-bib package. Max-Planck-Institut für Aeronomie, 2004. `http://www.ctan.org/tex-archive/macros/latex/contrib/custom-bib`.

[6] Bruce D'Arcus and John J. Lee. Open standards and software for bibliographies and cataloging, October 2003. `http://wwwsearch.sourceforge.net/bib/openbib.html`.

[7] Carsten Dominik and Stephen Eglen. RefTeX — Support for LaTeX Labels, References and Citations with GNU Emacs, 2004. `http://remote.science.uva.nl/~dominik/Tools/reftex`.

[8] Emmanuel Donin de Rosière. From stack removing in stack-based languages to BibTeX++. Diplôme d'étude approfondie, ENSTBr, September 2003. `http://www.lit.enstb.org/~keryell/eleves/ENSTBr/2002-2003/DEA/Donin_de_Rosiere`.

[9] Emmanuel Donin de Rosière. État de l'art sur les logiciels de gestion de références bibliographiques compatibles avec LaTeX et sur la suppression de la pile dans les langages à pile. Étude bibliographique de diplôme d'étude approfondie, ENSTBr, September 2003. `http://www.lit.enstb.org/~keryell/eleves/ENSTBr/2002-2003/EB/Donin_de_Rosiere`.

[10] M. Anton Ertl. A new approach to Forth native code generation. In *EuroForth '92*, pages 73–78, 1992.

[11] M. Anton Ertl. *Implementation of Stack-Based Languages on Register Machines*. PhD thesis, Technische Universität Wien, 1996.

[12] Étienne Gagnon. *SableCC, an object-oriented compiler framework*. PhD thesis, School of Computer Science, McGill University, Montreal, 1998.

[13] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[14] Scott E. Hudson. *CUP User's Manual*. Georgia Institute of Technology, March 1996.

[15] Jean-Michel Hufflen. European Bibliography Styles and MlBibTeX. In *EuroTeX 2003* (this volume), ENST Bretagne, France, June 2003.

[16] Inmos. *The Transputer Databook*, 1989.

[17] ISO. *ISO R77: Référence bibliographiques : Éléments essentiels*, 1958.

[18] Java security, 2003. `http://java.sun.com/security`.

[19] Ronan Keryell. Publication list, 2004. http://www.lit.enstb.org/~keryell/publications/biblio/html.

[20] Larousse. *Le Petit Larousse Illustré*, volume 1. Larousse, 2002.

[21] LaTeX — a document preparation system. http://www.latex-project.org, 2004.

[22] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Citeseer, the NEC research institute scientific literature digital library, 2002. http://citeseer.nj.nec.com.

[23] Oren Patashnik. *BibTeXing*, 1988.

[24] Todd A. Proebsting and Scott A. Watterson. Krakatoa: Decompilation in Java (does bytecode reveal source?). In *Third USENIX Conf. Object-Oriented Technologies and Systems (COOTS)*, pages 185–197, 1997.

[25] H.-P. V. Vliet. Mocha, Java bytecode decompiler, 2003. http://www.brouhaha.com/~eric/computers/mocha.html.

[26] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[27] Thomas Widmann. Bibulus — a Perl/XML replacement for BibTeX. In *EuroTeX 2003* (this volume), ENST Bretagne, France, June 2003.

[28] Byung-Sun Yang, Soo-Mook Moon, and Erik R. Altman. LaTTe: A Java VM just-in-time compiler with fast and efficient register allocation. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 128–138, 1999.

European Bibliography Styles and MlBibTeX

Jean-Michel Hufflen
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 Besançon Cedex
France
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

We explain why MlBibTeX has evolved and why the last version (1.3) provides a new language — close to XSLT — for bibliography styles. We also show that this evolution has been influenced by an investigation regarding bibliography styles used throughout European countries.
Keywords: bibliographies, multilingual features, European bibliography styles, BibTeX, MlBibTeX, bst, nbst, XML, XSLT.

Résumé

Nous présentons les raisons de la récente évolution de MlBibTeX, la dernière version (1.3) adoptant un nouveau langage — proche de XSLT — pour les styles bibliographiques. Nous montrons aussi en quoi une enquête à propos des divers styles bibliographiques en usage dans les pays d'Europe a influencé cette évolution.
Mots-clés : bibliographies, multilinguisme, styles bibliographiques européens, BibTeX, MlBibTeX, bst, nbst, XML, XSLT.

Zusammenfassung

Für die bibliographischen Styles benutzt MlBibTeX seit der Version 1.3 eine neue Sprache, die mit XSLT nahe verwandt ist. Die Hintergründe dieser Entwicklung werden im folgenden dargelegt. Dazu werden die bibliographischen Styles, die in Europa benutzt werden, ausführlich untersucht, um den Einfluss dieser Analyse auf die neue Version von MlBibTeX zu zeigen.
Stichwörter: Bibliographien, mehrsprachigen Funktionen, europäischen bibliographischen Styles, BibTeX, MlBibTeX, bst, nbst, XML, XSLT.

Introduction

Although end-users of LaTeX [27] can directly manage a bibliography for each document by means of the `thebibliography` environment, this text processor is often used with the BibTeX bibliography program [30]. BibTeX allows its users to group *bibliographical entries* in .bib files comparable to data bases. An example of such an entry is:

```
@BOOK{zelazny1969,
        AUTHOR = {Roger~Joseph Zelazny},
        TITLE = {Damnation Alley},
        PUBLISHER = {Putnam},
        YEAR = 1969}
```

Let us assume that this entry for `zelazny1969` is cited within a document. If we run LaTeX, such citations are stored in the auxiliary .aux file. Then this information is used by BibTeX to generate a file containing items cited throughout the document. If we run LaTeX again, this generated file (.bbl file) is processed as part of the whole document, and *references* will look like:

> [1] Roger Joseph Zelazny. *Damnation Alley*. Putnam, 1969.

In this reference and the bibliography of this article, we have used the plain bibliography style, that is, references are labelled with numbers. There exist many other bibliography styles, most of them described in [9, § 13.2]. The bst language, used to put bibliography styles into action, is based on handling a stack, using postfix syntax. It is described in [29] and an example is given in Figure 1: the function used within bibliography styles of BibTeX to format the date of a reference.

In the last decade, LaTeX's multilingual features have been considerably extended. Some packages suitable for one particular language — for example, french

```
FUNCTION {format.date}
{ year empty$
    { month empty$
        { "" }
        { "there's a month but no year in "
          cite$ * warning$
          month
        }
      if$
    }
    { month empty$
        'year
        { month " " * year * }
      if$
    }
  if$
}
```

(The '*' operator denotes the concatenation of strings within the bst language.)

FIG. 1: Formatting dates in the .bst language.

[7], german [32], ... — have been developed. More generally, the babel package [2] can process all the languages it knows without giving any privilege to a particular one.

Since BIBTEX does not provide as many multilingual features as LATEX does — even if the insertion of some slight multilingual features have been put into action [9, § 13.8.2 & 13.9] — we started a new implementation, so-called MlBIBTEX (for 'Multilingual BIBTEX') [14]. Let us recall that this new bibliography program is in practice compatible with 'old' BIBTEX and extends its syntax by using square brackets as syntactic delimiters.[1]

In particular, this reimplementation allows end-users to specify language-dependent optional parts: for example, let us look at the entry zelazny1969 given in Figure 2. The string given by the syntax `[...] ! french` — '!' meaning 'only' — will appear only if the corresponding reference is processed in French, that is, within a 'References' section for a document written in French, as far as possible. The syntax '`[...] : english`' is for a language change: even if zelazny2000 concerns a French translation of a book (see the value of the LANGUAGE field), some parts — the author's name and the original title — are to be processed in English. This information about 'foreign words' could be useful if LATEX hyphenates words belonging to such parts. Putting this information into the generated file does not cause errors, in the sense that switches to another language occur within the text produced by MlBIBTEX only if this language is available, either by means of a specific package or by means of a suitable option of the babel package.

Last, let us notice the use of an abbreviation for the month name: see the value of the MONTH field. It can be processed in English and yield the string 'April'. Likewise, it can be processed in French (resp. German) in which case the result will be '*avril*' (resp. '*April*').

In addition, MlBIBTEX allows users to control the use of languages precisely: each entry cited can be processed according to its own language (*language-dependent* approach) or all the entries can be processed w.r.t. the document's language (*document-dependent* approach). See [16, 22] for more details about these approaches.

Between Versions 1.1 and 1.3, MlBIBTEX's implementation has deeply changed and now we are using a new language for bibliography styles [19]. This new language — so-called nbst for 'new bst' — is close to XSLT[2] [36], the language used to process and build XML[3] documents. Here we are going to explain why. From a 'philosophical' point of view, this article completes [22], which describes all the features of MlBIBTEX Version 1.3, and [23], where we show how to make MlBIBTEX fit for a particular language.

In the first section, 'A gulp of history', we show that MlBIBTEX's first version could meet the requirements for a multilingual bibliography program, but it would result in a heavy program, in complicated bibliography styles, and the whole would be hard to maintain. Furthermore, we were able to get in touch with people originating from different countries, especially at EuroTEX conferences. So, we started an investigation about bibliographies in practice, and the second section 'Questions and answers' describes it and sums up its results. In the third section, we show why we think that the new version — which will be ready in July 2003 [21] — should be suitable at least for European bibliography styles. Besides, this new version using some features borrowed from XML opens new directions which we sketch in the last section.

Reading this article requires only a basic knowledge of LATEX and BIBTEX. It also requires some knowledge about XML. Good introductions to this meta-language are [8, 33] and [5] as a pocket guide. A more 'official' reference issued by the W3C[4] is [38].

A gulp of history

When we started the first version of MlBIBTEX in October 2000, we already knew that natural languages were an open question.[5] But we thought that making 'Ref-

1. In fact, 100%-compatible if 'old' .bib files do not include any occurrence of square brackets: see [14] for more details.

2. EXtensible Stylesheet Language Transformations.

3. EXtensible Markup Language.

4. World Wide Web Consortium.

5. In particular, we confess that reading [28] greatly impressed us about the diversity and relationships among natural languages.

```
@BOOK{zelazny1969,
        AUTHOR = {Roger~Joseph Zelazny},
        TITLE = {Damnation Alley},
        PUBLISHER = {Putnam},
        YEAR = 1969,
        NOTE = {[Titre de la premi\‘{e}re traduction fran\c{c}aise : ‘‘Les culbuteurs de l’enfer’’] !
                french},
        LANGUAGE = english}

@BOOK{zelazny2000,
        AUTHOR = {[Roger~Joseph Zelazny] : english},
        TITLE = {Route~666},
        PUBLISHER = {Deno\"{e}l},
        NUMBER = 666,
        SERIES = {Pr\’{e}sence du futur},
        YEAR = 2000,
        MONTH = apr,
        NOTE = {Nouvelle traduction de ‘‘[Damnation Alley] : english’’ par Thomas Bauduret}
        LANGUAGE = french}
```

FIG. 2: Examples of multilingual features in MlBibTeX.

erences' sections for a word processor was a narrow domain with quite simple rules, so we were able to build such references by means of substitutions applied to keywords such as *and*, *chapter*, *editor*, ... and substitutions applied to delimiters for quotations: "'...'" in British English, '«...»' in French, '„...“' in German, and so on. Anyway, we thought that European languages were sufficiently close each to others to put this solution into action. So, the first version (1.1) of MlBibTeX [14, 15] was designed this way. Even though this first version worked for several languages, we quickly realised that using only substitutions was incorrect. The simplest example is given by dates: in French, English and German, the month name within a date is put first, before the year. In Hungarian, the year comes first:[6]

September 2003	(English & German)
septembre 2003	(French)
2003 szeptember	(Hungarian)

We were able to fix this problem by extending the `format.date` function within bibliography styles. Specifically, we were able to add a test about the language used when we are building a reference. But let us recall that the bst language is not modular: several bibliography styles cannot share the definition of a single function put in one file. At this time we decided to enrich our implementation of the bst language by adding an 'INCLUDE' directive. This statement would look for a file and get access to the bst definitions included in it. But adding this directive was like applying a patch. Besides, although we were able to provide a correct multilingual implementation of the `format.date` function for most languages including Hungarian, at least no one but Hungarians signalled an error, we could see that we might have to rewrite it if there existed a language with a special way to put dates. The *truly* modular and progressive solution to this problem was to be able to define a *standard* way to put dates:

```
<function name="format.date">
  ...
</function>
```

and a *specific* function for languages putting dates in another way:

```
<function name="format.date"
          language="hungarian">
  ...
</function>
```

We wrote skeletons using an XML-like syntax as above, but at this stage, we had not decided yet to use a XML-like language for bibliography styles. Our idea was just to improve the bst language. To be able to perform experiments for this goal, we developed an interpreter of bst using Scheme. In particular, this functional language being interactive, this prototype would help users learn the bst language in an incremental and interactive way [17].

But when we demonstrated this prototype at the DANTE[7] conference in the fall of 2002, we were told

6. In fact, we wrongly put a date in Hungarian in [16]. We thank Gyöngyi Bujdosó, Peter Szabó and Ferenc Wettl, who informed us of this mistake and the correct way to put a date in Hungarian.

7. *Deutschsprachige Anwendervereinigung TEX e.V.*, TEX group of German-speaking users.

what we already knew: the bst language has difficult syntax, it is not user-friendly, and its variables are global only — it is wholly related to handling a stack. As a practical matter, it is easy to introduce small changes in an existing bibliography style (as shown in [9, § 13.8]), but programming the whole of a new bibliography style is tedious. Thus, why shall we enrich such a language? To make programming in it more difficult? In addition, there were already attempts to replace it: a new bibliographic module [13] suitable for ConTEXt, Hans Hagen's format [11], a reimplementation of BibTEX in Common Lisp [26], an interpreter of the bst language in Java [25], and works based on converting bibliographical (.bib) files to XML files [10].

For the rest of 2002, we explored three directions. First, an extended syntax for person and organisation names within fields such as AUTHOR and EDITOR, was tried in MlBibTEX's Version 1.2[8] [18]. Second, a project done by two graduate students extended BibTEX so that it could process bibliography files in Unicode [34]. Our goal was not to write a new program which would be practically usable, but only to sum up all the problems related to using Unicode. The results of this project are given in [24].

Third, we again considered the DTD[9] given in [16]. Originally, we designed this DTD in order to compare MlBibTEX's expressive power to a specification of multilingual bibliographical entries designed from scratch with tools originating from XML. We tried to use XSLT as a new language for bibliography styles, and XSL-FO[10] [40] to build printable outputs for bibliographies. In both cases, we took advantage of the pattern-matching XSLT provides via the `match` attribute of the `xsl:template` tag. Also, XPath expressions [35] were very suitable when we were moving within an XML tree for an entry. Still, it was difficult to implement multilinguism, even if XML knows an `xml:lang` attribute whose value is the code of a language. In particular, it was difficult to implement a refinement for a particular language, as shown above for the `format.date` function. Using modes (by means of the `mode` attribute of the `xsl:template` tag) provided by XSLT was not very convenient, and incorporating tests about the language used would have complicated our templates just like the same kind of tests in bst programs.

In addition, there were other drawbacks: for example, the problems caused by text nodes with whitespace characters [36, § 3.4], although this only affected the readability of generated files. As a second example, some operations were difficult by using the functions provided by XPath: abbreviating a first name, implementing the `change.case$` function of BibTEX, that converts a string to lowercase characters only or uppercase characters only. On the other hand, XML is becoming standard, and we thought that developing a new version close to XML was interesting.

That is why we decided to define a new language, close to XSLT, but not equal to XSLT. This new language is very close to XSLT, so we think that learning it should be easy for people knowing XSLT. Besides, work on defining further versions of XSLT is still in progress, as is defining interfaces between XML trees and programming languages. Likewise, we hope that the development of nbst is not finished yet. Thus we can think that in a future version, nbst will become equal to a new version of XSLT. Likewise, we are conformant to the paths specified in XPath, but defined our specific functions: for example, the `abbreviate` function for abbreviating a first name.[11] Dealing with strings should be eased in Version 2.0 of XPath [42], so a future implementation of MlBibTEX may be fully conformant to the XPath library. Anyway, before joining MlBibTEX and XSLT, our work shows what is needed — and not provided by XPath and XSLT — to put a multilingual bibliography program into action nicely.

Within this new framework, the result of parsing a .bib file is a tree built according to DOM,[12] the model issued by the W3C to represent an XML document as a tree (cf. [33, p. 306–308] or [37]). As an example, Figure 3 shows the representation w.r.t. DOM of the entry `zelazny1969` given in Figure 2. Let us notice that this choice allows us to avoid the problems caused by blank text nodes, because our parser rules out whitespace characters between two fields' specifications.

Questions and answers

Readers interested in typographical conventions regarding the typesetting of bibliographies can consult [4, § 15.54–15.76] or [12, p. 53–54]: these books are good references for the English language and give some information about bibliographies in languages other than English. But this information is not comparable to what can be found in a book specific to a particular language. Nevertheless, we do not know every language, no more than anyone else. On another point, we got some experience after developing the first version of MlBibTEX. To learn more about bibliographies, we wrote a questionnaire about keywords and usages in bibliographies. An abridged version is given in Figure 4. The answers we have received do not cover all the European languages, but we think that the range is representative.

8. This version was only a prototype. It has not been distributed publicly.

9. *D*ocument *T*ype *D*efinition.

10. E*X*tensible *S*tylesheet *L*anguage — *F*ormatting *O*bjects.

11. All these functions are documented in [22].

12. *D*ocument *O*bject *M*odel.

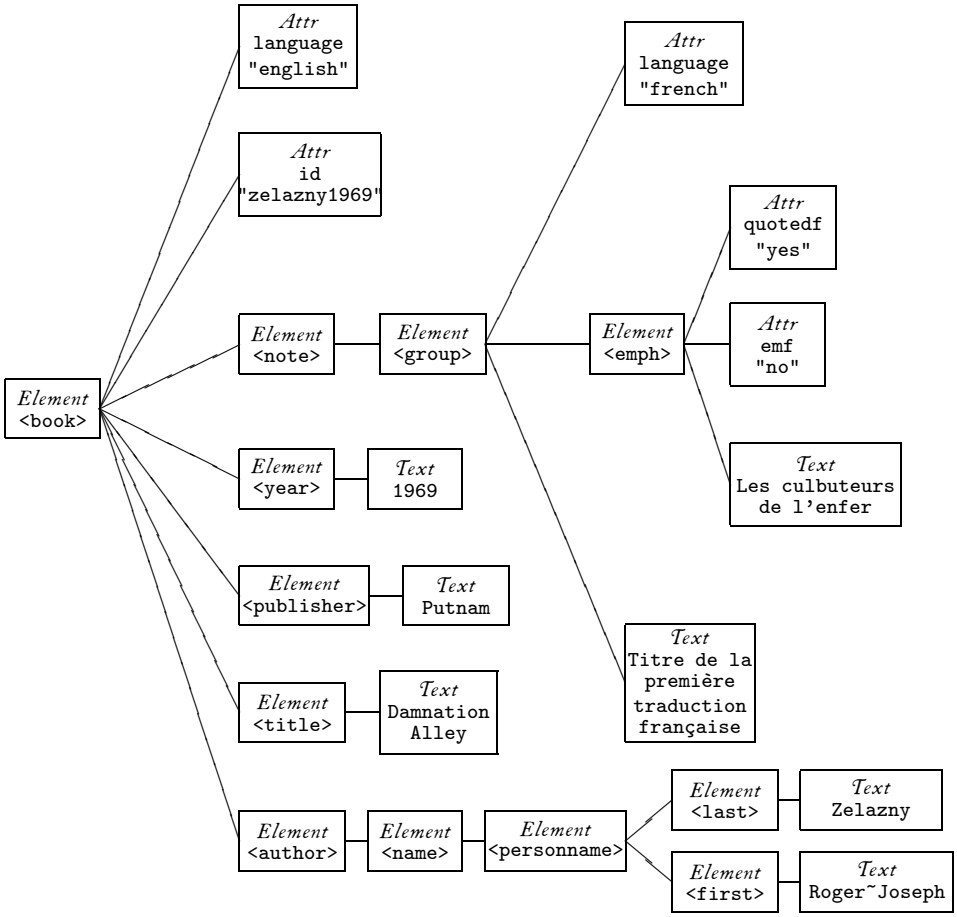


Fig. 3: A bibliographical entry as a tree according to DOM.

First of all, the notion of required and optional field is not the same in every language: for example, the publisher of a book is not required within a bibliography in Hungarian, but in contrast, the address where this book came out should be given, just before the year.

Usually, the translation of keywords does not cause problems — once we know the right word, of course. It can be solved by using LaTeX commands, for example:

```
\def\bbland{and}   (English)
\def\bbland{et}    (French)
\def\bbland{und}   (German)
\def\bbland{i}     (Polish)
...
```

The commands `\addto` and `\extras`⟨language-name⟩, provided by the babel package [2, § 6] are very suitable for implementing this kind of multilinguism:

```
\addto{\extrasfrench}{...
 \def\bbland{et}%
 ...
}
```

Likewise, the delimiters for quotations depend on languages and may be different from those used in English, but we can adapt the `bblquotedtitle` environment, already used in MlBibTeX. We show in [23] how to use this environment for first-level quotations as well as second-level ones, like:

```
\begin{bblquotedtitle}
Symphony no.~3
 \begin{bblquotedtitle}
 Symphony of Sorrowful Songs
 \end{bblquotedtitle}, op.~36
\end{bblquotedtitle}
```

which yields:

> "Symphony no. 3 'Symphony of Sorrowful Songs', op. 36"

in English. Let us notice that opening and closing this environment may do nothing: for example, the Russian language does not use quotation marks in bibliographies.

The only keyword that sometimes has no equivalent is 'of', as in:

No. 8 of *Doc Savage* Series

This quiz aims to help me put down the part of MlBibTeX's styles related to your language. [...]

Please translate the following expressions (you can give an abbreviation, provided that it is well-known):

and	edition	numero	technical report
and others	editor	of	translated by
before Christi	editors	(*one*) page	volume
chapter	in (*see below*)	(*several*) pages	with
edited by	Master's thesis	PhD thesis	

The '*in*' keyword introduces the title of a general work when you cite only a part. For example:

Greg Egan. 'Oracle'. In *the Mammoth Book of Best New SF* 14, edited by Gardner Dozois, p. 465–504. Robinson, London, 2001.

As you can see in the example above, the 'enclosing' work's title is italicised in an English-speaking bibliography, whereas the 'enclosed' work is given between quotation marks. How do you put titles of works extracted from a more general work in your language?

How are ordinal numbers abbreviated in your language? For example, in English, the abbreviations are '...st' for '...1', '...nd' for '...2', '...rd' for '...3', '...th' otherwise.

Make precise how dates—as they should appear within a bibliographical reference—are put down in your language. Give some examples:

- a complete date, e.g., 26th June 2002: ...
- a date with only a month name and a year, e.g., June 2002: ...
- a date with only a year, e.g., 2002: ...

How quotations are put in your language? This question is about top-level quotations as well as enclosed quotations. In other words, how would you write the following quotation in your language?

'Symphony no. 3 "Symphony of Sorrowful Songs", op. 36'

How should the following enries be arranged? That is, which fields are viewed as quotations? which are italicised? etc. Hereafter we give some examples: entries for a manual, an article in a journal, a book, a technical report issued by an institution. [...]

Do you think that other fields could be added, in connection with your language? If yes, describe them shortly.

How are your letters alphabeticised? By 'letters', we mean letters with or without accents. [...]

Fig. 4: Our questionnaire.

because some languages use *declensions*[13] and do not need this preposition. Often a sign is used in this case: for example, the dash character ('—') in Russian bibliographies. As another example, the order is reversed in German and Hungarian bibliographies:

Perry Rhodan 5

We can observe the same in French, even though the 'of' word is translatable in this language.

Declensions may appear in a subtle form within dates: for example, 'September 2003' is translated by 'сентябрь 2003' in Russian, but '1*st* September 2003' is said '1st *of* September 2003',[14] and a different case is

13. A language uses *declensions* when a word's function is directly expressed within this word, most often by a suffix. Originally, it seems that declensions existed in all languages, and they have disappeared in some modern languages: for example, in English, French and Portuguese. Other languages such as Estonian, Czech, German, Greek, Hungarian, Russian, Polish, Slovak ... still have declensions. To illustrate this with some examples in German:

- '*mein Freund*' is for 'my friend' as the subject of a sentence:
  Mein Freund wohnt in Brest.
  (my friend lives at Brest)
- when 'my friend' is used as a direct object:
  Ich besuche meinen Freund.
  (I am visiting my friend)
- when 'my friend' is for a person who receives something:
  Ich gebe meinem Freund ein Geschenk.
  (I am giving a gift to my friend)
- when 'my friend' is the owner of something:
  Der Wagen meines Freundes.
  (the car of my friend).

14. Let us remark that we can observe the same in Portuguese: '1 *de Setembro de* 2003'.

used for the month name: '1 сентября 2003'. The Polish language presents the same feature: '*wrzesień* 2003' but '1 *września* 2003'; so does the Greek language. This is not a problem for 'usual' references in scientific texts where the precise day is usually not given, but let us assume that we are citing articles in a daily newspaper: in such a case, writing the day number down is important and a new field should be added for that. To implement this feature, we need two LaTeX commands per month name. Focusing on the Polish language, that can be done by a `mode` attribute, which behaves as in XSLT:

```
<nbst:template match="sep"
               language="polish">
  wrzesie\'{n}
</nbst:template>

<nbst:template match="sep"
               language="polish"
               mode="genitive">
  wrze\'{s}nia
</nbst:template>
```

Some expressions may depend on a *gender*: in Romanian, ordinal numbers end with 'ul' in the masculine or 'a' in the feminine. The same feature holds in Greek and Portuguese. Here too, the problem is easily solved because ordinal numbers are usually given for an edition number, so we can know the grammatical gender of this word. On the contrary, the Polish form for 'translated by' is '*przełożył*' for a masculine translator and '*przełożyła*' for a feminine one, so we cannot know the right expression *a priori*. We confess that we have not found yet a satisfactory solution to this problem.

On another point, the English language uses several suffixes for ordinal numbers: 'st', 'nd', 'rd', 'th'. The French language uses* '*er*' and '*re*' within '1er' and '1re' in the masculine and feminine, '...^{e}' for all the other cases. Other languages systematically put a period ... after the number ('1.', '2.): this is the case in Czech, Estonian, German, Hungarian, Slovak, Polish. In fact, we could consider that adding a period is the general method and refine it for English and French.

Let us consider the alphabetical order among words: there are two distinct points of view about diacritical signs.[15] In some languages — French, German, Greek, Hungarian, Italian, Irish, Spanish, Portuguese, ... — a letter with an accent or other diacritical mark is regarded as a variant of the same letter without any mark. So, this letter and its variants are alphabeticised at the same place, except that the 'bare' letter takes precedence over an accented letter. Other languages — Czech, Estonian, Polish, Romanian, Slovak, Swedish, ... — take letters with diacritical marks as distinct from 'bare' letters. This convention is the only common point among these languages: in Polish, 'a' is placed just after 'a', whereas 'ä' is placed at the end of the alphabet in Estonian. In fact, each of these languages has its own order.[16]

In addition, some groups of letters may be regarded as one, and words beginning with such groups are listed separately in a dictionary, which complicates the algorithm for the alphabetic order. The 'ch' and 'll' groups in Spanish are well-known examples. This is also the case for 'cs', 'dz', 'dzs', 'gy', 'ly', 'ny', 'sz', 'ty', 'zs' in Hungarian. In fact, the main problem is not the adaptation of the alphabetical order for a particular language, which is not difficult once this order is known, but the definition of a universal order when references originating from several countries have to coexist. This last point appears to be impossible.

MlBibTeX's new version

As mentioned in the section 'A gulp of history', parsing a .bib file results in a tree built according the DOM model within MlBibTeX's Version 1.3. For the sake of compatibility with previous .bib files, the delimiters for a quotation are those used in American English. For example, let us once again consider the entry `zelazny1969` in Figure 2. The first French title is surrounded by double quotes ("...") and this quotation is implemented by an element in the DOM tree, as we saw in Figure 3. Some LaTeX commands used within .bib files — `\emph`, `\textbf`, etc. — are processed in an analogous way.

We implemented the `nbst` language by getting some experience with public implementations of XSLT. We also implemented our version of XPath with our specific functions. An example of a template formatting dates is given in Figure 5. The effect of the `nbst` tags can easily be guessed from the corresponding tags in XSLT. This template has to be completed by pattern-matching on month names, for example:

```
<nbst:template match="jan">
  \bbljan{} <!-- This LaTeX command should
                 be defined for the current
                 language. -->
</nbst:template>
```

and can be redefined for the Hungarian language — or for any language — by using a `language` attribute as shown below:

15. The English language does not have this problem: only the grave accent is sometimes used in poetry and the diaeresis in some old texts.

16. Among all the answers to our questionnaire, the language that seemed to us to be the most unusual example about letter ordering is Estonian, where 'z' and 'ž' are alphabeticised before 't', 'u', 'v', 'w'.

```
<nbst:template name="format.date">
  <nbst:choose>
    <nbst:when test="year">
      <nbst:choose>
        <nbst:when test="month">
          <nbst:apply-templates
            select="month"/>
          <nbst:text> </nbst:text>
          <nbst:value-of select="year"/>
        </nbst:when>
        <nbst:otherwise>
          <nbst:value-of select="year"/>
        </nbst:otherwise>
      </nbst:choose>
    </nbst:when>
    <nbst:otherwise>
      <nbst:if test="month">
        <nbst:warning>
          There's a month but no year in
          <nbst:value-of select="@id"/>
        </nbst:warning>
        <nbst:apply-templates select="month"/>
      </nbst:if>
    </nbst:otherwise>
  </nbst:choose>
</nbst:template>
```

FIG. 5: Putting a date with nbst.

```
<nbst:template name="format.date"
               language="hungarian">
  ...
</nbst:template>
```

As we discussed above, in the section 'Questions and answers', European bibliographies use a common framework, but we have to be able to use a special method for a particular language. To do that, a template with the `language` attribute has higher priority than a template without. This feature can also be useful to implement stylistic differences, that is, titles to be written using italicised characters instead of quotations or *vice versa*.

The implementation of the difference between language- and document-dependent approaches uses an attribute, too. Let us assume that the entries to be put in the bibliography of a document are the children of a tree. In 'classical' XSLT, such children can be processed recursively by means of the `xsl:apply-templates` tag. The same behaviour holds within nbst, but it is ruled by a so-called `use-language` attribute.

- The following statement:

  ```
  <nbst:apply-templates
    use-language="$document-language"/>
  ```

  means that the language used to process all the children — that is, the entries — is given by the `document-language` variable,
- whereas assigning the default `*self*` value to this variable:

  ```
  <nbst:apply-templates
    use-language="*self*"/>
  ```

  means that each reference is to be written according to the language of the corresponding entry.

For the sake of compatibility with MlBibTeX's previous versions, the language identifiers used as values of the `language` and `use-language` attributes are unambiguous prefixes of a multilingual package or an option of the babel package.[17]

As far as we know, there is no complete implementation of the `xsl:sort` tag in XSLT, according to the orders used in natural languages for strings. nbst provides partial support by means of the corresponding element, `nbst:sort`, but if this element is used to sort strings including letters outside the current language, the result is unspecified. This should be fixed in later versions.

In order to ease the transition between 'old' bst and nbst, bibliography style designers can call functions written using bst within nbst functions. See [20] for more details.

Future

The distribution of MlBibTeX's Version 1.3 [21] will include support for Czech, Danish, Dutch, English, Estonian, French, German, Greek, Hungarian, Irish, Italian, Norwegian, Polish, Portuguese, Romanian, Russian and Slovak. In addition, we explain in [23] how to add the support for a new language.

As for future directions, we think that MlBibTeX should be usable to generate bibliographies for other formats than LaTeX, and now we are investigating this for ConTeXt [11]. In particular, this will cause an adaptation regarding language identifiers, since ConTeXt deals with ISO codes for languages and countries (`en`, `fr`, `de`, ...) [1]. Another possible word processor is Texinfo [3], used for the documentation of GNU[18] software. Likewise, using a language close to XSLT for bibliography styles should ease building bibliographies for the DocBook format [43]. We are also interested in experiencing with Ω [31], as we could fully use this last program when we are able to deal with .bib files using Unicode.

Bibliographical entries can be organised into a data base, so we plan to rewrite the XML specification of MlBibTeX entries with XML Schema [33, p. 189–193],

17. This choice of a non-ambiguous prefix allows a language identifier to get access to several ways to process a language. For example, a language identifier set to `french` works with the french option of the babel package [2, 6] as well as the french package [7].

18. GNU's *N*ot *U*nix.

which offers more expressive power than DTDs, especially for data bases.

Another improvement, useful when several bibliographical data bases are shared by several people, would be the addition of *namespaces*, already present in XML [41]. To do that, we plan to use DOM Level 2 [39], and we will have to extend the syntax of .bib files.

Conclusion

We think that an important step has been made: replacing the bst language by a new language close to a standard, and allowing multilinguism within bibliography styles. This transition should be graceful for end-users since they can reuse their .bib files. Obviously, the present program has to be extended and improved: anyway, in order to reach 'actual' multilinguism, we have to include non-European languages. But the present version is ready for use, so we now expect constructive feedback.

Acknowledgements

I am pleased to thank all the people who answered my questionnaire and helped me go thoroughly into this notion of multilingual bibliographies. I confess that I asked some of them for more and more questions, I hope that I did not waste their time. So for being able to write this article, I am indebted to Janusz S. Bień, Giuseppe Bilotta, Gyöngyi Bujdosó, Kaja P. Christiansen, Peter Flynn, Hans Hagen, Zoltán Kása, Ольга Кушнаренко, Dag Langmyhr, Gyula A. Mayer, Paweł D. Mogielnicki, Simon Pepping, Horia F. Pop, Enrico Puppo, Pedro Quaresma de Almeida, Enn Saar, Petr Sojka, Seán Ua Súilleabháin, Ἀπόστολος Συρόπουλος, Marcel Takác. Thanks to the Centre of Teaching Computer Science of the University of Franche-Comté, which has supported most of the trips related to this work. Last but not least, I thank Yannis Haralambous (Γιάννης Χαραλάμπους), who kindly made this complete version available for the conference's audience, although I finished it late. Thanks also to this last version's proofreader, Karl Berry.

References

[1] Harald Tveit Alvestrand: *Request for Comments: 1766. Tags for the Identification of Languages*. UNINETT, Network Working Group. March 1995. http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1766.html.

[2] Joannes Braams: *Babel, a Multilingual Package for Use with LaTeX's Standard Document Classes*. Version 3.7. May 2002. CTAN:macros/latex/required/babel/babel.dvi.

[3] Robert J. Chassell and Richard M. Stallman: *Texinfo. The GNU Documentation System*. http://www.gnu.org/software/texinfo.

[4] *The Chicago Manual of Style*. The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.

[5] Robert Eckstein and Michel Casabianca: *XML Pocket Reference*. 2nd edition. O'Reilly & Associates, Inc. April 2001.

[6] Daniel Flipo: *A babel Language Definition File for French*. Version v1.5e. March 2001. http://www.tex.ac.uk.

[7] Bernard Gaulle : *Notice d'utilisation du style french multilingue pour LaTeX*. Version pro V5.01. Janvier 2001. CTAN:loria/language/french/pro/french/ALIRE.pdf.

[8] Michel Goossens: *XML—A New Start for the Web*. May 2000. http://webcast.cern.ch/Projects/WebUniversity/AcademicTraining/Goossens/.

[9] Michel Goossens, Frank Mittelbach and Alexander Samarin: *The LaTeX Companion*. 1st edition. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[10] Vidar Bronken Gundersen and Zeger W. Hendrikse: *BibTeX as XML Markup*. January 2003. http://bibtexml.sourceforge.net.

[11] Hans Hagen: *ConTeXt, the Manual*. November 2001. http://www.pragma-ade.com.

[12] *Hart's Rules for Composers and Readers at the University Press*. Oxford University Press. 39th edition. 1999.

[13] Taco Hoekwater: "The Bibliographic Module for ConTeXt". In: *EuroTeX 2001*, p. 61–73. Kerkrade (the Netherlands). September 2001.

[14] Jean-Michel Hufflen: "MlBibTeX: a New Implementation of BibTeX". In: *EuroTeX 2001*, p. 74–94. Kerkrade, The Netherlands. September 2001.

[15] Jean-Michel Hufflen: "Lessons from a Bibliography Program's Reimplementation". In: *LDTA 2002*, Vol. 65.3 of *ENTCS*. Elsevier, Grenoble, France. April 2002.

[16] Jean-Michel Hufflen: "Multilingual Features for Bibliography Programs: From XML to MlBibTeX". In: *EuroTeX 2002*, p. 46–59. Bachotek, Poland. April 2002.

[17] Jean-Michel Hufflen: *Interaktive BibTeX-Programmierung*. DANTE, Herbsttagung 2002, Augsburg. Oktober 2002.

[18] Jean-Michel Hufflen: *Towards MlBibTeX's Versions 1.2 & 1.3*. MaTeX Conference. Budapest, Hungary. November 2002.

[19] Jean-Michel Hufflen: *Die neue Sprache für MlBibTeX*. DANTE 2003, Bremen. April 2003.

[20] Jean-Michel Hufflen: "Mixing Two Bibliography Style Languages". In: *LDTA 2003*, Vol. 82.3 of *ENTCS*. Elsevier, Warsaw, Poland. April 2003.

[21] Jean-Michel Hufflen: *MlBibTeX—Multilingual BibTeX*. July 2003. http://lifc.univ-fcomte.fr/~hufflen/texts/mlbibtex/mlbibtex/.

[22] Jean-Michel Hufflen: "MlBibTeX's Version 1.3". *TUGboat*, Vol. 24, no. 2, p. 249–262. TUG 2003, Outrigger Waikoloa Beach Resort, Hawaii. July 2003.

[23] Jean-Michel Hufflen: "Making MlBibTeX Fit for a Particular Language. Example of the Polish Language". *Biuletyn GUST*, Vol. 21, p. 14–26. 2004.

[24] Laurent Humbertclaude et Bastien Sénèque : *Adaptation de BibTeX à l'utilisation d'Unicode*. Rapport de fin de projet DESS. Février 2003.

[25] Ronan Keryell : *BibTeX++*. Septembre 2001. http://www.lit.enstb.org/~keryell/projets/BibTeX++/exposes/2001-09-19/.

[26] Matthias Köppe: *A BibTeX System in Common Lisp*. January 2003. http://www.nongnu.org/cl-bibtex.

[27] Leslie Lamport: *LaTeX. A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[28] Jacqueline Manessy-Guitton : « La parenté généalogique ». *In* : André Martinet, rédacteur en chef, *Le langage*, Vol. 25 : *Encyclopédie de la Pléiade*, p. 814–864. Gallimard. 1968.

[29] Oren Patashnik: *Designing BibTeX Styles*. February 1988. Part of BibTeX's distribution.

[30] Oren Patashnik: *BibTeXing*. February 1988. Part of BibTeX's distribution.

[31] John Plaice and Yannis Haralambous: *Draft Documentation for the Ω System*. March 1998. http://www.loria.fr/services/tex/english/moteurs.html.

[32] Bernd Raichle: *Die Makropakete „german" und „ngerman" für LaTeX 2ε, LaTeX 2.09, Plain-TeX and andere darauf Basierende Formate*. Version 2.5. Juli 1998. Im Software LaTeX.

[33] Erik T. Ray: *Learning XML*. O'Reilly & Associates, Inc. January 2001.

[34] The Unicode Consortium: *The Unicode Standard Version* 4.0. Addison-Wesley. August 2003.

[35] W3C: *XML Path Language (XPath). Version* 1.0. W3C Recommendation. Edited by James Clark and Steve DeRose. November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[36] W3C: *XSL Transformations (XSLT). Version* 1.0. W3C Recommendation. Written by Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman and Steve Zilles. November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[37] W3C: *Document Object Model (DOM) Level* 1 *Specification (Second Edition). Version* 1.0. W3C Recommendation. Edited by Laurent Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor and Chris Wilson. September 2000. http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929.

[38] W3C: *Extensible Markup Language (XML)* 1.0 *(Second Edition)*. W3C Recommendation. Edited by Tim Bray, Jean Paoli, C. M. Sperberg-McQueen and Eve Maler. October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[39] W3C: *Document Object Model (DOM) Level* 2 *Core Specification (Second Edition). Version* 1.0. W3C Recommendation. Edited by Arnaud Le Hors, Philippe Le Hégaret, Laurent Wood, Gavin Nicol, Jonathan Robie, Mike Champion and Steve Byrne. November 2000. http://www.w3.org/TR/2000/WD-DOM-Level-2-Core-20001113.

[40] W3C: *Extensible Stylesheet Language (XSL). Version* 1.0. W3C Recommendation. Edited by James Clark. October 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/.

[41] W3C: *Namespaces in XML* 1.1. W3C Recommendation. Edited by Tim Bray, Dave Hollander, Andrew Layman and Richard Tobin. April 2002. http://www.w3.org/TR/xml-names11/.

[42] W3C: *XML Path Language (XPath)* 2.0. W3C Working Draft. Edited by Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie and Jérôme Siméon. November 2002. http://www.w3.org/TR/2002/WD-xpath20-20021115.

[43] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.

Second Version of encTEX: UTF-8 Support

Petr Olšák
Czech Technical University in Prague
petr@olsak.net

Abstract

The UTF-8 encoding keeps the standard ASCII characters unchanged and encodes the accented letters of our alphabets in two bytes. The standard 8-bit TEX is not ready for the UTF-8 input because it has to manage the single character as two tokens. It means you cannot set the `\catcode`, `\uccode`, etc., of these single characters and you cannot do `\futurelet` of the next character in the normal way. The second version of my encTEX solves these problems.

The encTEX program is fully backward compatible with the original TEX. It adds ten new primitives by which you can set or read the conversion tables used by the input processor of TEX or used during output to the terminal, log and `\write` files.

The second version creates the possibility of converting the multi-byte sequences to one byte or to a control sequence. You can implement up to 256 UTF-8 codes as one byte and an unlimited number of other UTF-8 codes as a control sequence. All internals in 8-bit TEX work as usual if the normal "one byte encoding" of input files is used.

I think that the UTF-8 encoding will be used more commonly in the future. In such a situation, there is no other way than to modify the input processor of TEX; otherwise, the 8-bit TEX will be dead in a short time.

Résumé

Le codage UTF-8 garde les caractères ASCII inchangés et encode les lettres accentuées de nos alphabets en deux octets. Le TEX à 8 bits standard n'est pas prêt pour une entrée UTF-8 car il doit gérer les caractères comme deux octets. Cela signifie que vous ne pouvez pas changer `\catcode`, `\uccode`, etc. de ces caractères et vous ne pouvez pas faire un `\futurelet` du caractère qui suit, dans la sens habituel. La deuxième version de mon encTEX résoud ces problèmes.

encTEX est totalement compatible avec le TEX original. Il ajoute dix nouvelle primitives par lesquelles vous pouvez établir ou lire les tables de conversion utilisées par le processeur d'entrée de TEX ou utilisées pendant la sortie au terminal, et aux fichiers log et `\write`.

La seconde version donne la possibilité de convertir des séquences multi-octets vers un octet ou une commande. Vous pouvez implémenter jusqu'à 256 codes UTF-8 comme un octet and un nombre illimité de codes UTF-8 comme commandes. Toute la machinerie interne de TEX fonctionne comme si les fichiers d'entrée sont dans un «codage normal à un octet».

L'auteur pense que le codage UTF-8 va être de plus en plus courant dans l'avenir. Dans cette situation il n'y a pas d'autre moyen que de modifier le processeur d'entrée de TEX, sinon la version originale de TEX à 8 bits va disparaître sous peu.

What is encTEX?

EncTEX is a TEX extension which allows re-encoding of input stream in the input processor of TEX (before tokenization) and backward re-encoding of output stream during `\write` and output to the terminal and log. It is implemented as a patch to the change file `tex.ch`. The patches are ready for Web2C distribution on [1] and (maybe) encTEX will become a standard Web2C extension, as it is for MiKTEX. Try to use the `-enc` option on command line to test if your TEX is equipped with this extension. If not, you can get and apply the patches and rebuild TEX binaries. The patches affect TEX, eTEX, pdfTEX and pdfeTEX programs. All of these programs can make use of this extension.

The first version of encTEX was released in 1997. This version only implemented byte to byte conversion, by modifying TEX's internal *xord* and *xchr* vectors. EncTEX introduced three primitives in its first version: `\xordcode` (reads or sets the values of *xord* vector for input re-encoding), `\xchrcode` (reads or sets the values of *xchr* vector for output re-encoding) and `\xprncode` (reads or sets the values of newly introduced *xprn* vector which controls the "print-ability" of characters—it controls the possibility of the character conversion to `^^ab` form on output side). See my article [2] for more details.

The first version of encTEX was not widely used because TCX tables were re-introduced in the Web2C distribution immediately after encTEX was released. Roughly speaking, the TCX tables do the same job as the first version of encTEX, although less flexibly. There was

no reason to combine the TCX tables with encTEX.

The second version of encTEX was designed and prepared by me in December 2002 and released in January 2003. This version introduces seven more primitives so users can control the multi-byte input re-encoding and reverse output re-encoding. Groups of bytes on input stream can be converted to one byte or to a control sequence. The conversion is done before tokenization, but a control sequence generated by this conversion is not re-tokenized again and the token processor does not go to the "ignoring spaces" state after such a control sequence. The backward conversion during \write allows you to convert one byte or a control sequence to the original group of bytes.

The second version of encTEX is backward compatible with the first one, of course. Detailed documentation is available [1]. Very nice on-line html documentation written by David Nečas (Yeti) is also available [5].

Motivation

I am the maintainer of a csplain format—the basic part of the CSTEX package (for Czech and Slovak users). csplain is similar to the very well-known plain TEX format (by Don Knuth, [4]). Moreover, csplain solves the processing of all letters from Czech and Slovak alphabets. It means that CS-fonts (encoded by ISO-8859-2) are used by default instead of the Computer Modern fonts, the hyphenation tables for Czech and Slovak languages are input in the same encoding and all Czech and Slovak letters are treated as single non-composite symbols. These symbols have their \catcodes set to 11 (letter), thus they can be used in control sequences too.

Czech and Slovak alphabets are encoded by many mutually incompatible standards and pseudo-standards in various operating systems and environments. All these encodings have to be converted internally to ISO-8859-2 in csplain at the input processor level and converted back to the input encoding during \write, terminal and log output. Only this rule preserves the independence of the TEX processing from the operating system.

If the source text of the Czech or Slovak document is transported from one environment to another, re-encoding to the standard of the target environment must be done, either automatically or manually by the user. The main principle is that the Czech and Slovak characters in source text have to be displayed correctly by the operating environment before the document is processed by csplain.

I created the cstrip test in 1998 [3]. You can verify if you are really using the csplain format with this test. It verifies if TEX's input processor is set correctly depending on your operating environment: all Czech and Slovak characters have to be mapped into ISO-8859-2 and they have to be written back to the input encoding on terminal, log and \write files. The ^^ab form is not permitted for Czech and Slovak letters.

We were able to set the input processor properly for csplain in old TEX distributions. For example, emTEX used TCP tables. On the other hand, the Web2C distribution disabled its similar TCX tables in 1997, thus users were not able to implement the csplain format correctly in operating environments where different encoding of our alphabets from ISO-8859-2 were used. This was the main motivation of encTEX extension of TEX.

Now, the new encoding standard UTF-8, derived from Unicode, is used very often. The non-ASCII characters are encoded in two or more bytes. If this encoding standard is used in our operating environment then we need to be able to set multi-byte conversion in the input processor of TEX. There is no other way to carry out the cstrip test. This was my motivation for the second version of encTEX.

Multi-byte re-encoding

The detailed documentation is included in the encTEX package. Thus, only a short overview of the principles is presented here.

The second version of encTEX introduces seven new TEX primitives to define and control re-encoding between multi-byte input/output and TEX's internal representation. These are:

- \mubyte and \endmubyte primitives defining the conversions,
- \mubytein, an integer register controlling input conversion,
- \mubyteout, an integer register controlling output conversion,
- \mubytelog, an integer register controlling output to terminal and log file,
- \specialout, an integer register controlling \special argument treatment, and
- \noconvert, a primitive suppressing output conversion.

The default values of all the new registers are such that encTEX behaves compatibly with unmodified TEX (incidentally, it means zeroes).

You can set the conversion table via the pair of primitives \mubyte and \endmubyte. Examples:

```
\mubyte ^^c1   ^^c3^^81\endmubyte % Á
\mubyte ^^c4   ^^c3^^84\endmubyte % Ä
```

It means that, for example, the group of two bytes ^^c3^^81 will be converted to one byte ^^c1 (if \mubytein is positive) and this byte is converted back to byte sequence ^^c3^^81 during \write (if \mubyteout is positive) and to log and terminal (if \mubytelog is positive).

If your operating environment uses UTF-8 encoding then the two bytes ^^c3^^81 are displayed as Á. You can do the "normal things" with this character in your text editor:

```
\catcode ‘Á=11  \def\myÁsequence{...}
...
\def\run{\futurelet \next \dotest}
\def\dotest{\ifx \next Á...}
\run Áha
...
\uccode‘Á=‘Á \lccode‘Á=‘á \sfcode‘Á=999
...
```

This behavior is very desirable for the csplain format and cstrip test. You can convert your old csplain documents to the new UTF-8 encoding and you can process them with csplain in operating environments supporting UTF-8. You get absolutely the same result as in the old days. This backward compatibility is most important for me.

Next example:

```
\mubyte \Alpha      ^^ce^^91\endmubyte
\mubyte \Beta       ^^ce^^92\endmubyte
...
\mubyte \leftarrow  ^^e2^^86^^90\endmubyte
\mubyte \uparrow    ^^e2^^86^^91\endmubyte
...
```

For instance, the group of three bytes ^^e2^^86 ^^90 is now converted to the \leftarrow control sequence and this control sequence is converted back to ^^e2^^86^^90 during \write if \mubyteout $\geq$ 3. The UTF-8 encoding of math characters is implemented in this way; see the utf8raw.tex file in the encTEX distribution, and math-example.tex for more complex examples.

The UTF-8 encoding tables for encTEX were prepared by David Nečas [6]. He has made his own Python script which converts the NamesList.txt [7] with Unicode declarations of characters to the \mubyte ... \endmubyte tables. This script is included in the encTEX distribution.

There is another way to declare math symbols:

```
\mubyte \utfAlpha   ^^ce^^91\endmubyte
...
\def\utfAlpha{\ensuremathmode \Alpha}
...
\def\ensuremathmode #1{\ifmmode #1\else
   $#1$\fi}
```

This second solution is more robust because you can write math symbols in the UTF-8 encoding without needing to start math mode explicitly. Note that these symbols are displayed as natural math symbols in your text editor. I did not use this solution in my macros distributed with encTEX because this concept is not compatible with common TEX documents where all math mode switches are explicitly written.

More funny examples

You can use encTEX capabilities for purposes other than character encodings. Consider this next simple example:

```
\mubyte \TeX        TeX\endmubyte
\mubyte \copyright  (C)\endmubyte
\mubyte \dots       ...\endmubyte
```

If you write "TeX and friends" (without a backslash) then input processor of encTEX converts this stream to \TeX, ⟨*space*⟩, a, n, d, ⟨*space*⟩, f, etc. This is the desired behavior. Moreover, if \mubyteout $\geq$ 3 then the \TeX control sequence is not expanded during \write, but rather is converted back to its input byte sequence "TeX". On the other hand, if you write \LaTeX, then the input is converted to two control sequences \La\TeX, which is not desired. You can solve this problem by defining the "\La" macro or declaring:

```
\mubyte \LaTeX      LaTeX\endmubyte
\mubyte \LaTeXe     LaTeX2e\endmubyte
```

Note that both byte sequences in this example begin by the same text "LaTeX". If the two characters "2e" follow immediately then a \LaTeXe control sequence is generated (by the second line of this example) else the \LaTeX control sequence is generated. (The order of the lines in this example is unimportant.)

What happens, if this setting is active and you write \LaTeX (including the backslash)? Nothing bad. The empty control sequence before the generated control sequence \LaTeX is suppressed by encTEX, thus only the \LaTeX control sequence is output.

I implemented a program vlna for adding tildes after the Czech one-letter prepositions (v, k, s, u, o, z) entirely in encTEX using \mubyte. It correctly handles math mode (no tildes are added there). It's available in the encTEX distribution as an example of crazy applications of encTEX in the file vlna.tex.

References

[1] http://www.olsak.net/enctex.html, the main page of the encTEX project.

[2] Petr Olšák: *EncTEX—A little extension of TEX*, in: *TUGboat* 19(4), (1998), pp. 336–371.

[3] ftp://math.feld.cvut.cz/pub/cstex/base/cstrip.tar.gz.

[4] Donald Knuth: *The TEXbook*.

[5] http://trific.ath.cx/tex-mf/enctex/

[6] http://trific.ath.cx/, David Nečas's home page.

[7] http://www.unicode.org/Public/UNIDATA/NamesList.txt

ALI-BABA AND THE 4.0 UNICODE CHARACTERS
New input and output concepts under Unicode

Thomas Milo
The Netherlands
tmilo@decotype.com

ABSTRACT

New input and output concepts under Unicode: Literary, Classical and Qur'ānic Arabic in Unicode involve more characters than the Arabic keyboard can accommodate. To enter sophisticated Arabic, Basis Technology and DecoType jointly developed an IME (input method editor): ALI (Arabic, Latin Input). When used at full power, ALI overstretches conventional font technology. Only DecoType's Arabic Calligraphic Engine ACE, dubbed BABA for the occasion, handles elaborate ALI output. Therefore, the ALI-BABA pair points the way toward technology for dealing with Arabic text –from dialect transcriptions to the most complicated Qur'ānic quotation, covering all aspects of both abstract text level and rendering it to the highest possible standards of computer generated typography.

RÉSUMÉ

Des nouveaux concepts d'entrée et de sortie sous Unicode. L'arabe litérraire, classique et coranique requiert plus de caractères que le clavier de saisie arabe ne peut fournir. Pour saisir de l'arabe «sophistiqué», les socieétés Basis Technology et Decotype ont développé en commun un EME (Éditeur de méthode d'entrée) du nom de ALI (Arabic, Latin Input). Utilisé à tout son potentiel, ALI dépasse la technologie conventionnelle de fonte. Seule la technologie d'arabe calligraphique de DecoType (appelée ici BABA pour l'occasion), est en mesure de gérer efficacement la sortie de ALI. C'est pour cette raison que le couple ALI-BABA montre la voie vers une nouvelle technologie de composition arabe – des transcriptions dialectales jusqu'aux citations coraniques les plus complexes, en couvrant tous les aspects du niveau textuel abstrait et en affichant ce texte aux plus hauts standards de typographie informatique.

Editor's note: This article is rather different in style from the rest of the proceedings, as the author is not a TEX user, and prepared it with the tools he normally uses. Due to the nature of the material and pressing publication deadlines, we felt it best to print it this way, rather than take the considerable additional time that would have been necessary to typeset it in the customary fashion.

INTRODUCTION

The extent of Arabic script

The scope of Arabic or Islamic script is very broad indeed. By definition it covers the Islamic civilization, both in its historical development and in its geographical expansion.

For literary and scholarly publishing all orthographical units or graphemes are relevant, as are all stages of the orthography of the Arabic language through the ages as well as all contemporary and historical orthographies of all other languages that use Arabic script or once did so.

Computing in Arabic

In its early stages in the '70s of the last century, Arabic computing was totally focused on contemporary governmental and short term business needs. As input a small subset of Arabic graphemes sufficed; output quality, let alone any typographical quality, was of no relevance. Against that background, Arabic computing took off with the typewriter metaphor: A very limited keyboard (*input*) with a matching set of glyphs (*output*). Precision of spelling (input) and typographical accuracy (output) in Arabic computing are primitive when compared with the quality of Arabic typesetting that it is now necessarily superseding.

Figure 1: The typewriter metaphor – simplified input.

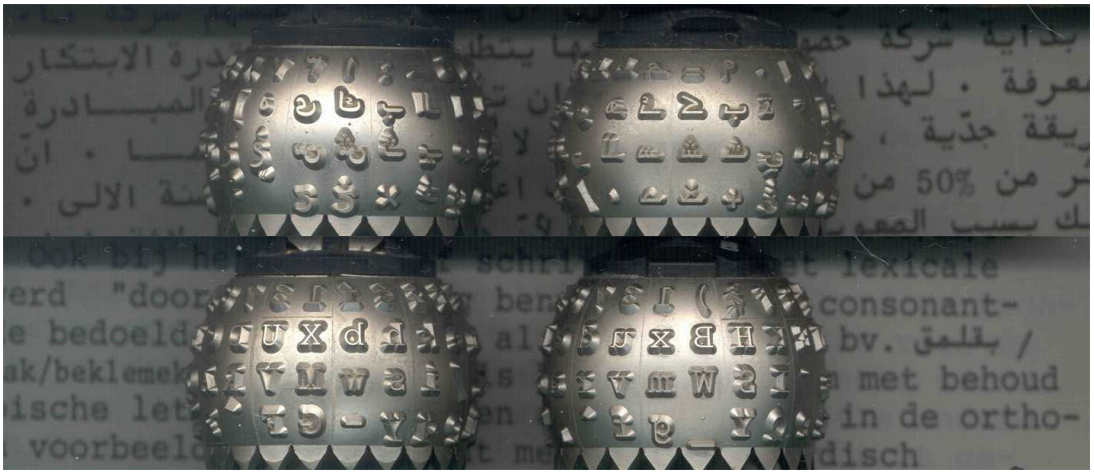


Figure 2: The typewriter metaphor – simplified output:
only two glyphs per key, using the shift mechanism that was designed to handle upper case in Latin script.

In the 19th and early 20th century, the typesetter of Arabic used no keyboard.
For input and output, the typographer had in front of him four or more cases with metal glyphs capable of representing even the most heavily annotated orthography and designed to match the quality of well-written manuscripts.

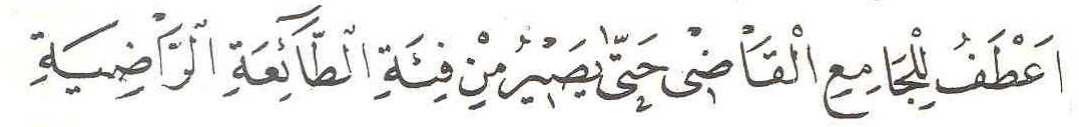


Figure 3: Fragment of nasḫ *writing taken from an Ottoman manuscript*
Computer typesetting to date can not yet register all graphemes used nor can it match its graphic quality.

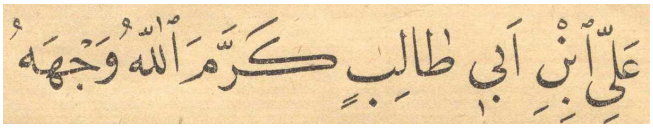


Figure 4: Fragment of nasḫ *typesetting taken from an Ottoman book.*
Note the use of superscript and subscript alifs in this particular spelling variant.

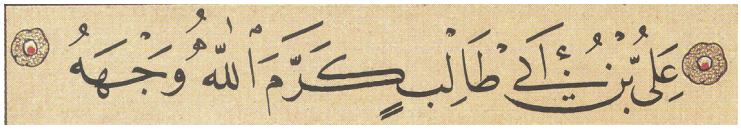


Figure 5: Nasḫ *writing by Mustafa Izzet Efendi, the 19th century Ottoman top calligrapher whose hand was the model for the typeface in figure 4.*

These glyphs represented anything from ligatures, allographs, parts of letters and superscript/subscript annotation marks (vowels, Qur'ānic punctuation).

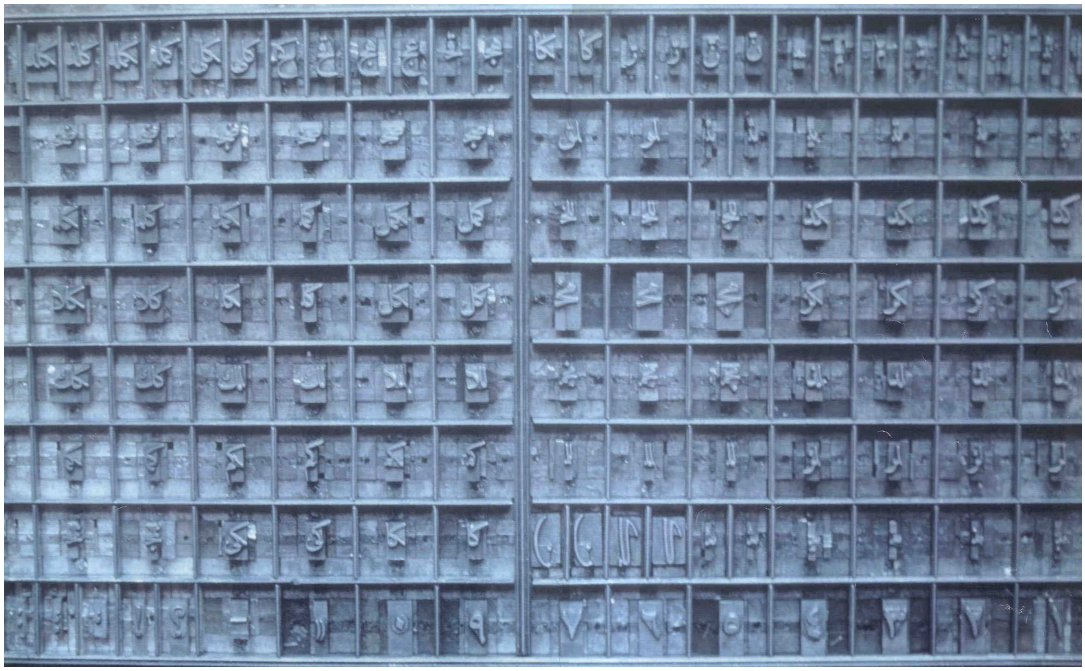


Figure 6: Detail of a case of movable types used in Egypt to typeset the Royal Qur'ān (1920's). The total glyph set amounted to about 500 pieces capturing some three dozen graphemes in their contextual variation. There was no mechanism to select the correct contextual variant (allograph): it all depended on the skill of the typesetter and his familiarity with calligraphic practice.

Today, in an age where technology allows man to walk on the Moon, for Arabic publishing the transition from mechanical typesetting to computer typography had the paradoxical effect of imposing severe limitations and causing graphic quality to plummet: input no longer handles sophisticated spelling and output no longer follows the classic esthetics of *nasḫ* script accurately.

INPUT — MORE GRAPHEMES

The present repertoire of Arabic characters in Unicode is about to reach the point that Ottoman typography reached in the 19th century: it can handle Literary Arabic, Classical Arabic and Qur'ānic Arabic — theoretically. But making practical use of this latest, richer set is no sinecure: Unicode has outgrown the keyboard based on the typewriter metaphor.

In order to make the larger repertoire of Arabic in Unicode accessible, Basis Technology and DecoType jointly developed the ALI — Arabic, Latin Input. It is an IME (input method editor) based on proven CJK (Chinese-Japanese-Korean) IME technology. ALI allows you to enter complex, fully vocalized Arabic orthography with a Latin keyboard.

This IME uses a straightforward Latin transcription. The method of transcription provides *linguistic* information and should not be confused with transliteration, which provides *graphic* information. Since the spelling principles of Arabic script differ considerably from that of any Latin orthography, transliteration tends to be unpronounceable. Transcription aims to be pronounceable.

To clarify the difference between transliteration and transcription we need to compare the Latin representation of the phrase shown in figure 3 (the computing limitations under scrutiny here, actually impose a slightly altered spelling[1]):

أَعْطَفُ لِلْجَامِعِ ٱلْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

Transliterated in ASCII:

OaEoTafu lilojaAmiEi {loqaADiy Hat~aY yaSiyru mino fi}api {lT~aA}iEapi {lr~aADiyapi

This is a meticulous but unpronounceable representation of the script.[2]

Transcribed in ASCII:

a`Tafu li l-jaami`i l-qaaDii Hattae yaSiiru min fi'äti T-Taa'i`äti r-raaDiyäti

This is an accurate representation of speech with little direct correspondence with the Arabic scripted original.[3] A comparison with the transliteration sample shows that it actually requires less Latin characters.

Our design aim was to stay within the ASCII-set (the key codes common to all computer keyboards). When transcription is typed – including all vowels using Latin characters — the correct Arabic spellings are computed simultaneously and rendered in Arabic script. In this

[1] As the reader proceeds, he will notice a number of serious Arabic font errors in the typeset text. The "guinea pig" font used for the examples is the DecoType Naskh typeface designed to Windows 95 specifications and unfortunately not capable of handling vowels and ligatures according to the rules of *nasḫ* typography: vowels do not follow to skeleton text, print on top of each other and misplaced lengthening bars (*taṭwīl — kešīde*) cause the font to behave erroneously. Moreover, the font will display default squares instead of the latest Unicode additions. These design limitations are representative for the state of the art and in this treatise they are used to illustrate the argumentation. In the second part, this font is contrasted with the DecoType Authentic Naskh typeface, designed to ACE (Arabic Calligraphic Engine) specifications: this form of computerized Arabic script not subject to traditional constraints.

[2] ASCII: American Standard Code for Information Interchange. The transcription shown is designed for computer use by Tim Buckwalter, to enable data interchange in this smallest computing character set available using even left and right parentheses to represent Arabic letters. (http://www.qamus.org/transliteration.htm).

[3] ALI converts the ASCII sequence *a+* (for *tā' marbūṭä*) automatically to the mnemonic *ä.*

process orthographic complexities and spelling rules are automatically resolved. E.g., the sound *hamz* is typed as a single apostrophe /'/ regardless the contextually varying spelling in the Arabic script:

ijraa'u-n	إِجْرَاءٌ
ijraa'uhu	إِجْرَاؤُهُ
ijraa'ahu	إِجْرَاءُهُ
ijraa'ihi	إِجْرَائِه
ijraa'aatu-n	إِجْرَاءَاتٌ
ijraa'iyyu-n	إِجْرَائِيٌّ

The transcription method used for ALI is designed to enable automatic conversion to and from Arabic orthography: a full roundtrip. As all inspected existing Latin-based transcription methods[4] are ambivalent in certain aspects (for instance, final -a and -ah cannot be reliably converted), disambiguation needed to be introduced in such cases. For instance, the final **/a/** in the transcription of **/layla/** requires additional precision.

conventional	ALI	modern Arabic
layla	**layla**	لَيْلَ
layla	**laylä**	لَيْلَة
layla	**laylae**	لَيْلَى
layla	**layla-n**	لَيْلًا (لَيْلاً)

Since the use of annotated vowels in Arabic is optional, vowel generation can be turned off partly or completely. ALI offers a configuration screen to reduce the amount of annotation. This enables the user to generate different degrees of orthographic precision in Arabic from one and the same Latin transcription:

Fully vocalized

أَعْطَفُ لِلْجَامِعِ ٱلْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

[4] http://ee.www.ee/transliteration/pdf/Arabic.pdf.

Waṣlä removed

أَعْطَفُ لِلْجَامِعِ الْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ الطَّائِعَةِ الرَّاضِيَةِ

Vowels removed

أعطف للجامع القاضي حتّى يصير من فئة الطّائعة الرّاضية

Shadda and *alif-hamzä* removed

اعطف للجامع القاضي حتى يصير من فئة الطائعة الراضية

Figure 7: ALI*-based Arabic Editor's vowel stripping controls allow the Arabic representation to become simpler*[5].

Being an Input Method Editor, ALI hinges on Arabic transcription instead of Arabic script. This facilitates conversion from one regional typographical norm to another. A classical example is the presence or absence of dots under the Arabic letter *yā'* in final position, or the stripping of dots on *tā' marbūṭä* and vice versa without loss of information. A much less known fact is that fully annotated Arabic can have variations in types of vowel marks. For this, too, ALI offers a configuration screen to manage the amount of sophistication that is required. Again, from one and the same Latin transcription:

Standard computer orthography

أَعْطَفُ لِلْجَامِعِ ٱلْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

Conventional typography has no dots on final *yā'*

أَعْطَفُ لِلْجَامِعِ ٱلْقَاضِى حَتَّى يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

Older spellings often don't use *hamzä* with vocalized *alif*

اَعْطَفُ لِلْجَامِعِ ٱلْقَاضِى حَتَّى يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

[5] This setting can also be developed for ALI to generate so-called archigraphemic Arabic representation, i.e., the archaic ambivalent skeleton text without dots and vowels.

Superscript *alif* to differentiate *alif maqṣūrä* from *yā'*

اَعْطَفُ لِلْجَامِعِ ٱلْقَاضِى حَتَّىٰ يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

Extra *sukūn* for long vowels

اَعْطَفُ لِلْجَامِعِ ٱلْقَاضِى حَتَّىٰ يَصِيرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

Subscript "dagger" *alif* for long *kasrä* (not supported by many fonts)

اَعْطَفُ لِلْجَامِعِ ٱلْقَاضْ□ى حَتَّىٰ يَصْ□يْرُ مِنْ فِئَةِ ٱلطَّائِعَةِ ٱلرَّاضِيَةِ

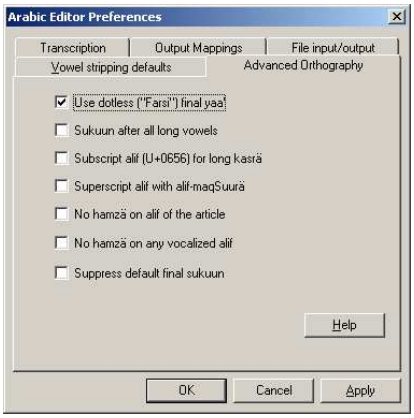


Figure 8: ALI*-based Arabic Editor's vowel elaboration controls allow the Arabic spelling to be more complex.*

With a simple expansion ALI can also provide transcriptions in any academic convention like those adopted by the US Library of Congress, British Library, E.J. Brill's Encyclopaedia of Islam, die Deutsche Morgenländische Gesellschaft, etc. etc. For instance:

reversible ASCII transcription, connecting vowels, linked words:
a`Tafu li l-jaami`i l-qaaDii Hattae yaSiiru min fi'äti T-Taa'i`äti r-raaDiyäti

reversible typographic transcription, connecting vowels, linked words:
aʿṭafu li l-jāmiʿi l-qāḍī ḥattae yaṣīru min fiʾäti ṭ-ṭāʾiʿäti r-rāḍiyäti

reversible typographic transcription librarian's style, separate words:
aʿṭafu li-l-jāmiʿ ʾl-qāḍī ḥattae yaṣīru min fiʾäti ʾl-ṭāʾiʿäh ʾl-rāḍiyäh

irreversible standard transcription, librarian's style, separate words:
aʿṭafu li-l-jāmiʿ ʾl-qāḍī ḥattā yaṣīru min fiʾäti ʾl-ṭāʾiʿah ʾl-rāḍiyah

On the other hand, ALI can compute the transcription of text in Arabic script. This is a very powerful feature that can be used to create variant spellings in Arabic to suppress misprints caused by legacy font incompatibilities. Depending on the value of the setting Rendering Compatibility Conventions, variant spellings can be generated – again! – from one and the same transcription:

Correct order (*lām-fatḥatān-alif*)

layla-n لَيْلًا

Tweaked order (*lām-alif- fatḥatān*) to avoid breaking up of *lām-alif*

layla-n لَيْلاً

In publishing, this feature of ALI can help text editors to stabilize the order in which Arabic characters are encoded. Moreover, in the case of fully vocalized Arabic, ALI makes it possible to find Arabic letters that lack a vowel marker:

(أَعْطَفُ لِلْجَامِعِ ٱلْقَاضِي حَتَّى يَصيرُ مِن فِئَة ٱلطَّائِعَة ٱلرَّاضِيَة)

Transcription of fully annotated text

a`Tafu li l-jaami`i l-qaaDii Hattae yaSiiru min fi'äti T-Taa'i`äti r-raaDiyäti

Defective annotation revealed by underscore

a`Tafu li l-jaami`i l-qA_Dii Hattae yaS_y_ru min_ fi'ä T-Taa'i`äti r-raaDiyäti

NOTE - Transcribed Arabic contains more textual information than Arabic orthography. By introducing a graphically transparent morpheme separator for Arabic Unicode text (comparable to U+00AD Soft Hyphen) it would be possible to retain transcription data that would otherwise be lost in Arabic spelling, e.g.: to prevent **/li l-jaami`i/** to revert to **/liljaami`i/**.

A totally different aspect of ALI is that it can be used to apply elongations, known by the Arabic term *taṭwīl* or the Persian *kešīde*, without having to fall back on Arabic Keyboard mode. And, since such esthetic elongation has no status in transcription, ALI automatically strips a word of *taṭwīl* when one brings up the Latin transcription.[6].

OUTPUT — MORE ALLOGRAPHS

When used for fully vocalized Classical Arabic, the Unicode text produced by ALI exceeds the capabilities of computer fonts. In fact, present font technology bars the production of this level of Arabic text.

اَعْطَفُ لِلْجَامِعِ ٱلْقَاضْ□ىْ حَتَّىْ يَص□ْيْرُ مِنْ فِئَة ٱلطَّائْعَة ٱلرَّاْضِيَة

This is where we pull a special card out of our sleeve: ACE, the Arabic Calligraphic Engine, for the occasion dubbed BABA to match ALI. This technology has the advantage that it was conceived with Classical Arabic in mind from the beginning:

1. ACE covers all Arabic Unicodes required for the most sophisticated forms of Qur'ānic, Classical and Literary Arabic;
2. ACE can handle any combination or layer of annotation marks;
3. ACE resolves clashes between annotation marks dynamically;

[6] Ideally, aesthetic elongation doesn't belong in text code, as it is covered by the font rendering mechanism when creating block justification in text layout. A practical example of this can be seen in Adobe InDesign ME with DecoType fonts (there is is called "Naskh justification").

اَعْطَفُ لِلْجَامِعِ الْقَاضِي حَتَّىٰ يَصِيرُ مِنْ فِئَةِ الطَّائِعَةِ الرَّاضِيَةِ

Where ALI produces variant sequences to suppress legacy font quirks, ACE technology intercepts these variant sequences to stabilize results (DT copyrighted). But this technology can do more: traditional writing methods are sometimes inseparable from conventional spellings, therefore:

4. ACE optimizes Unicode strings according to the conventions required for the selected typeface. For instance, modern spellings often place the Fathatan on top of the trailing *alif*, instead of before it. For a font inspired by the *muḥaqqaq* style this would be inappropriate indeed. In such cases ACE is capable of suppressing any Rendering Compatibility Conventions (legacy font tweaks) generated by ALI:

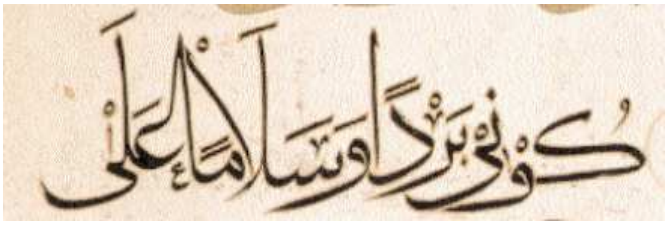


Figure 8: Qur'ân fragment(21:69: "…be cool and safe for…" in muḥaqqaq calligraphy by Aḥmad Ibn as-Suhrawardi[7].

ALI transcription

kuunii barda-n wa salaama-n `alae

ALI output in full orthography with *fatḥatān* (twin strokes) before *alif*:

كُوْنِى بَرْدًا وَسَلَامًا عَلَىْ

ALI output in reduced ("font kludge") orthography with *fatḥatān* on *alif*:

كُوْنِى بَرْدا وَسَلَامًا عَلَىْ

ACE output with typeface-specific correction of the *fatḥatān* in the lower line:

كُوْنِى بَرْدًا وَسَلَامًا عَلَىْ

كُوْنِى بَرْدًا وَسَلَامًا عَلَىْ

As mentioned in the lasts paragraph of the section about input, Unicode has a code point called *taṭwīl* (U+0640). Here we encounter the typewriter metaphor at its narrowest: the typewriter had a lengthening bar and now the computer industry has a lengthening bar, too. This crowbar-like glyph becomes a spanner in the works of contextual shaping when used in final position or following discontinuous letters (e.g., *rā'*/*zain*).

Taṭwīl is in fact an Arabic translation of the Persian-Ottoman calligraphic notion of *kešīde* "stretched", "lengthened" (and actually called *madd* by Arab calligraphers). When applied on a connection between letters it creates an elegant curve; when applied on certain final letters it gives them an esthetically pleasing swash.

ACE technology aims to treat *taṭwīl* as a *kešīde* in the true sense of the word. Therefore it uses a pair of novel technologies to make the daily use of the "lengthening bar" fool-proof:

[7] See: www.islamicart.com/main/calligraphy/late.html.

5. ACE secures the calligraphically correct execution of a *taṭwīl* or *kešīde*. *Trashideh*© technology (DT copyrighted) prevents, in a typeface-specific way, the execution of *taṭwīl* codes in calligraphically illegal positions, e.g., in initial position or, in the case of *ruqʿä* script, in all positions:

Normal font rendering any Unicode 0640 as a lengthening bar

أَعْطَفُ لِلْجَامِعِ الْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ الطَّائِعَةِ الرَّاضِيَةِ

ACE-generated *nasḫ* output with typeface-specific *kešīde* in legal positions only

أَعْطَفُ لِلْجَامِعِ الْقَاضِي حَتَّى يَصِيرُ مِنْ فِئَةِ الطَّائِعَةِ الرَّاضِيَةِ

ACE-generated *ruqʿä* output with typeface-specific omission of *kešīde*[8]

أعطف للجامع القاضي حتى يصير من فئة الطائعة الراضية

Finally, computer-generated elongation is defeated by the failure to shape vowels accordingly. Here is a solution that gets more mileage out of the elongation bar:

6. ACE technology provides for proportional annotation marks that automatically match the underlying stretched forms (DT copyrighted).

classic orthography	and "font kludge" spelling	straightened out by ACE
يَوْمِيًّا	يَوْمِيّاً	يَوْمِيًّا
يَوْمِيًّا	يَوْمِيّاً	يَوْمِيًّا
يَوْمِيًّا	يَوْمِيّاً	يَوْمِيًّا
يَوْمِيًّا	يَوْمِيّاً	يَوْمِيًّا

CONCLUSION

The ALI-BABA pair offers a new perspective for technology dealing with Arabic text — from the simplest office letter to the most complicated Qur'ānic quotation, covering all aspects of the abstract text level, and rendering it to the highest possible standards of computer generated typography[9].

Thomas Milo
Amsterdam, June 2003

[8] *Ruqʿä* script allows *kešīde* in final position, but the in the DecoType *Ruqʿä* used for this illustration this is not yet implemented.

[9] The transcription concept described in this article is implemented in the Basis technology Arabic Editor. An evaluation copy can be downloaded from this URL: http://www.basistech.com/arabic-editor/.

Generating Multiple Outputs from Ω

John Plaice
School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
plaice@cse.unsw.edu.au
http://www.cse.unsw.edu.au/~plaice

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
Yannis.Haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

Abstract

In this paper, we describe how to generate multiple outputs (DVI, PostScript, PDF, XML, ...) from the same Ω document. The Ω engine is augmented with a library for manipulating multidimensional contexts. Each macro can be defined in multiple versions, and macros can thereby adapt to differing contexts. Macros can be specialized for several different output formats, without changing the overall structure. As a result, the same document can be used to easily produce different output formats, with appropriate specializations for each of them, without having to make any changes to the document itself.

Résumé

Dans cet article nous décrivons le processus de génération de sorties multiples (DVI, PostScript, PDF, XML, ...) à partir du même document Ω. Le moteur Ω a été muni d'une bibliothèque de sous-routines dédiée à la manipulation de contextes multi-dimensionnels. Les macros TEX peuvent être spécialisés selon le format de sortie, sans changer leur structure globale. Ainsi, le même document peut, sans la moindre modification, produire facilement différents formats de sortie avec les spécialisations ad hoc.

Introduction

We present in this paper a new approach to generating typeset and structural material from Ω in a number of different output formats. This approach generalizes the existing approaches of DVI postprocessors capable of interpreting DVI `\special`'s, specialized modifications to the typesetting engine, judicious use of alternate versions of macros, and external interpreters of subsets of LATEX.

Key to this new approach is the introduction in Ω of *versioned macros* and *versioned ΩTPs* that can adapt their behavior to a dynamically running tree-structured context that permeates the entire typesetting process. As a result, when a text is to be typeset for a new output format, then new versions of macros can be written at any level, *without changing the existing macros*, thereby minimizing the amount of additional work to be undertaken.

Versioned macros and ΩTPs have ramifications well beyond the structural issues involved in generating material for different output formats: versioning the typesetting process also provides a high-level interface for multilingual typesetting, an issue that has hindered the development of the Ω system since its inception. See the paper presented at *TUG* 2003, with Chris Rowley [3], for a detailed discussion.

However, it is not sufficient simply to be able to generate different versions of macros and ΩTPs; the TEX document model is very simple, and the one-pass document manipulation approach — analogous to the Pascal language in which it was written — built into the software acts like a straitjacket when one wishes to pass as input or to generate as output significantly different document structures.

Therefore, at least three additional components need to be added to Ω in order for it to be fully adaptable to different formats. First is the ability to directly apply ΩTPs and other filters to the input stream, even before, and possibly bypassing, the macro processing stage.

Second is the ability to directly apply ΩTPs to the output stream, possibly without even generating DVI output. Third is to supply general hooks that allow the user to manipulate internal document *structure*, and not simply horizontal and vertical boxes.

In this article, we present the work that we have undertaken towards these goals. We begin with a brief background, describing what we consider to be the main contributions of existing extensions of the general TEX framework (not just to the TEX engine), and show how these different approaches all contribute to a better understanding of the general problem of generating different outputs from the same files.

The model for contexts that we have adopted was developed by Paul Swoboda in his PhD thesis [7]. It is the most highly developed presentation of *intensional versioning*, an approach to the development of software variants first proposed by the first author and William W. Wadge [6]. We give a discussion of intensional programming and versioning, then give a detailed presentation of contexts and context operators.

We then show how contexts have been integrated into Ω. To do this, new Ω primitives are introduced for creating and using different versions of macros, and for changing and manipulating the runtime context. In addition, means for having versions of internal and external ΩTPs are defined.

These technical sections are followed by a discussion of how the internals of the Ω engine should be reorganized to facilitate the generation of multiple outputs.

TEX and its Extensions

The TEX document model supposes that a stream of text, interspersed with control sequences, is to be transformed into a series of *pages*, each of which is a vertical box that contains other boxes, either vertical or horizontal. Each page is generated into DVI output, in the process losing some of the information contained in the page.

The boxes are generated on the fly. Although certain items can be stored for later use in registers, accessed much as one would in assembly programming, TEX's document model essentially consists of the following processes:

- transforming streams of characters into streams of typeset glyphs and boxes (*main loop*);
- building math lists from TEX math, then transforming the lists into streams of typeset glyphs (*math mode*);
- transforming streams of typeset glyphs, with inserted hyphenation points, into streams of horizontal boxes, corresponding to lines (*paragrapher*);
- building boxes, corresponding to tables, from alignment specifications;
- building pages from streams of boxes and glue (*page builder*).

TEX's operation is undertaken in one pass, and it is very difficult, if not impossible, to be able to manipulate intermediate data structures as they are being built.

The different extensions to TEX and the different DVI postprocessors have all taken different approaches, which is quite normal given their divergent aims.

First are the DVI postprocessors, `dvips` (generating PostScript) and `dvipdfm` (generating PDF). Each of these programs transforms DVI output, augmented with DVI `\special`'s, specifically designed for use with that program and generated by TEX through its macro mechanism, into the relevant output format.

The main advantage of this approach is that it encourages modularity, in the sense that the typesetter is separate from the pretty-printer. However, one can only put into `\special`'s information that is made available to the user. Information about intermediate data structures is not directly available, so can only be approximated.

Second is the LATEX2HTML approach. This tool does not do typesetting, rather it reorganizes the structure of the text into HTML. It does not use the TEX engine, but itself parses a large (reasonable) subset of LATEX. For parts that cannot be directly translated into HTML, such as mathematics, then it generates small LATEX files, calls LATEX, then `dvips`, then transforms then into PNG files. Although LATEX2HTML is a useful tool, in its current form it will never have access to TEX's internal data structures, since it never calls TEX.

Third, also for generating HTML, is TEX4HT, which produces HTML files that resemble DVI pages generated by TEX. TEX4HT is also standalone, but it does use TEX for parsing and typesetting the input. It makes use of extensive DVI `\special`'s.

Fourth are the extensions to the TEX engine, namely e-TEX, pdfTEX, and Ω. The e-TEX extensions focus mainly on improving the macro expansion facilities. They do not change the typesetting, but do provide the very useful ability to reparse an input sequence.

The pdfTEX extensions are two-fold. First is some experimental work simulating some of Peter Karow's *hz* program. Second, more commonly used, are the extensions to generate PDF directly rather than DVI. In addition to its new pretty-printer for TEX pages, pdfTEX provides built-in mechanisms, using *whatsit* nodes, for generating such things as PDF forms and margin items.

Although pdfTEX is practical, in the sense that one can quickly generate PDF files from a TEX file, the fact that all of the functionality is hard-coded limits the possibility for extending the same system. For example, the current pdfTEX does not allow EPS files to be included in the PDF files that it generates.

The Ω extensions are of a more general nature.

In the Ω model, before glyph selection, the character stream to be typeset is segmented and processed by a series of filters, each reading from standard input and writing to standard output. Once all of the filters are applied, the stream is passed to the standard TEX character-level typesetter.

In addition, Ω also includes an experimental pretty-printer for MathML and XML. One of the goals of this work is to provide the means for recovering structure, particularly in mathematics expressions, in TEX and LATEX files that do not have perfect markup. However, this functionality is not in any way integrated with the ΩTP mechanism.

From this discussion, one can start to elucidate what is needed. One should be able to have, in a single system:

- internal data structures corresponding to page, paragraph, line, etc., that can be *explicitly* manipulated by the user;
- input mechanisms not based on the macro language that generate these data structures;
- more advanced macro-processing and other programming languages to manipulate these data structures;
- mechanisms to pretty-print the data structures in multiple output formats;
- limiting the output-format-specific additions to the internals of the typesetting engine.

For this whole approach to work, it is important that as one moves from one mechanism to another, changing input or output formats, and what is expected of the typesetting engine, that the whole framework be sufficiently *flexible* that one does not have to completely reprogram everything. In other words, the system must adapt to a complex context with many parameters. The next few sections focus in detail on how to deal with context in Ω; they are followed by a discussion of the points raised in the above wishlist.

Intensional Programming

Intensional programming [4] is a form of computing that supposes that there is a multidimensional context, and that all programs are capable of adapting themselves to this context. The context is pervasive, and can simultaneously affect the behavior of a program at the lowest, highest and middle layers.

When an intensional program is running, there is a *current context*. This context is initialized upon launching the program from the values of environment variables, from explicit parameters, and possibly from active context servers. The current context can be modified during execution, either explicitly through the program's actions, or implicitly, through changes at an active context server.

A context is a specific point in a multidimensional space, i.e., given a dimension, the context will return a value for that dimension. The simplest contexts are dictionaries (lists of attribute-value pairs). A natural generalization is what will be used in this paper: the values themselves can be contexts, resulting in a tree-structured context. The set of contexts is furnished with a partial order $\sqsubseteq$ called a *refinement relation*.

For example, to describe Australian English, we could use the context:

```
<script:<Latin>+
 lang:<English;dialect:<Australian>>>
```

where `script` and `lang` are called *dimensions*, and `lang:dialect` is called a *compound dimension*. See below for more details.

During execution, the current context can be queried, dimension by dimension, and the program can adapt its behavior accordingly. In addition, if the programming language supports it, then contextual conditional expressions and blocks can be defined, in which the *most relevant* case, with respect to the current context and according to the partial order, is chosen among the different possibilities.

In addition, any entity can be defined in multiple *versions*, which are mappings from contexts to objects. Whenever an identifier designating an entity appears in an expression or a statement, then the most relevant version of that entity, with respect to the current context, is chosen. This is called the *variant substructure principle*. The general approach is called *intensional versioning* [6].

The ISE programming language [5] was the first language combining both intensional programming and versioning. It is based on the procedural scripting language Perl, and it has greatly facilitated the creation of multidimensional Web pages. Similar experimental work has been undertaken under the supervision of the first author with C, C++, Java, and Eiffel. And, when combined with a context server (see Paul Swoboda's PhD thesis [7]), it becomes possible for several documents or programs to be immersed in the same context.

Structuring the Context

We use the same notation to designate contexts and versions of entities. This section has three subsections. First, we define contexts and the refinement relation. Then, we define *version domains*, which hold versioned entities. Finally, we define *context operators*, which are used to change from context to context. In the following section, we will show how all of these are to be used.

Contexts and Refinement. Let $\{(\mathbb{S}_i, \sqsubseteq_i)\}_i$ be a collection of sets of ground values, each with its own partial order. Let $\mathbb{S} = \cup_i \mathbb{S}_i$. Then the set of contexts $\mathbb{C}$ ($\ni C$) over $\mathbb{S}$

is given by the following syntax:

$$C ::= \bot \mid A \mid \Omega \mid \langle B; L\rangle \quad (1)$$

$$B ::= \epsilon \mid \alpha \mid \omega \mid v \quad (2)$$

$$L ::= \emptyset \mid d{:}C + L \quad (3)$$

where $d, v \in \mathbb{S}$.

There are three special contexts:

- $\bot$ is the *empty context* (also called *vanilla*);
- A is the *minimally defined context*, just more defined than the empty one;
- Ω is the *maximally defined context*, more defined than all other contexts.

The normal case is that there is a *base value* B, along with a *context list* (L for short), which is a set of *dimension-context* pairs. We write δL for the set of dimensions of L.

A sequence of dimensions is called a *compound dimension*. It can be used as a path into a context. Formally:

$$D = \cdot \mid d{:}D \quad (4)$$

If C is a context, $C(D)$ is the subtree of C whose root is reached by following the path D from the root of C:

$$C(\cdot) = C \quad (5)$$

$$\langle B; d{:}C' + L\rangle\ (d{:}D) = C'(D) \quad (6)$$

As with contexts, there are three special base values:

- ϵ is the *empty base value*;
- α is the *minimally defined base value*, just more defined than the empty base value;
- ω is the *maximally defined base value*, more defined than all others.

The normal case is that a base value is simply a scalar.

To the set $\mathbb{C}$, we add an *equivalence* relation $\equiv$, and a *refinement* relation $\sqsubseteq$. We begin with the equivalence relation:

$$\bot \equiv \langle \epsilon; \emptyset\rangle \quad (7)$$

$$A \equiv \langle \alpha; \emptyset\rangle \quad (8)$$

$$\Omega \equiv \left\langle \omega; \sum_{d \in \mathbb{S}} d{:}\Omega \right\rangle \quad (9)$$

$$\frac{L_0 \equiv_L L_1}{\langle B; L_0\rangle \sqsubseteq \langle B; L_1\rangle} \quad (10)$$

Thus, $\bot$ and A are notational conveniences, while Ω cannot be reduced. The normal case supposes an equivalence relation $\equiv_L$ over context lists:

$$\emptyset \equiv_L d{:}\bot \quad (11)$$

$$d{:}\langle B; L + L'\rangle \equiv_L d{:}\big(\langle B; L\rangle + \langle B; L'\rangle\big) \quad (12)$$

$$L \equiv_L \emptyset + L \quad (13)$$

$$L \equiv_L L + L \quad (14)$$

$$L + L' \equiv_L L' + L \quad (15)$$

$$L + (L' + L'') \equiv_L (L + L') + L'' \quad (16)$$

The $+$ operator is idempotent, commutative, and associative. Now we can define the partial order over entire contexts:

$$\bot \sqsubseteq C \quad (17)$$

$$C \sqsubseteq \Omega \quad (18)$$

$$\frac{C \neq \bot}{A \sqsubseteq C} \quad (19)$$

$$\frac{C_0 \equiv C_1}{C_0 \sqsubseteq C_1} \quad (20)$$

$$\frac{B_0 \sqsubseteq_B B_1 \quad L_0 \sqsubseteq_L L_1}{\langle B_0; L_0\rangle \sqsubseteq \langle B_1; L_1\rangle} \quad (21)$$

which supposes a partial order $\sqsubseteq_B$ over base values:

$$\epsilon \sqsubseteq_B B \quad (22)$$

$$B \sqsubseteq_B B \quad (23)$$

$$B \sqsubseteq_B \omega \quad (24)$$

$$\frac{B \neq \epsilon}{\alpha \sqsubseteq_B B} \quad (25)$$

$$\frac{v_0, v_1 \in \mathbb{S}_i \quad v_0 \sqsubseteq_i v_1}{v_0 \sqsubseteq_B v_1} \quad (26)$$

The last rule states that if v_0 and v_1 belong to the same set $\mathbb{S}_i$ and are comparable according to the partial order $\sqsubseteq_i$, then that order is subsumed for refinement purposes.

The partial order over contexts also supposes a partial order $\sqsubseteq_B$ over context lists:

$$\emptyset \sqsubseteq_L L \quad (27)$$

$$\frac{L_0 \equiv_L L_1}{L_0 \sqsubseteq_L L_1} \quad (28)$$

$$\frac{C_0 \sqsubseteq C_1}{d{:}C_0 \sqsubseteq_L d{:}C_1} \quad (29)$$

$$\frac{L_0 \sqsubseteq_L L_1 \quad L_0' \sqsubseteq_L L_1'}{L_0 + L_0' \sqsubseteq_L L_1 + L_1'} \quad (30)$$

Rule 30 ensures that the $+$ operator defines the least upper bound of two context lists.

Context and Version Domains. When doing intensional programming, we work with *sets* of contexts, called *context domains*, written $\mathcal{C}$. There is one operation on context domains, namely the *best-fit*. Given a context domain $\mathcal{C}$ of existing contexts and a requested context C_{req}, the best-fit context is defined by:

$$\mathit{best}(\mathcal{C}, C_{\text{req}}) = \max\{C \in \mathcal{C} \mid C \sqsubseteq C_{\text{req}}\} \quad (31)$$

If the maximum does not exist, there is no best-fit context.

Typically, we will be versioning *something*, an object of some type. This is done using *versions*, simply (C, object) pairs. Version domains $\mathcal{V}$ then become functions mapping contexts to objects. The *best-fit object* in a version domain is given by:

$$\mathit{best}_O(\mathcal{V}, C_{\text{req}}) = \mathcal{V}(\mathit{best}(\operatorname{dom}\mathcal{V}, C_{\text{req}})) \quad (32)$$

Context Operators. Context operators allow one to selectively *modify* contexts. Their syntax is similar to that of contexts.

$$C_{\text{op}} ::= C \mid [P_{\text{op}}; B_{\text{op}}; L_{\text{op}}] \quad (33)$$
$$P_{\text{op}} ::= -- \mid \epsilon \quad (34)$$
$$B_{\text{op}} ::= - \mid B \quad (35)$$
$$L_{\text{op}} ::= \emptyset_{L_{\text{op}}} \mid d{:}C_{\text{op}} + L_{\text{op}} \quad (36)$$

A context operator is applied to a context to transform it into another context. (It can also be used to transform a context operator into another; see below.) The $-$ operator removes the current base value, while the $--$ operator in P_{op} is used to clear all dimensions not explicitly listed at that level.

Now we give the semantics for $C\ C_{\text{op}}$, the application of context operator C_{op} to context C:

$$C_0\ C_1 = C_1 \quad (37)$$
$$\Omega\ C_{\text{op}} = \text{error} \quad (38)$$
$$\langle B; L\rangle\ [--; B_{\text{op}}; L_{\text{op}}] = \left\langle B; L\backslash(\delta L - \delta L_{\text{op}})\right\rangle\ [\epsilon; B_{\text{op}}; L_{\text{op}}] \quad (39)$$
$$\langle B; L\rangle\ [\epsilon; B_{\text{op}}; L_{\text{op}}] = \left\langle (B\ B_{\text{op}}); (L\ L_{\text{op}})\right\rangle \quad (40)$$

The general case consists of replacing the base value and replacing the context list. First, the base value:

$$B - = \epsilon \quad (41)$$
$$B_0\ B_1 = B_1 \quad (42)$$

Now, the context list:

$$L\ \emptyset_{L_{\text{op}}} = L \quad (43)$$
$$(d{:}C + L)\ (d{:}C_{\text{op}} + L_{\text{op}}) = d{:}(C\ C_{\text{op}}) + (L\ L_{\text{op}}) \quad (44)$$
$$L\ (d{:}C_{\text{op}} + L_{\text{op}}) = d{:}(\emptyset\ C_{\text{op}}) + (L\ L_{\text{op}}), \quad d \notin \delta L \quad (45)$$

Context operators can also be applied to context operators. There are two cases:

$$[P_{\text{op}}; B_{\text{op}_0}; L_{\text{op}_0}]\ [\epsilon; B_{\text{op}_1}; L_{\text{op}_1}] = \left[P_{\text{op}}; (B_{\text{op}_0}\ B_{\text{op}_1}); (L_{\text{op}_0}\ L_{\text{op}_1})\right] \quad (46)$$

$$[P_{\text{op}}; B_{\text{op}_0}; L_{\text{op}_0}]\ [--; B_{\text{op}_1}; L_{\text{op}_1}] = \left[--; (B_{\text{op}_0}\ B_{\text{op}_1}); \left((L_{\text{op}_0}\backslash(\delta L_{\text{op}_0} - \delta L_{\text{op}_1}))\ L_{\text{op}_1}\right)\right] \quad (47)$$

Now that we have given the formal syntax and semantics of contexts, version domains, and context operations, we can move on to typesetting.

The Running Context in Ω

As is standard, the abstract syntax is simpler than the concrete syntax, which offers richer possibilities to facilitate entry. Here is the concrete syntax for contexts:

C	::=	<>	Empty context
	|	~~	Minimum context
	|	^^	Maximum context
	|	<*val*>	Base value
	|	<*L*>	Subversions
	|	<*val*+*L*>	Base & subversions
val	::=	~	Minimum value
	|	^	Maximum value
	|	*string*	Normal value
L	::=	$dim{:}C\ [+\ dim{:}C]^*$	
dim	::=	*string*	

As for the context operation, here is the syntax:

C_{op}	::=	C	Replace the context
	|	[]	No change
	|	[val_{op}]	Change base
	|	[L_{op}]	Change subversions
	|	[val_{op}+L_{op}]	Change base & subs
val_{op}	::=	-	Clear base
	|	*val*	New value
	|	--	Clear subversions
	|	*val*+--	New base, clear subs
	|	---	Clear base & subs
L_{op}	::=	$dim{:}C_{\text{op}}\ [+\ dim{:}C_{\text{op}}]^*$	

In Ω, the current context is given by:

```
\contextshow{}
```

If D is a compound dimension, then the subversion at dimension D is given by:

```
\contextshow{D}
```

while the base value at dimension D is given by:

```
\contextbase{D}
```

This context is initialized at the beginning of an Ω run with the values of environment variables and command-line parameters. Once it is set, it can be changed as follows:

```
\contextset{C_op}
```

Adapting to the Context

During execution, there are three mechanisms for Ω to modify its behavior with respect to the current context: (1) *versioned execution flow*, (2) *versioned macros*, and (3) *versioned ΩTPs*.

Execution Flow. The new `\contextchoice` primitive is used to change the execution flow:

```
\contextchoice{{C_op_1}=>{exp_1},
               ...
               {C_op_n}=>{exp_n}
              }
```

Depending on the current context C, one of the expressions exp_i will be selected and expanded. The one chosen will correspond to the *best-fit* context among $\{C\ C_{\mathrm{op}1},\ \ldots,\ C\ C_{\mathrm{op}n}\}$ (see the discussion above of Context and Version Domains).

Macros. The Ω macro expansion process has been extended so that any control sequence can have multiple, *simultaneous* versions, at the same scoping level. Whenever *controlsequence* is expanded, the *most relevant*, i.e. the *best-fit*, definition, with respect to the current context, is expanded.

A version of a control sequence is defined as follows:

\vdef{C_{op}}*controlsequence args*{*definition*}

If the current context is C, then this definition defines the $C\ C_{\mathrm{op}}$ version of *controlsequence*. The scoping of definitions is the same as for TEX.

This approach is upwardly compatible with the TEX macro expansion process. The standard TEX definition:

\def*controlsequence args*{*definition*}

is simply equivalent to

\vdef{<>}*controlsequence args*{*definition*}

i.e., it defines the empty version of a control sequence.

As stated above, during expansion the best-fit definition of *controlsequence*, with respect to the current context, will be expanded whenever it is encountered. It is also possible to expand a particular version of a control sequence, by using:

\vexp{C_{op}}*controlsequence*

ΩTPs and ΩTP-lists. Beyond the ability to manipulate larger data structures than does TEX, Ω allows the user to apply a series of filters to the input, each reading from standard input and writing to standard output. Each of the filters is called an ΩTP (Ω Translation Process), and a series of filters is called an ΩTP-list.

There are two kinds of ΩTPs: internal and external. Internal ΩTPs are finite state machines written in an Ω-specific language, and that are compiled before being interpreted by the Ω engine. External ΩTPs are standalone programs, reading from standard input and writing to standard output, like Unix filters.

Internal and external ΩTPs handle context differently. For external ΩTPs, the context information can be passed on through an additional parameter to the system call invoking the external ΩTP:

program -context=*context*

Internal ΩTPs have been modified so that every instruction can be preceded by a context tag. Using the simplest syntax, this becomes:

<<*context*>> *pattern* => *expression*

When an internal ΩTP is being interpreted, an instruction is only examined if its context tag (defaulting to the empty context) is less than the current running context.

When ΩTPs and ΩTP-lists are being declared in Ω, the `\contextchoice` operator can be used to build versioned ΩTP-lists. These will be particularly useful for multilingual typesetting. See [3] for more details.

The Internals of the Typesetting Engine

The versioned macros and ΩTPs presented in the previous sections clearly facilitate the development of software that is more flexible, in the sense that if new parameters are added to a system, new code only needs to be written for those parts affected directly by the new parameters.

But it is still not clear how these mechanisms will help solve this problem of having multiple input and output formats for use with the same typesetting system, in particular, inside Ω.

We outline the solution here, since at the time of writing, we have not yet finalized the syntax.

Essentially, all of Ω's internal data structures will be made directly accessible to the user. These include at least:

- streams of characters;
- streams of glyphs;
- math lists;
- input to each ΩTP application;
- output from each ΩTP application;
- paragraphs;
- tables;
- pages;
- other kinds of boxes.

For each of these data structures will be specified a canonical serialization and deserialization. For each of the algorithms that can be applied to these data structures, a means for applying the algorithms from the user level will be defined as well.

As a result, it will be possible to completely manipulate what we might call the "canonical input" and the "canonical output" of the typesetter. Then inputting from different formats and outputting to different formats becomes much simpler. For input from a specific input format, an ΩTP-list must be defined to translate that input format into the "canonical input". For output to a specific output format, an ΩTP-list must be defined to translate from the "'canonical output" to the output format. These ΩTP-lists may well be parameterized by the current context to achieve specific results.

This approach is suitable for generic content that is found in all input and output files, but what about elements that are relevant for only specific kinds of input or output? For these elements, it is the versioned macros that will come into play. Some macros will have versions defined to deal with these elements only when the right conditions occur in the context.

Conclusions

Given the successful experimental work in generating MathML and XML directly from the Ω engine, we consider that the approach presented in this paper is both elegant and doable. In fact, we consider that this approach makes it possible to integrate into a single framework most of the existing extensions to TEX, and its original view that a document should be transformed into a DVI file that is in turn converted to (for example) PostScript.

However, the implications go well beyond just integrating existing work, even if that goal is both laudable and desirable. New possibilities will arise, since the user will no longer be forced to use the TEX document model, in which a document is just a series of pages. Ω then becomes usable for typesetting small pieces of text at a time, say to generate EPS files on-demand for the uses of other applications, such as online multilingual mapping tools or air traffic control systems.

For a piece of software to survive from one generation to the next, it must be able to adapt to the arising needs of coming times, and continually provide new possibilities. We believe that the approach presented here will ensure the long-term viability of Ω, hence of the TEX community.

References

[1] Donald Knuth. *Computers and Typesetting*. 5 volumes, Addison-Wesley, 1986.

[2] Omega Typesetting and Document Processing System, `http://omega.enstb.org`

[3] John Plaice, Yannis Haralambous and Chris Rowley. A multidimensional approach to typesetting. *TUGboat* 24(1), 2003, Proceedings of the TUG Annual Meeting, pp. 105–114.

[4] John Plaice and Joey Paquet. Introduction to intensional programming. In *Intensional Programming I*, World-Scientific, Singapore, 1996.

[5] John Plaice, Paul Swoboda and Ammar Alammar. Building intensional communities using shared contexts. In *Distributed Communities on the Web*, *LNCS* 1830:55–64, Springer-Verlag, 2000.

[6] John Plaice and William W. Wadge. A new approach to version control. *IEEE-TSE* 19(3):268–276, 1993.

[7] Paul Swoboda. *A Formalization and Implementation of Distributed Intensional Programming*. PhD thesis, University of New South Wales, Sydney, Australia, 2003.

[8] Extensible Markup Language (XML), `http://www.w3c.org/XML`

Enhanced Font Features for Future Multilingual Digital Typography with Sound-Script-Language Attribute Integration

B. V. Venkata Krishna Sastry
Hindu University of America
Orlando, Florida, USA
sastry_bvk@hindu-university.edu

Abstract

Digital technology has the potential to deliver the 'integrated sound value of the script symbol of a selected language' in one media platform, in an interactive, multimodal way. This is the feature of the 'primary digital document', that needs to be delivered by the digital typography, in the multi-lingual, multi-modal pervasive computing environment of the users of the digital/hybrid library. In its current state, typography is dependent on the strengths and trends of the programming languages, technologies and related standards. These are designed with the 'font/character based paradigm' at their foundation. The study points out as to how the limitations of the current programming languages and the related standards affect the future of 'digital typography' in delivering its stated goal in the digital/hybrid library scenario. The importance of shifting from the current level to the proposed 'integrated sound-script-language' feature of the 'font-character' in the 'digital typography for pervasive computing' is highlighted. The advantageous position of the model of Indic language Vedic Sanskrit for the above paradigm shift is indicated. The critical nature and benefits of this paradigm shift from the digital/hybrid library scenario for the assimilation of the current technology trends is demonstrated by a 'digital book model' developed by the author. The digital book deals with content related to the Vedic phonetics.

Résumé

La technologie numérique a le potentiel de délivrer le «son intégré du symbole écrit de la langue choisie» sur une même plate-forme, de manière interactive et multimodale. C'est une propriété du «document numérique primaire» que l'on souhaite transmettre par le biais de la typographie numérique, dans l'environnement informatique multilingue, multimodal et universel des utilisateurs de la bibliothèque numérique hybride. Dans l'état actuel, la typographie dépend des forces et des tendances des langages de programmation, des technologies et des standards. Ceux-ci ont été conçus en se basant sur le paradigme «fonte-caractère». Notre étude démontre à quel point les limitations des langages de programmation actuels et des standards affectent l'avenir de la «typographie numérique» et sa capacité d'atteindre son but dans le scénario de bibliothèque numérique/hybride. Nous soulignons l'importance de passer de l'état actuel à la propriété d'intégration du son, de l'écriture et de la langue du couple «fonte-caractère» dans la «typographie numérique pour informatique universelle». Nous indiquons la position avantageuse du sanskrit védique dans ce changement de paradigme. Nous démontrons la criticalité et les avantages de ce changement de paradigme par un «modèle de livre numérique», développé par l'auteur. Le contenu de ce livre numérique traite de phonétique védique.

Introduction

As computers increasingly permeate our daily lives, and influence the way we place our (creative — expressive — communicative and responsive) communications in the digital media, 'digital typography' plays a critical role in these aspects linked to data and its utility for 'creating — formatting — handling — storing — distributing — accessing — archiving'. These are the various issues related to the 'digital document'.[1] At the current level of technology and usage, the word 'digital document' covers very many dimensions and formats — from plain ASCII text to a web page with multimedia content. The future digital document[2] in the digital/hybrid library will

1. This is the vision expressed in the MIT projects, http://oxygen.lcs.mit.edu.

2. The Digital Library Federation (DLF) was founded in 1995 to establish the conditions for creating, maintaining, expanding, and preserving a distributed collection of digital materials accessible to scholars, students, and a wider public. The Federation is a leadership organization operating under the umbrella of the Council on Library and Information Resources. It is composed of participants who manage and operate digital libraries. Hybrid Libraries of the Future delivers both print and electronic services. In considering changes in academic libraries, including integration with computing services, this model seems more useful than the pure digital library concept. The HYLIFE project is about how

feature hybrid multilingual multimodal communicative data, accessed by the users in unscheduled interactive ways. The critical issues currently being considered in the scenario of digital/hybrid libraries are the following: (a) (Text — Script — language related) — Monolingual, cross lingual and multilingual. (b) (Speech related) Spoken document retrieval; spoken document corpora, music. (c) (Interface creation — storage — distribution — access) Multimodal interfaces for text and spoken document, inter-operability. (d) (OS — Software-platform-network hardware device related) Recognition algorithms, techniques for spoken audio indexing, retrieval models, web interface, metadata; algorithms and protocols — Human Computer Interfaces — Digital library interface (e) (Utilities — solutions related) Library catalogue, collations, data mining, special purpose libraries and services.[3]

Digital document

Digital documents have, related to their origin and format, a background of very many issues such as: software programs, operating systems, hardware encoding, character related standards, file formats, memory requirements, color handling, font resolution related to the display screen resolution. All these have a bearing on 'document portability-compatibility-effective usability'; and therefore on 'user satisfaction'. In the present state of the document creation in the digital media, controlling many of these are beyond the 'author-user' abilities.

Digital typography Compared to the conventional print media output, to which the earlier phase of typography was dedicated, the output in a digital environment has different challenges to address. The advancement of the technologies is in the direction of integrating the 'speech and vision' modes of human interaction with the computer for better effectiveness in a pervasive computing environment. Therefore the focus of the digital typography must cover a larger ground than the conventional typography approach, to be in tune with the trends of the technological innovations.

Literate Programming By definition, it is the "combination of documentation and source together in a fashion suited for reading by human beings". In general, literate programs combine source and documentation in a single file. Literate programming tools then parse the file to produce either readable documentation or compilable source. If we extend our understanding of the word 'reading' to cover not only visual text oriented reading, but also the speech mode of reading, this defines the directions in which TEX has to look for the future. The 'typographic font' which served the 'print media needs' and the 'display media needs' has now to stretch itself to the level of the 'Sound-Script-Language attribute integrated font' as it was in the original design of 'script logic', the mother of typography.

LATEX, as a document preparation system, also needs to focus on the scope of 'right content' along with the 'right mode of presentation, storage and distribution' in such a scenario. In order to make the 'digital document' effective, the design of the document, which was earlier in the control of the 'designers' must allow for contributions from the 'author of the document'. The media in which the author was bound to position the 'content' (= the 'data' and the thought expressive process) has undergone a paradigm shift and a total upgrade. The limitation of the 'paper-print-text media' is overcome in the 'integrated digital media'.

What is the expectation from digital typography for a digital document?

It is at this stage that the limitations preventing 'typography' from delivering the desired goal of 'digital document' as above surface. The human paradigm of communication-documentation has been that the 'Script symbol is always associated with a sound value in a given language'[4] environment. So far in the machine paradigm, the approach towards the 'language and the sound value of the script symbol' has been the isolated juxtaposition[5] of the script symbol with the 'sound value'[6] with a 'language'[7] attribute.[8]

best to deliver the mixture of print and electronic services likely to be required of higher education libraries in the foreseeable future.

3. DESIRE Project — DESIRE was a European information gateway project (1998–2000): Development of a European Service for Information on Research and Education. Among many other resources, DESIRE produced free software and some publications, including the excellent DESIRE Information Gateways Handbook; Library of Congress (U.S.) digital library resources and projects LOC Standards for Digital Library Projects; international standard, ISO 14721:2002; Metadata Standards: Dublin Core (Stuart Weibel: OCLC, USA).

4. This language environment is again a deep rooted issue with operating systems and the standards which needs to be elaborated. The language environment is in many cases defined by the OS at the time of installation.

5. SALT (Speech Application Language Tags) is the latest in this series, `http://www.w3.org`.

6. For the details look at the functioning of the TTS programs and the speech recognition process in the documentation for Microsoft's .NET Speech SDK 1.0 Beta 2.

7. Feature tags and language tags: Features provide information about how to use the glyphs in a font to render a script or language. Language system tags identify the language systems supported in an Open Type font. See information at the Microsoft Typography site.

8. The concept and criticality of the 'code page' throughout the issues related to the fonts, multilingual software and the Unicode standards is a vast and involved topic that needs to be carefully read. For details, refer to `http://www.unicode.org`.

For over forty years, the design paradigm of computation has centered on machines and not people. Our own creations in terms of machines, networks, programming languages, software and the program interfaces have required us to interact with them on their terms, inputting their languages and manipulating their keyboards or mice. The entire edifice is pinned on one design principle paradigm — the ability of the visual scripted 'character'[9] (and character sequences) to provide a communication link between the humans and the machines. With this 'character oriented' design philosophy built into the core of the operating systems, software, programming languages, instruction sets[10] and the various standards which cover these aspects, we have not been able to 'define' what we really need to the 'machines' as projected in our visions and goals. When the vision is to deliver 'natural language processing', the nature of the natural language has not been defined. What we have is a 'structured query language' (SQL). When the vision is of 'speech recognition', the vital element of 'sound' is totally missing from consideration in all the reference standards and programming languages. 'Audio' is treated as an 'external add on/markup reference', which is not the appropriate paradigm. The language choice, which should remain under user selection, presently resides in the scope of the OS.

If we find a solution for this issue, then we will able to automate many repetitive and logically describable tasks presently carried out by humans in typography. The technology has advanced to a far greater level compared to the days when the original 'character based paradigm' was built at the core of the machines and the standards. There are newer techniques of rendering the characters on the display devices. But, the very fundamental limitation of the 'character based paradigm' of the communication interface itself, has never been revisited. The applications which propose to deal with the 'speech' and other aspects of human interaction communication and interface, are still relying upon the 'character based linking paradigm' to deliver the solutions in these areas.

In pursuit of the above approach, we notice that the critical speech interface for human interaction is built by the industry as an 'extension, markup' feature over the 'character'; and not as an integrated feature on the paradigm of human communication. This is the key area that needs to be addressed in the area of digital typography to meet the deliverable goals for the digital document of the digital/hybrid library. In the envisaged digital/hybrid library environment, digital documents will come in multiple volumes and a variety of data forms through real time online resources and offline databases in multiple media, multilingual formats and modes, addressing the visual and auditory modes of reception. The digital typography environment is presently heavily preoccupied with the Roman alphabet character set, widespread on the keyboards of modern computers. The world is not just limited to the 'English/Latin/Roman alphabet' and character set. Due to the deep positioning of the Roman alphabet character set represented in the ASCII and other code standards for digital programming, the other languages of the world need to be represented through these character sets only in digital space. This is the place where the 'accentuation markings', diacritics, considered in the IPA convention come into consideration.

Why should there be an integrated voice interface in a digital document?

Users generally do not have equal language skills for understanding and appreciating a 'document' in a language. The skills related to language are — reading, writing, speaking. The outpouring from a contemporary 'search' provides informative documents, mainly in the form of 'text mode'; all of which the user may/may not be able to read, understand or need help to understand properly. In such cases the 'voiced output' speech interface becomes more effective and meaningful. Present-day typography is not in a position to deliver this solution in an integrated way with the text. Text readers have serious limitations, as do ASR's. The proposal for the enhanced font features with Sound-Script-Language attribute integration aims at providing this solution. The goal of this proposal is to get a unifying ground of 'communication-documentation' that needs to be held as an integrated 'digital hybrid document.'

Interface design for human paradigm vs. machine paradigm — integrated sound-script-language

In the present state of technology, such a hybrid data entity is seen in the 'interactive-multimedia content' created through the use of special multimedia authoring software. This needs a minimum level of hardware power. Even here, there are serious limitations for user

9. The character related standards right from the earliest standards considered in the early programming languages to the current discussion focus on the IPA and Unicode are all focused on 'visual character' — the non-agglutinating, non-phonetic, Roman alphabet character set and does not deal with the 'phonetic value' of the script symbol. Check the ISO 639 standards and related RFC's for more information in this regard, including ISO/IEC 10646. The reference point these standards take is primarily the typewriter pattern of characters or a calligraphy based approach. They have never entered into the realm of 'script logic' itself, as to what the symbol stands for in the 'language'.

10. In the final level of programming where a voice interface is needed, it is a set of code based on the 'characters', which calls on a 'library/data base of the speech'. But it is not capable of integrally linking itself to a sound value. The final code written as XML/SGML for SALT Version 1.0 may be referred to.

interactivity with relation to the possible permissible modes of interaction. The focus is still on keyboard input and the mouse clicks.

In current technology trends, the effort has been to create human interfaces with the machine in various modes. The objective is to have an integrated speech and vision interface. The goal is to have a technology which will enable the user to communicate with the machine in multi-modal, interchangeable, intelligent modes of interaction, ideally in much the same way that humans communicate with other humans. In order to realize this goal, there is a need to identify some common ground between the 'human and the machine styles of communication handling'.

At the present state of the art and philosophy of the computer science and technology, such a 'common ground' between the 'human speech/natural language communication process' and the 'machine's programming language oriented task/output delivery' is identified and limited to the level of the 'script character'(-font) of 'language' (as a code page), in the digital media. This is brought into the issues of the standards. The integrated relation of the 'script character' in human linguistic audio mode communication has not been represented properly in the standards, programming languages and industry practices. What is happening in the 'core system level processing' may be termed as 'table matching (by filtering and elimination mainly) and super fast routing/couriering ending with mono-modal unidirectional deliverance — display on predetermined paths and frames', giving an illusion of compatibility with human communication due to the speed of the system and the complexity of the program. Thus, with only a partial non-integrated ground being established and standardized between the 'man and the machine' in the area of 'communicative interaction', the stated objectives for digital document creation continue to remain a separate area of study.

Vedic Sanskrit as a model of human language providing the integrated sound-script-language concept

It is acknowledged that the Indic language identified as 'Vedic Sanskrit' is the most ancient human language which continues to have a living tradition in India. This uses a non-Roman script, which was in its almost final shape a thousand years ago. The script called 'devanagari Sanskrit' is used by many Indian languages. The devanagari script is the script adopted for the Hindi language, which is the national language of India. The philosophy and grammar of Vedic Sanskrit provides the guidelines in the formation of the concept of 'integrated sound-script-language'. The grammar of Sanskrit language authored by Panini (circa 500 B.C.E) and the related living tradition is available even to this day. The linguistic design and philosophy of this language is developed purely in the 'human linguistic communication mode, especially in the aspect of speech'. There is a well laid down rule base, which can be adopted in to the 'speech mode of communication of any language of the world, irrespective of the script they intend to use'. The script logic of this language, the agglutinating structure of the script which enables the representation of combined sounds in terms of basic phonetic units, the accent marking conventions to mark the prosody, and the phonetic alphabetical order on the human voice mechanism are some of the features of this language which can be used for 'fonts' developing the integrated sound-script-language paradigm as indicated.

The benefits of the enhanced font features

This 'integration' is seen as the key interface that is needed for the creation of the digital document of the future and needs to be demanded of digital typography. Namely, the integration of the 'script' symbol character with a 'pronounced as' feature. Thus, the dictionary functionality gets truly integrated in the multilingual scenario through this paradigm shift.

Scope for newer applications through this paradigm

There are many users (literate as well as semi-literate and non-literate) who are familiar with more than one language in their spoken aspect; but they can not write/read the script. Such users can 'listen' to a 'multilingual document' (a reality at the global level) because the 'sound value' of the script and the language would be available. And this makes the document more intelligible.

The content creator can always bring in the 'multilingual' content without being limited by the 'fonts-display-phonetic value-diacritical marks' issues and focus on the primary task. Language education can become much more effective through this paradigm of typography. This paradigm can also be used for language specific dictation machines. This makes digital document creation simpler, faster and more reliable. The proofing of the text is facilitated to ensure the right reading for the right meaning. Proofing in multilingual documents can become better. The multilingual transliterations can become more practical and effective because of this 'reading of the text', especially in the spelling oriented languages of the world. The demand for this upgrade of the 'font' has to come from digital typography, as it would be the primary beneficiary of such integration.

Conclusion

In conclusion, it is seen that digital typography should push for enhanced features of 'fonts' with the 'sound and

the 'language' attributes in an integrated way to meet the goal of 'literate programming' and the demands of digital hybrid libraries. Failure to reach this target will just leave every document another 'graphic' on a 'media' which is no better than an unintelligible picture or a scribbling.

The theme of 'back to typography' actually becomes meaningful when the 'sound value of the script of the language' is brought back 'live' in the digital media. Digital typography should not be happy just transferring the 'visual image' created by the human author in a media of his/her time and context to a 'cyber document'; it also has to capture, preserve and deliver all that really makes that 'document' significant and meaningful. This will go a long way toward preserving the typographical heritage in the digital era, for not only the print media book production process, but also digital books. By making this demand for the integrated feature of sound and language value in the 'character-font', there will be a new impetus for literate programming in the digital typography of the production of digital/hybrid documents.

The following product is a scaled down version of a digital hybrid document. The content is an advanced topic relating to the phonetics of the Vedic Sanskrit language and the Vedic recitation in a special mode. This is sacred and historically significant content. This rule base of the Vedic Sanskrit phonetics was found documented in a paper manuscript in the private collections of a Sanskrit scholar, Professor K.T. Pandurangi, Bangalore, India.

Vision of a digital book

In order to port the totality of the contents of this paper media handwritten manuscript document to a 'digital hybrid document', a search was made for the appropriate software and tools. Multimedia authoring technology and CD-ROM media was the best we could find to meet the needs. Based on the level of expertise and the resources that were available to us, a scaled down version of the product has been prepared. Apart from the focus of this product as a digital/hybrid document preserving the heritage document, for the purpose of this presentation we also focus on the need to work toward the enhancement of 'font features' for future multilingual digital typography with sound-script-language attribute integration.

The benefits of the 'integration of the sound-script-language attributes' on one digital platform for the effective utilization of the 'digital document' is the message intended to be conveyed through this presentation.

This product also addresses some other issues, including survival of heritage documents in the digital era. It is a direction in which digital typography can help.

Nature of the document

The heritage document on hand which was to be brought to the digital library environment was a paper manuscript, 36 folios of approximate size 4x6 inches, about one hundred fifty years old, containing unpublished information. It needs to be preserved for future studies. This manuscript was in the private collection of a religious professor. There was only one copy of this document, which he was not willing to spare for public use. Upon study of the manuscript at the holder's location, it was noticed that the text, written in devanagari script of 19th century (the writing style in the Maharashtra-Karnataka border region in India) deals with a critical issue related to the Vedic phonetics. As per the information in the manuscript, the original work was composed in the period prior to 16th century A.C.E.

The contents of the book are useful for the advanced study of Vedic phonetics. The small booklet contains about three hundred illustrations drawn, in a cryptic way, from the entire Rig Vedic lore. To explain this, the text script reading alone is not sufficient. The accented intonation of the passages indicated by the first two or three words in the specific mode needs to be supplemented by a proficient scholar trained in the traditional mode of chanting the Vedas, Vedic phonetics, and grammar. The book does not have commercial publication value because it is a very advanced level reference, which would be utilized only by researchers. The present manuscript is a replicated copy, which has seen careful correction by the scribe-author. The corrections carried out by the author can be noticed only on careful observation of the manuscript.

The challenge

The challenge is to port this heritage manuscript to a digital hybrid library document, using the single copy available, which would not be spared for circulation. Further, the challenge is to get a good image of the manuscript and make it available for online and offline researchers, including a facility to take to print media when the time comes. The issues involved were broadly identified in the work area as follows. The technical and academic solutions that were practical within our resources at the time of production were put into use as indicated below.

Document imaging

The available middle range level scanner which could capture the image of the basic document was used to get a reasonably good 'electronic image of the text'. Judgment was exercised on the resolution with reference to the output file size, the color balance, and like factors. The first level image was obtained. The image did not have the requisite clarity. The primary image was retained as

such. Then this image was processed through image enhancement software to enhance the picture quality, improve the contrast, reduce the color noise, etc., all with one intention — that the 'text' can be read with better clarity on the screen. Thus, the first level of 'content extraction' at its primary level — in the form of an image — was completed.

In the product, which is a scaled down version of the final book, you will see the original image and also the digitally enhanced image. The digitally cleansed image has much improved clarity for reading, compared to the original image. This was the first phase of the transfer of the manuscript paper media to 'digital media'. At this stage, we had achieved the ability to port and deliver/carry the original image (unclear and difficult) and also the digitally enhanced image (clear for reading) together to any place throughout the world where there were scholars able to help with the next level of content evaluation.

Content management in non-computer media and lack of computer awareness/access environment

The subject dealt with by the manuscript is a very specialized topic. Many of the subject experts in this area are not computer savvy, nor have computer access. Therefore we had to discover a way to have the content in a deliverable print media. The ordinary printing through the printers were tried out; but the outputs even at their best resolutions were not of acceptable quality. Therefore, the technology group recommended the use of digital photo imaging output technology. We produced the Kodak photo print of the original and the digitally enhanced image together on a photo grade quality paper. And this was found to be extremely satisfactory in terms of quality and acceptance by the scholars for further work.

The success at this stage was the decision to switch to digital photo production technology on 'photographic quality paper', giving up on conventional color printer output. Accordingly, the appropriate digital image editing software and formats were re-identified and the output was obtained. This is one of the ways that could be adopted for 'hardcopy delivery' in the digital hybrid library environment in view of the output quality. The digital imaging needs to be appropriate for the working result.

Getting to the phonetic value of the script and the text

Once we were able to take an acceptable quality image of the original to the scholars for help, they were able to read the 'script' in which the text was written. It was noticed that the conventions of writing the original text with the devanagari script had its own peculiarities of that historic period, as well as regional uniqueness; and also the 'scribe's unique stamp in terms of the calligraphic style'. The plain character based reading was not sufficient to properly understand the text. The reading aloud of the text and comparison with the 'voice tradition' became a necessity. Fortunately, we had the scholarly resources at hand and with their help, the proper reading of the text was completed. Thus the correlation of the voice tradition with the scripted text was accomplished, which lends more firmness, authenticity, and useful content value. The voice output of the text was also brought in to the digital media and placed alongside the text as tagged data, through our software programming.

Actual technical content of the text

The scholars in the content evaluation were able to furnish useful information. A part of this information is placed on the CD. More of this can be integrated in the desired way. This area has not posed much of a problem.

Content evaluation of the text

The difficulties in getting the knowledgeable scholars in the most advanced area of the Vedic knowledge described in this text with the requisite voice training, voice clarity with a technical teaching expertise and demonstration capability was difficult. However, we were able to identify them, and get the additional information. Thus, a sample of the final voice was placed on the CD.

Transcription of the traditional handwritten text with the help of modern digital fonts

Now began the next phase of the work — presenting the heritage document in the fonts of the 'modern digital era' using 'digital typography'. Based on the combination of voice and text that was made available to the text publishing agency, the new text was received as an output based on the available software.

It is at this point that the real hidden problems with the fonts, encoding, software compatibility, proofing, media and other issues cropped up. The absence of the 'integration of the phoneme value' on the 'script symbol' in the digital typography environment, in sharp contrast to the human paradigm of 'scripting', was a glaring handicap at this point.

Issues related to the multilingual transliteration

The problems of transliteration highlighted the limitation of digital typography multilingual software and the standards related to it in meeting the needs of the content being handled, to provide truly acceptable output. Other issues were similar — divergence of the transliteration standards followed by the software, portability

and compatibility, the relative problems that are yet to be resolved regarding IPA and Unicode compatibility, the deep rooted problems in language related matters dealt with at the code page level of the OS, and version and upgrade problems of the software. The scholars were not totally satisfied with the software related transliteration outputs. The publishing software which handled many of the transliteration related fonts and symbols were not able to give us related softcopy compatible for our program integration in our multimedia presentation. We had to settle for the final 'image output of the transliterated text', as our purpose was to deliver the content. This necessity to go by the 'image mode' output of the text does limit on 'text-search' by character approach. Voice based search is still a far dream.

Voice part of the content

We recorded only part of the text reading to manage within our resources. The work of really having the illustrated voice explanation with the technicalities explained in the work is yet an unfinished task, presently beyond our resources.

Database indexing of the folios

The indexing of the folios has been completed. Only the first two and last two folio links are activated.

Providing online digital facilities for notes and online access

CD-ROM media was chosen as the output, so that the product can be used offline. The built-in facility within the product provides for calling an editor for making the user notes. The product is made with a Windows-compatible feature. Thus when this CD-ROM is open in one window, the user has the option of calling up any word processor (regular or multilingual) installed on the machine and make their own personal notes. The Windows-compatible feature also helps in having another window opened up in which the second digital book provided along with this primary work can also be accessed. CD navigation is structured through the use of original multimedia software.

Enhancing the utility of the product

Supplemental information in the form of links to dictionaries and reference books has been planned for the CD navigation structure. We considered providing the reference book of 'Rig-veda' on the same CD with the same facilities and interfaces, as a model so that the study experience would be integrated and complete. On the CD you would notice the availability of the work 'Rig-veda', which can be simultaneously accessed by opening the CD in another window.

Technology utilization for content illustration and appreciation

It is planned to achieve this goal by providing the appropriate video and the graphics, animations, and pictures, as considered necessary by the scholars. A planned provision is indicated in the present version of the product. This will enhancem the utility of the digital document.

Technology issues

In the absence of the single platform where we can completely fulfill our needs at reasonable cost levels, the technical team did a tremendous amount of experimentation and the current product structure is the result. It is during this study that the need for the enhanced font features brought out in this paper became apparent for the digital typography environment. Live voice is needed (as traditional), in contrast to the concept of 'text readers' (which heavily depend on the character oriented programming and just give a mechanical voice output) which are not sufficient at this level of content. The deliverable product was made in a form suitable for online and offline reference so as to fall under the category of 'digital/hybrid document'. The interface has been designed to meet user requirements for studying in script mode and in voice mode; the interface provides for access to the requisite references in all the modes including the web. The facility to use multilingual, multimodal methods, depending upon the user hardware facilities, is open for integration.

What does this product mean for digital typography?

When there are already a number of programs for publishing, imaging, text readers, OCR products, TTS and ASR solutions, multimedia authoring, web based education, and so on, why should this issue be so critically highlighted?

It is true that all these technologies are available in isolation, but integration of these using the paradigm of the human conventions of communication and learning is the challenge. The technologies mentioned above fail to achieve a smooth and satisfactory integration and deliver an acceptable cost effective output from the point of view of an author-scholar and user. On close scrutiny of the issue, the remedial measure is best initiated at the level of the 'font' character concept used in current digital technology solutions. This paper highlights the issue through an application of the current technologies to deliver a model product for the future digital hybrid library as a digital document with significant heritage content. The present product is a result of many technologies, resulting from the interfaces that the user needs in digital documents and digital/hybrid libraries. For the production of such end user friendly multilingual-multimodal

digital books, the digital typography has to shift from its present paradigm of 'character based' programming techniques to the proposed 'enhanced font features with sound-script-language attribute integration'.

Acknowledgments

I acknowledge gratefully the pains taking work done by Sri Parthasarathy, an expert in the multimedia related work and the overall integration of the various pieces of information into one single navigable product. The opening screen, the animations, and the closing animations are an indication of his creative contributions to this product.

References

[1] Pike, Kenneth L., *Talk, Thought, and Thing: The Emic Road Toward Conscious Knowledge*, The University of Texas at Arlington. Summer Institute of Linguistics.

[2] Unicode Standard Annex #28 Unicode 3.2, Mar 2002. Standards and Specifications. http://www.unicode.org/reports/tr28/

[3] ISO/IEC 9573-13: International Organization for Standardization. Information technology, SGML support facilities, Techniques for using SGML. Part 13: Public entity sets for mathematics and science. [Geneva], 1991. (ISO/IEC TR 9573-13:1991).

[4] ISO/IEC 9995-7: Information technology, Keyboard layouts for text and office systems. Part 7: Symbols used to represent functions. [Geneva], 1994. (ISO/IEC 9995-7:1994).

[5] ISO/IEC 14651: International Organization for Standardization. Information technology, International string ordering and comparison, Method for comparing character strings and description of the common template tailorable ordering. [Geneva], 2001. (ISO/IEC 14651:2001).

[6] Acero, A., *Acoustical and Environmental Robustness in Automatic Speech Recognition*, Kluwer Academic Publishers, 1993, Boston, MA.

[7] Anderson, Donald M., *Calligraphy: The art of written forms*, Holt, Rinehart, and Winston, 1969, New York. (This is more a book about the history of written communication than a book on calligraphy. Includes Latin and non-Latin scripts.)

[8] The Hewlett-Packard Book of Characters. Hewlett-Packard Company, 1990, Boise, ID. http://www.fonts.com. (The description of characters for Hewlett-Packard printers and code pages.)

[9] The Art & Technology of Typography. Compugraphic Corporation, 1988, Wilmington, MA. (A pamphlet released by Compugraphic for its typesetting customers. It has common descriptions and examples of typographic terms, characters and uses.)

[10] Guide to Macintosh Software Localization. Apple Computer, Inc. Addison-Wesley Publishing Company, 1992, Reading, MA. ISBN 0-201-60856-1 (pbk). (Part of the *Inside Macintosh* series of technical specifications. Specifically relates to user interfaces and text worldwide on Macintosh systems. Shows examples of user interface guidelines, keyboard layouts, localised character sets, language descriptions, country and writing systems.)

[11] HTML 4.0. World Wide Web Consortium. Massachusetts Institute of Technology, Institute National de Recherche en Informatique et en Automatique, Keio University, 1997.

Contemporary Hungarian Types and Designers*

Gyöngyi Bujdosó
Department of Computer Graphics and Library and Information Science
Institute of Informatics
University of Debrecen
H-4010 Debrecen, P.O. Box 12
Hungary
ludens@math.klte.hu

Abstract

Several people have asked for information on how to use Janson fonts with TEX. This wonderful antique font was designed by a Hungarian type designer, M. Tótfalusi Kis Miklós (Nicholas Kis) in the 17th century. His name is well-known, but few people know the other, wonderful types such as Pannon Antikva, Tyrnavia, etc., created by talented Hungarian designers.

The aim of this paper is to give a short overview of some new types by Hungarian type designers, and portray the work and the achievements of cooperation between TEX people and typographers.

Résumé

Souvent les gens se demandent comment utiliser la fonte Janson sous TEX. Cette merveilleuse fonte ancienne a été dessinée par un graveur de caractères hongrois, M. Tótfalusi Kis Miklós (Nicholas Kis), au XVII^e siècle. Son nom est bien connu, mais peu de gens connaissent ses autres caractères, comme Pannon Antikva, Tyrnavia, ou alors d'autres caractères dessinés par des dessinateurs hongrois talentueux.

Le but de cet article est de donner un bref aperçu de certains de ces caractères, et de préfigurer les formidables fruits d'une collaboration possible entre la communauté TEX hongroise et les gens du métier.

Motivations

Several people have asked for information on how to use Janson fonts with TEX. This wonderful antique font was designed by a Hungarian type designer, M. Tótfalusi Kis Miklós. His name and his Janson font are well-known among TEX people. During a BachoTEX conference the author was asked about the Pannon Antikva designed by Edit Zigány, as well as about other typefaces made by Hungarian designers and their states of digitization. After writing several e-mails, surfing on the web, and phoning to publishing and printing houses it seemed that there were no type designers in Hungary! In fact, we have several talented type designers, but it was quite time consuming to find them because of the lack of contact between TEX people and designers.

The aim of this brief overview is to collect and present some nice examples, and to mention the names of some talented Hungarian type designers, with font samples where possible. At the end of the paper the author reports some new results of the localization of TEX in Hungary.

Some wonderful typefaces

When we are speaking about Hungarian typefaces we cannot leave out the forerunners, who have had telling effects on contemporary type design.

The famous typeface known as *Janson* is the work of Nicholas Kis (1650–1702). The Hungarian name of Nicholas Kis is M. Tótfalusi (or Misztótfalusi) Kis Miklós. He made this typeface (among others) during his stay in Amsterdam (1680–88) where he had been sent to publish a new Hungarian edition of the Bible. The title page of this second, modernized, corrected Hungarian translation of the Bible can be seen in Figure 1.

After some months of study he began to cut complex fonts and fix defective matrices. The resulting Janson is quite a famous typeface (Figure 2). The name Janson is actually a misnomer; this typeface was long thought to have been made by the Dutch designer Anton Janson, who was a practicing type founder in Leipzig during the years 1668–87. This font from the Baroque period is a good general-purpose font. TEXers can use the Janson Text PostScript fonts from Adobe (Figure 2), Corel and

*. This research was supported by the Hungarian National Foundation for Scientific Research (OTKA), Grant No. T-032361.

Fig. 1: The Hungarian translation of the Bible. Edited, modernized and corrected by Nicholas Kis. The picture is from [4].

ABCDEFGHIJKL *ABCDEFGHIJKL*
MNOPQRSTUVW *MNOPQRSTUVW*
XYZabcdefghijklmn *XYZabcdefghijklmn*
opqrstuvwxyz&0123 *opqrstuvwxyz&0123*
456789ÆÁÂÄÀÅÇÉ *456789ÆÁÂÄÀÅÇÉ*

Fig. 2: Janson Text — Adobe [1]

Monotype. There are additional typefaces directly influenced by the types of Nicolas Kis, e.g. Stempel Janson, Mergenthaler Linotype Janson and Bitstream Kis [3].

Concerning the typography of the XXth century we must first mention the typeface *Margaret* made by Zoltán Nagy (1920–1998) that won a prize (3rd place) at an ITC competition in New York. This typeface is used on the pages of Hungarian passports [5]. Oszkár Boskovitz has been working on the digitization of the types of Z. Nagy, implementing and modifying them with the approval of the designer's heirs.

László Bujáki, a student of Zoltán Nagy, began to localize the English fonts for the Eastern European languages by changing the accents to accented letters under the guidance of Nagy. He used his own type *Cheri* to typeset Nagy's book entitled "A betűtervezés technikája" (Technics of type design), published in twelve copies on the occasion of his professor's seventieth birthday. Unfortunately these fonts disappeared along with photomechanical typesetting. However, one of his fonts, called *Balaton* (Figure 3), was digitized by one of his students ten years after its design. This wonderful Doric type is excellent for placards, posters and logos. We are looking forward to his new typeface called *Bulaton* that is expected to be ready in three years.

balaton 99

abc

Fig. 3: Balaton by László Bujáki

The typeface *Pannon* (Figure 4) designed by Edit Zigány in the 70's is well-known among typographers. Her aesthetic, harmonic and smoothly engraved bodytype can be used in professional publishing. It won a prize (2nd place) at the Leipzig competition for typefaces. Oszkár Boskovitz has been working on its digitization — implementing and adapting it with the designer's approval. Hopefully it will come out soon.

The roman typeface Pannon has been designed by Edit Zigány, who, with a previous variation of this alphabet, won a prize at the Leipzig competition for typefaces.

The designer of Pannon had been inspired by the font used in the 1470s by the Venetian printer Nicolas Jenson. Relationship may be discovered primarily in the gradations of strokes, in the character and the full effect of the text. Strict attachment to the original typeface was out of question as it had to be adjusted to the unit system of monotype setting.

According to renaissance traditions, the Pannon has preserved its relations to calligraphic letters. This, on the one hand ensures natural shapes and forms, i.e. good readability to this roman typeface and, on the other hand helped to solve the problem of necessary formal alterations.

Fig. 4: Pannon by Edit Zigány

We are also looking forward to a book by Oszkár Boskovitz that will summarize the Hungarian typefaces made in the 70's and 80's.

György Szőnyei is a type designer and graphic artist who has won several prizes at major competitions. His typefaces are like graphical compositions. We have to mention here the Labirinth, the Möbius kanji font, and the types Archian and Archian Wilmos that are like games in the fields of graphics and geometry.

These days Gábor Kóthay and Amondó Szegi, the talented type designers of the Fontana Type Foundry [7], the Fontmunkások and the Job Art Studio [6], are probably the most productive and successful type designers in Hungary. They designed and realized a number of wonderful typefaces. The aim of making the beautiful Tyrnavia (Figure 5) was the reconstruction of the Kis type designed in Cluj (Kolozsvár). They have made several types for different purposes: body text, titles, posters, placards, decorations, webdings, etc., in different styles: serious, playful, archaistic, modern, (pen, felt-tip, brush, pencil) script, artistic, facsimile-like, and so on. Just a few of their wonderful typefaces can be presented here (Figures 5–12, following page). These samples were made by the author from the sample pages sent by the designers, for the purpose of presenting them at the conference.

The names of some of their types without samples:

- Gábor Kóthay —
  - text types: Alphabet, Archetype, Minerva, Moda, Tisza, Spirit, Zephyr; scripts: Anglia, Bacchus, Fizz, Zanzibar;
  - some peculiar fonts: Arcade, Birdland, Depeche, Destijl, Faximile Flyer, Loop, Plexo, Totem, Versus;
  - special purpose fonts, webdings: Betabet, Bubble, Cats, Hungaria, Disasters, Subway, Surfing.
- Amondó Szegi — Gehenna, Mantra, Telegdi.

Hopefully we will be able to see and use more and more such excellent fonts in the future.

Report on the localization of TEX

With Gábor Kóthay we began to speak about free fonts for TEX. The starting point of the discussion was that the Fontana Type Foundry has some wonderful free fonts [7]. Also, Gábor Kóthay and Amondó Szegi have talked with Sun concerning some fonts for the Hungarian edition of OpenOffice. Gábor Kóthay is not averse to making some fonts free for TEX, and we will speak about it in the near future.

The first document style designed by Hungarian typographers is under construction. Hopefully it will be available from the web pages of MAPTEX later this year.

Finally, we have received some ideas for solving the problems (see [2]) of controlling the length of the last line of paragraphs and the hyphenation of words containing hyphens.

Acknowledgements. The author is indebted to the designers mentioned in the article as well as to Péter Maczó for their valuable suggestions and help.

References

[1] Janson Text, Adobe Type Library, `http://www.adobe.com/type/browser/P/P_055.jhtml`

[2] Gyöngyi Bujdosó and Ferenc Wettl, "On the localization of TEX in Hungary", TUG 2002 Conference proceedings, *TUGboat* 23(1), 2002, 21–26.

[3] Nicholas Fabian, *The Hungarian Type designer Nicholas Kis*, `http://www.myfonts.com/person/kis/miklos/totfalusi/`

[4] *In memoriam Tótfalusi Kis Miklós*, `http://www.sk-szeged.hu/kiallitas/totfalusi/nyito.html`

[5] Péter Maczó, "Tótfalusi Kis Miklós nyomdokain" (Treading in Nicholas Kis's footsteps), *Magyar Grafika*, 2002/5, 2–7.

[6] Job Art Studio, `http://www.jobart.hu/`

[7] Fontana Type Foundry, `http://www.fontanatype.hu/frabout.html`

[8] Luc Devroye, "Type in Hungary", `http://cgm.cs.mcgill.ca/~luc/hungary.html`

SPECIMEN
Characterum,
seu
Literarum,
Id eſt :
Typorum Latinum
Probatiſsimorum
Secundum suas Differentias
Exhibitorum,
qui
TYRNAVIÆ
in Collegii Soc. Jesu
Officina Fusoria Formantur,
Atque pro Ornamento
Edendorum
Librorum
Ad proprium usum proſtant.

Anno Domini
MDCCLXXIII.

Fig. 5: Tyrnavia by Gábor Kóthay

MuseFace

A typeface design is, at its most basic level, a st
You may still be using the same twentysix lett
digits, but they are interpreted in a unique way.
typeface design might be likened to an original

Fig. 6: MuseFace by Amondó Szegi

MÈLICO

MELODICO.ARMONICO.LIRICO.
MUSICALE

Fig. 7: Melico by Amondó Szegi

Glosso

AZ EVAN=
GELIOMOK
NAC, MELLYEKET VASARNA.

Fig. 8: Glosso by Amondó Szegi

Dessau

Az uj anyag, az acel tette lehetove a ketlabas s
jelentett elonyt, ahol jobb terkihasznalasra volt s
uleseken ulok labank helyre van szuksege. Ez a
elkerulni, hogy a test a femhez erjen, ezert a cso
a karfajanal faval burkoltak. A hattamlat szov

Fig. 9: Dessau by Gábor Kóthay

INCOGNITO

4 cartographic fonts in SmallCaps, Italic and
SwashCaps (Occidens & Oriens) style
+TerraIncognita Pi font (Type 1 PostScript fonts)

Őket az Ur Isten példájúl vetette minden Istenes életű emberek eleiben : méltó-is, hogy az ő példájok foganatosb légyen mi bennünk, a' lelki jókban-való elő-menetelre, hogysem némely lankadtak tunyasága a' hátra maradásra. Oh melly nagy lelki

Fig. 10: Incognito by Gábor Kóthay

304

№ 70
Brevier Antiqua Matrices per ... 90.
Typus fusus ponderat ... 10. ... 250.

№ 71
Brevier Cursiva Matrices per ... 90.
Typus fusus ponderat ... 6. ... 216.

Fig. 11: LaDanse by Gábor Kóthay

AQUAMARINE

AaBbCcDdEeFfGgHh
AaBbCcDdEeFfGgHh

Fig. 12: Aquamarine by Gábor Kóthay

Font Creation with FontForge*

George Williams
444 Alan Rd.
Santa Barbara, CA 93109, USA
gww@silcom.com
http://bibliofile.duhs.duke.edu/gww/

Abstract

FontForge is an open source program which allows the creation and modification of fonts in many standard formats. This article will start with the basic problem of converting a picture of a letter into an outline image (used in most computer fonts). Then I shall describe the automatic creation of accented characters, and how to add ligatures and kerning pairs to a font, and other advanced features. Finally I shall present a few tools for detecting common problems in font design.

Résumé

FontForge est un logiciel libre *Open Source*, permettant la création et la modification de fontes dans plusieurs formats standard. Dans cet article nous allons d'abord présenter le problème de base de la conversion d'une image d'une lettre en une image vectorielle. Ensuite nous parlerons de la création automatique de caractères accentués et de l'adjonction de ligatures et de paires de crénage à une fonte. Enfin, nous présenterons quelques outils qui permettent de détecter des problèmes fréquents du processus de création de fonte.

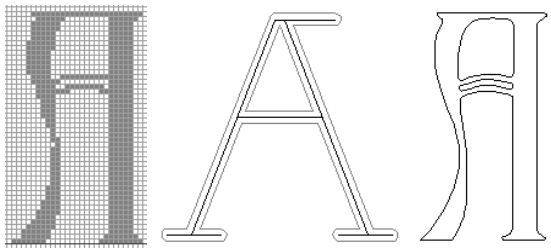
Introduction

FontForge creates fonts, allows you to edit existing fonts, and can convert from one font format to another.

What is a font? A hundred years ago a font was a collection of little pieces of metal with the same height and one design for each of the letters of the alphabet and some extra symbols like punctuation.

But the world has changed. Fonts are more abstract now; they are described by data in a computer's memory. Three main types of computer fonts are in use today.

Fig. 1: bitmap, stroked and outline fonts



The simplest font type is a *bitmap font*. Each character (actually each glyph) in the font is a tiny little picture of that character expressed on a rectangular grid of pixels. This format can provide the best quality font possible with each glyph perfectly designed, but there are two main disadvantages: there needs to be a different design for each size of the font, and these little pictures end up requiring a large amount of memory.

The other two formats avoid these problems, but often require some reduction in output quality.

A *stroked font* expresses each glyph as a set of stems, with a line drawn down the center of the stem, and then the line is drawn (stroked) with a pen of a certain width.

The final type is an *outline font*. Each glyph is expressed as a set of contours, and the computer darkens the area between the contours. This format is a compromise between the above two: it takes much less space than the bitmap format, but more space than the stroked format, and it can provide better looking glyphs than the stroked format but not as nice as the bitmap format. It makes greater demands on the computer, however, as we shall see when we discuss hints, later on.

What is a character? And a glyph? A character is an abstract concept: the letter "A" is a character, while any particular drawing of that character is a glyph. In many cases there is one glyph for each character and one character for each glyph, but not always.

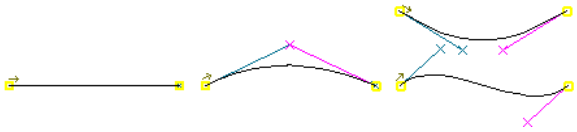
The glyph used for the Latin letter "A" may also be used for the greek letter "Alpha", while in Arabic writing most Arabic letters have at least four different glyphs (often vastly more) depending on what other letters are around them.

What is a contour? A contour is just a closed path; each

*. Original title: *Font Creation with PfaEdit*.

glyph is composed of contours. Usually this path is composed of several curved segments called splines. Each spline is defined by two end points and either 0, 1 or 2 control points which determine how the spline curves. The more control points a spline has, the more flexible it can be.

Fig. 2: splines with 0, 1 and 2 control points



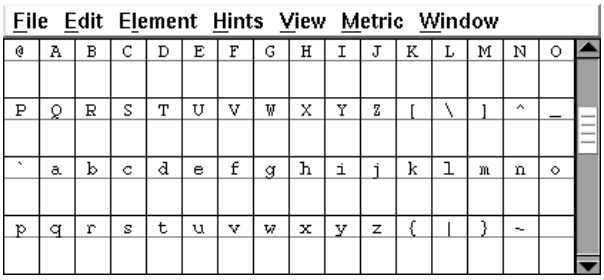
Font creation

You can create an empty font either by invoking FontForge with the `-new` argument on the command line

```
$ fontforge -new
```

or by invoking the `New` item from the `File` menu. In either case you should end up with a window like this:

Fig. 3: A newly created blank font

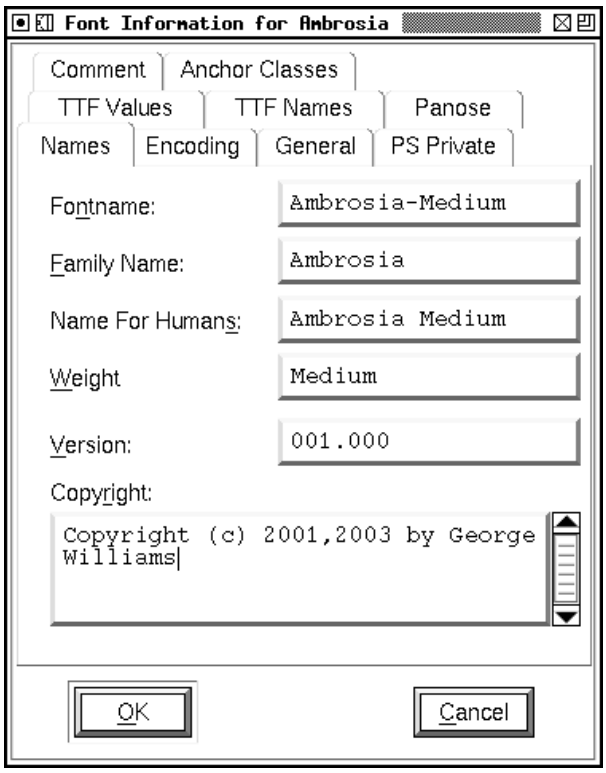


Such a font will have no useful name as yet, and will be encoded with the default encoding (usually Latin1). Use the `Element->Font Info` menu item to correct these deficiencies. This dialog has several tabbed subdialogs; the first one allows you to set the font's various names (see fig. 4):

- the family name (most fonts are part of a family of similar fonts)
- the font name, a name for PostScript, usually containing the family name and any style modifiers
- and finally a name that is meaningful to humans

If you wish to change the encoding (to TEX Base or Adobe Standard perhaps) the Encoding tab will present you with a pulldown list of known encodings. If you are making a TrueType font then you should also go to the `General` tab and select an em-size of 2048 (the default coordinate system for TrueType is a little different from that of PostScript).

Fig. 4: Font name information



Character creation

Once you have done that you are ready to start editing characters; for the sake of example, let's create a capital 'C'. Double click on the entry for "C" in the font view above (fig. 3). You should now have an empty Outline Character window (fig. 5).

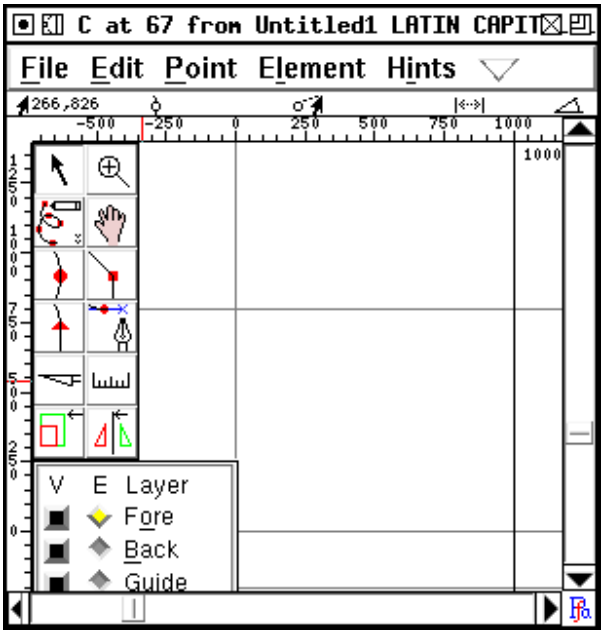
The outline character window contains two palettes snuggled up on the left side of the window. The top palette contains a set of editing tools, and the bottom palette controls which layers of the window are visible or editable.

The foreground layer contains the outline that will become part of the font. The background layer can contain images or line drawings that help you draw this particular character. The guide layer contains lines that are useful on a font-wide basis (such as a line at the x-height). To start with, all layers are empty.

This window also shows the character's internal coordinate system with the x and y axes drawn in light grey. A line representing the character's advance width is drawn in black at the right edge of the window. FontForge assigns a default advance width of one em (in PostScript that will usually be 1000 units) to a new character.

Select the `File->Import` menu command to import an image of the character you are creating, assuming

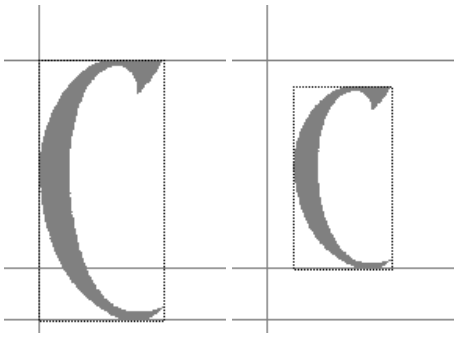
Fig. 5: An empty character



that you have one. It will be scaled so that it is as high as the em-square. In this case that's too big and we must rescale the image (fig. 6) as follows.

Make the background layer editable (by selecting the `Back` checkbox in the layers palette), move the mouse pointer to one of the edges of the image, hold down the shift key (to constrain the rescale to the same proportion in both dimensions), depress and drag the corner until the image is a reasonable size. Next move the mouse pointer onto the dark part of the image, depress the mouse and drag the image to the correct position.

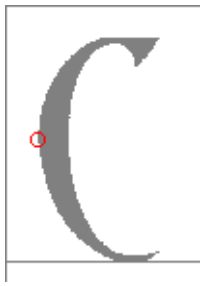
Fig. 6: Background image



If you have installed the potrace or autotrace program you can invoke `Element -> AutoTrace` to generate an outline from the image; you should probably follow this by `Element -> Add Extrema` and `Element -> Simplify`. But I suggest you refrain from autotracing, and trace the character yourself (results will be better).

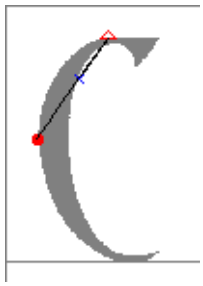
Change the active layer back to foreground (in the layers palette), and select the curve point tool from the tools palette (third from the top of the tools palette, on the left; the icon is a curve running through a circle). Then move the pointer to the edge of the image and add a point. I find that it is best to add points at places where the curve is horizontal or vertical, at corners, or where the curve changes inflection. (A change of inflection occurs in a curve like "S" where the curve changes from being open on the left to being open on the right.) If you follow these rules hinting will work better.

Fig. 7: Tracing 1: beginning



It is best to enter a curve in a clockwise fashion, so the next point should be added up at the top of the image on the flat section. Because the shape becomes flat here, a curve point is not appropriate, rather a tangent point is (the icon on the tools palette is the next one down, with a little triangle). A tangent point makes a nice transition from curves to straight lines because the curve leaves the point with the same slope the line had when it entered.

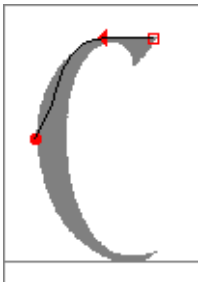
Fig. 8: Tracing 2: tangent point



At the moment this "curve" doesn't match the image at all. Don't worry about that, we'll fix it later; and anyway it will change on its own as we continue. Note that we now have a control point attached to the tangent point (the little blue x). The next point needs to go where the image changes direction abruptly. Neither a curve nor a tangent point is appropriate here, instead

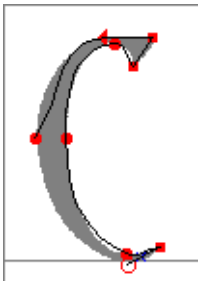
we must use a corner point (the icon on the tools palette with the little square).

Fig. 9: Tracing 3: corner point



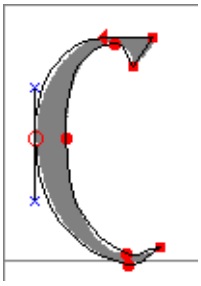
As you can see, the curve now starts to follow the image a bit more closely; we continue adding points until we are ready to close the path.

Fig. 10: Tracing 4: continuing



We close the path just by adding a new point on top of the old start point.

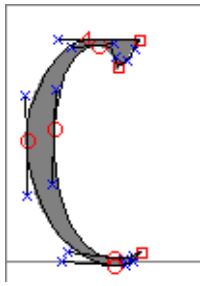
Fig. 11: Tracing 5: closing the curve



Now we want to make the curve track the image more closely; to do this, we must adjust the control points (the blue "x"es). To make all the control points visible select the pointer tool and double-click on the curve. Then move the control points around until the curve looks right.

Finally we set the advance width. Again with the pointer tool, move the mouse to the width line on the right edge of the screen, depress and drag the line back to a reasonable location.

Fig. 12: Tracing 6: make it look right



And we are done with this character.

Navigating to characters

The font view provides one way of navigating around the characters in a font. Simply scroll around it until you find the character you need and then double click to open a window looking at that character.

Typing a character will move to that character.

But some fonts are huge (Chinese, Japanese and Korean fonts have thousands or even tens of thousands of characters) and scrolling around the font view is an inefficient way of finding your character. `View -> Goto` provides a simple dialog which will allow you to move directly to any character for which you know the name (or encoding). If your font is a Unicode font, then this dialog will also allow you to find characters by block name (e.g. there is a pull-down list from which you may select Hebrew rather than Alef).

The simplest way to navigate is just to go to the next or previous glyph. And `View -> Next Glyph` and `View -> Prev Glyph` will do exactly that.

Loading background images better

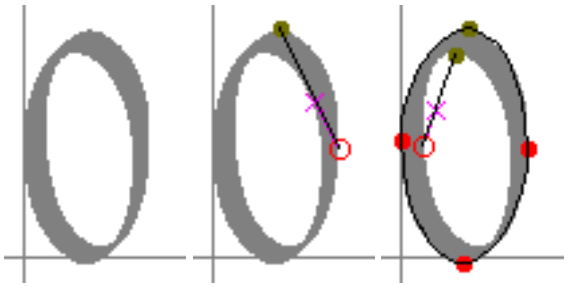
In the background image of the previous example the bitmap of the letter filled the canvas of the image (with no white borders around it). When FontForge imported the image it needed to be scaled once in the program. But usually when you create the image of the letter you have some idea of how much white space there should be around it. If your images are exactly one em high then FontForge will automatically scale them to be the right size. So in the following examples all the images have exactly the right amount of white space around them to fit perfectly in an em.

FontForge also has the ability to import an entire bitmap font (for example a "pk" or "gf" font produced by METAFONT or the "bdf" format developed by Adobe for bitmaps) to provide properly scaled background images for all characters in a font.

Creating the letter "o" — consistent directions

Let us turn our attention to the letter "o" which has a hole (or counter) in the middle. Open the outline view for the letter "o" and import a background image into it.

Fig. 13: Tracing o



Notice that the first outline is drawn clockwise and the second counter-clockwise. This change in drawing direction is important. Both PostScript and TrueType require that the outer boundary of a character be drawn in a certain direction (they happen to be opposite from each other, which is a mild annoyance); within FontForge all outer boundaries must be drawn clockwise, while all inner boundaries must be drawn counter-clockwise.

If you fail to alternate directions between outer and inner boundaries you may get results like the one on the left:  . If you fail to draw the outer contour in a clockwise fashion the errors are more subtle, but will generally result in a less pleasing result when the character is rasterized.

Technical and confusing: the exact behavior of rasterizers varies. Early PostScript rasterizers used a "non-zero winding number rule" while more recent ones use an "even-odd" rule. TrueType uses the "non-zero" rule. The example given above is for the "non-zero" rule. The "even-odd" rule would fill the "o" correctly no matter which way the paths were drawn (though there would probably be subtle problems with hinting).

To determine whether a pixel should be set using the even-odd rule, draw a line from that pixel to infinity (in any direction) and count the number of contour crossings. If this number is even the pixel is not filled. If the number is odd the pixel is filled. Using the non-zero winding number rule, the same line is drawn, contour crossings in a clockwise direction add 1 to the crossing count, while counter-clockwise contours subtract 1. If the result is 0 the pixel is not filled; any other result will fill it.

The command `Element -> Correct Direction` looks at each selected contour, figures out whether it qualifies as an outer or inner contour and reverses the drawing direction when the contour is drawn incorrectly.

Creating letters with consistent stem widths, serifs and heights

Many Latin (and Greek and Cyrillic, LGC for short) fonts have serifs, that is, special terminators at the end of stems. And in almost all LGC fonts there should only be a small number of stem widths (i.e. the vertical stem of "l" and "i" should probably be the same).

FontForge does not have a good way to enforce consistency, but it does have various commands to help you check for it, and to find discrepancies.

Let us start with the letter "l" and go through the familiar process of importing a bitmap and defining its outline.

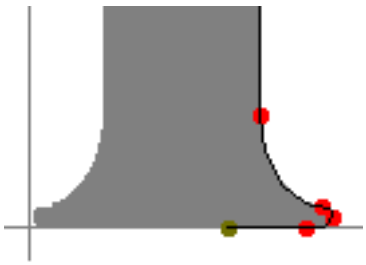
Fig. 14: Beginning "l"

Use the magnify tool to examine the bottom serif, and note that it is symmetric left to right.

Fig. 15: Magnified "l"

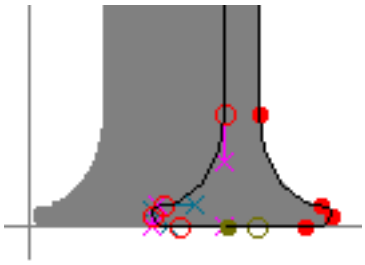
Trace the outline of the right half of the serif:

Fig. 16: Half traced "l"



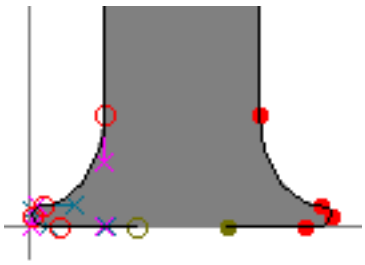
Select the outline, invoke `Edit -> Copy`, then `Edit -> Paste`; then invoke `Element -> Transform` and select `Flip` (from the pull down list) and check `Horizontal`.

Fig. 17: Pasted "l"



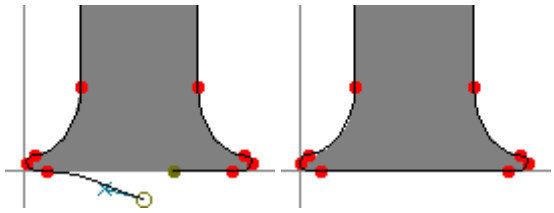
Drag the flipped serif over to the left until it snuggles up against the left edge of the character:

Fig. 18: Dragged "l"



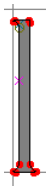
Deselect the path, and select one end point and drag it until it is on top of the end point of the other half.

Fig. 19: Joining "l"



Finish off the character.

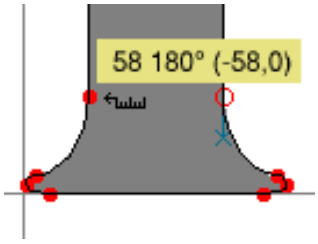
Fig. 20: Finished "l"



But there are two more things we should do. First let's measure the stem width, and second let's mark the height of the "l".

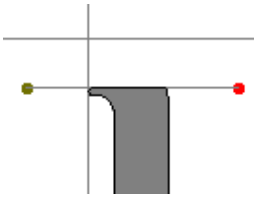
Select the ruler tool from the tool palette, and drag it from one edge of the stem to the other. A little window pops up showing the width is 58 units, the drag direction is 180 degrees, and the drag was -58 units horizontally, and 0 units vertically.

Fig. 21: Measuring stem width



Go to the layers palette and select the Guide radio box (this makes the guide layer editable). Then draw a line at the top of the "l". This line will be visible in all characters and marks the ascent height of this font.

Fig. 22: Making a guideline

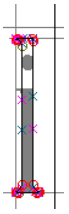


The "i" glyph looks very much like a short "l" with a dot on top. So let's copy the "l" into the "i"; this will automatically give us the right stem width and the correct advance width

The copy may be done either from the font view (by selecting the square with the "l" in it and pressing `Edit -> Copy`) or from the outline view (by `Edit -> Select All` and `Edit -> Copy`). Similarly, the Paste may be done either in the font view (by selecting the "i" square and pressing `Edit -> Paste`) or the outline view (by opening the "i" character and pressing `Edit -> Paste`).

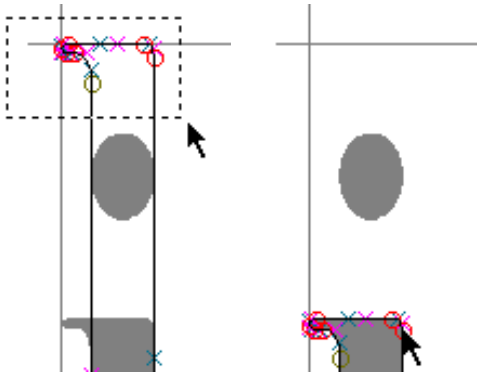
Now, import the "i" image, and copy and paste the "l" glyph.

Fig. 23: Import "i"



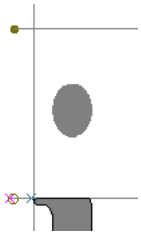
Select the top serif of the outline of the l and drag it down to the right height:

Fig. 24: Correcting "i"



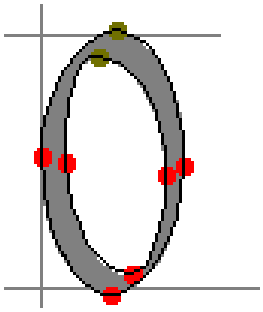
Go to the guide layer and add a line at the x-height:

Fig. 25: Making another guideline



Looking briefly back at the "o" we built before, you may notice that it reaches a little above the guideline we put in to mark the x-height (and a little below the baseline). This is called the overshoot or o-correction, and is an attempt to remedy an optical illusion. A curve actually needs to rise about 3% of its diameter above the x-height for it to appear on the x-height.

Fig. 26: Comparing "o" to guidelines



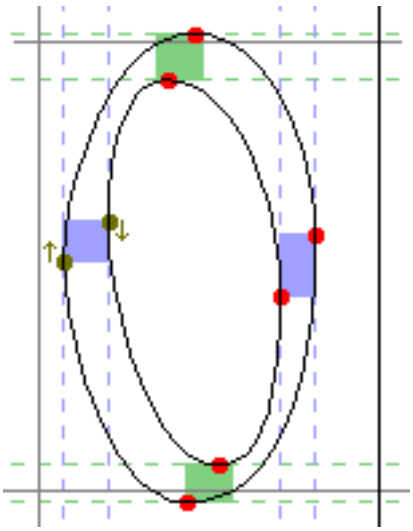
Continuing in this manner we can produce all the base glyphs of a font.

Hints

At small point sizes on display screens, computers often have a hard time figuring out how to convert a glyph's outline into a pleasing bitmap to display. The font designer can help the computer out here by providing what are called *hints*.

Basically every horizontal and vertical stem in the font should be hinted. FontForge has a command `Element -> Autohint` which should do this automatically. Or you can create hints manually — the easiest way is to select two points on opposite sides of a stem and then invoke `Hints -> Add HHint` or `Hints -> Add VHint` respectively for horizontal or vertical stems.

Fig. 27: "o" with hints



Accented letters

Latin, Greek and Cyrillic all have a large complement of accented characters. FontForge provides several ways to build accented characters out of base characters.

The most obvious mechanism is simple copy and paste: `Copy` the letter "A" and `Paste` it to "Ã"; then `Copy` the tilde accent and `Paste it Into` "Ã". (N.B. `Paste Into` is subtly different from `Paste`. `Paste` clears out the character before pasting, while `Paste Into` merges the clipboard into the character, retaining the old contents.) Then open up "Ã" and position the accent so that it appears properly centered over the A.

This mechanism is not particularly efficient; if you change the shape of the letter "A" you will need to regenerate all the accented characters built from it. To alleviate this, FontForge has the concept of a Reference to a character. Thus, you can `Copy a Reference` to "A", and `Paste` it, then `Copy a Reference` to tilde and `Paste it Into`, and then again adjust the position of the accent over the A.

Then if you change the shape of the A the shape of the A in "Ã" will be updated automagically — as will the width of "Ã".

But FontForge knows that "Ã" is built out of "A" and the tilde accent, and it can easily create your accented characters itself by placing the references in "Ã" and then positioning the accent over the "A". (Unicode provides a database which lists the components of every accented character (in Unicode)). Select "Ã", then apply `Element -> Build -> Build Accented` and FontForge will create the character by pasting references to the two components and positioning them appropriately.

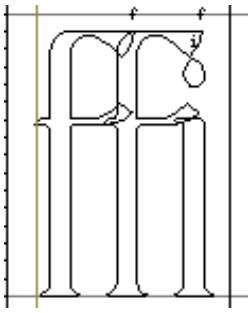
FontForge has a heuristic for positioning accents — most accents are centered over the highest point of the character — but sometimes this will produce bad results (e.g. if the one of the two stems of "u" is slightly taller than the other the accent will be placed over that stem, rather than being centered over the character), so you should be prepared to look at your accented characters after creating them. You may need to adjust one or two (or you may need to redesign your base characters slightly).

Ligatures

One of the great drawbacks of the standard Type 1 fonts from Adobe is that none of them come with "ff" ligatures. Lovers of fine typography tend to object to this. FontForge can help you overcome this flaw (whether it is legal to do so is a matter you must settle by reading the license agreement for your font). FontForge cannot create a nice ligature for you, but what it can do is put all the components of the ligature into the character with `Element -> Build -> Build Composite`. This makes it slightly easier (at least in Latin) to design a ligature.

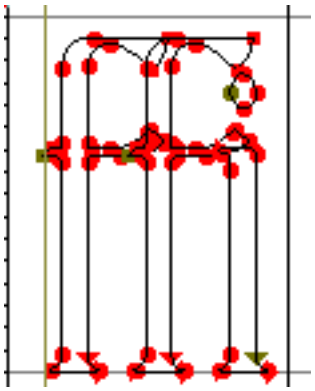
Use the `Element -> Char Info` dialog to name the character; we'll use "ffi" as an example. This is a standard name and FontForge recognizes it as a ligature consisting of f, f and i. Apply `Element -> Default ATT -> Common Ligatures` so that FontForge will store the fact that it is a ligature. Then use `Element -> Build -> Build Composite` to insert references to the ligature components.

Fig. 28: ffi made of references



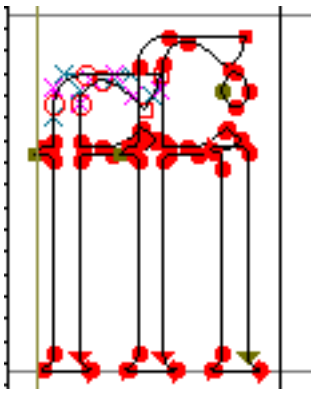
Use `Edit -> Unlink References` to turn the references into a set of contours.

Fig. 29: ffi without references



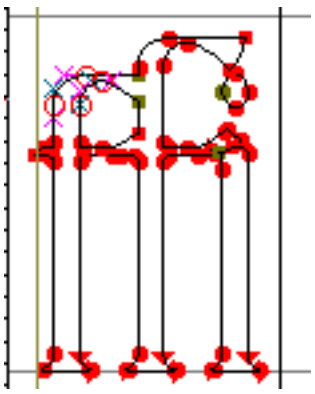
Adjust the components so that they will look better together. Here the stem of the first f has been lowered.

Fig. 30: ffi adjusted



Use `Element -> Remove Overlap` to clean up the character.

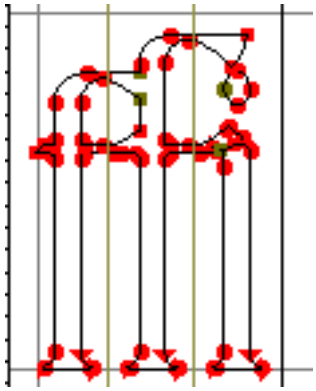
Fig. 31: ffi cleaned up



Some word processors will allow the text editing caret to be placed inside a ligature (with a caret position between each component of the ligature). This means that the user of that word processor does not need to know s/he is dealing with a ligature and sees behavior very similar to what s/he would see if the components were present. But for the word processor to be able to do this, it must have some information from the font designer giving the appropriate locations of caret positions.

As soon as FontForge notices that a character is a ligature it will insert enough caret location lines into it to fit between the ligature's components. FontForge places these on the origin, and if you leave them there FontForge will ignore them. But once you have built your ligature you might want to move the pointer tool over to the origin line, press the button and move the caret lines to their correct locations. (Only AAT and OpenType support this).

Fig. 32: ffi with ligature carets



Metrics

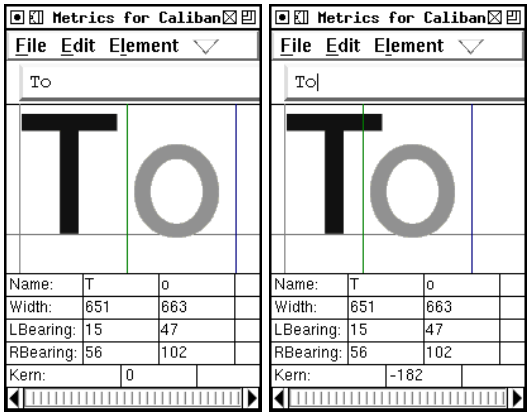
Once you have created all your glyphs, you should presumably examine them to see how they look together. There are three commands designed for this:

- `Windows->New Metrics View` opens a window which displays several glyphs at a very large size. You can change the advance width of each glyph here to make a more pleasing image.
- `File->Print` prints a sample text using the font, or all the glyphs of the font, or several glyphs one per page, or several glyphs at a waterfall of point sizes.
- `File->Display` opens a dialog which allows you to display a sample text in this (or indeed several) fonts.

Kerning

Even in fonts with the most carefully designed metrics there are liable to be some character combinations which look ugly. Some combinations are fixed by building ligatures, but most are best approached by kerning the intercharacter spacing for that particular pair.

Fig. 33: kerning in the Metrics view



In the above example the left image shows the unkerned text, the right shows the kerned text. To create a kerned pair, select the two glyphs, then use `Windows->New Metrics View` and move the mouse to the rightmost character of the pair and click on it, the line (normally the horizontal advance) between the two should go green (and becomes the kerned advance). Drag this line around until the spacing looks nice.

Checking a font for common problems

After you have finished making all the characters in your font, you should check it for inconsistencies. FontForge has a command, `Element->Find Problems`, which is designed to find many common problems (as you might guess).

Simply select all the characters in the font and then bring up the `Find Problems` dialog. Be warned though: Not everything it reports as a problem is a real problem, some may be an element of the font's design that FontForge does not expect.

The dialog can search for many types of problems:

- Stems which are close to but not exactly some standard value.
- Points which are close to but not exactly some standard height.
- Paths which are almost but not quite vertical or horizontal.
- Control points which are in unlikely places.
- Points which are almost but not quite on a hint.
- and more ...

I find it best just to check for a few similar problems at a time; switching between different types of problems can be distracting.

Generating a font

The penultimate[2] stage of font creation is generating a font. N.B.: The `File->Save` command in FontForge will produce a format that is only understood by FontForge and is not useful in the real world.

You should use `File->Generate` to convert your font into one of the standard font formats. FontForge presents what looks like a vast array of font formats, but in reality there are just several variants on a few basic font formats: PostScript Type 1, TrueType, OpenType (and for CJK fonts, also CID-keyed fonts) and SVG.

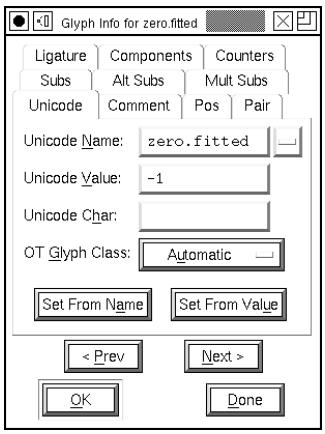
OpenType advanced typography

In OpenType and Apple's Advanced Typography fonts it is possible for the font to know about certain common glyph transformations and provide information about them to a word processor using that font (which presumably could then allow the user access to them).

Simple substitutions Suppose that we had a font with several sets of digits: monospaced digits, proportional digits and lower case (old style) digits. One of these styles would be chosen to represent the digits by default (say the monospaced digits). Then we could link the default glyphs to their variant forms.

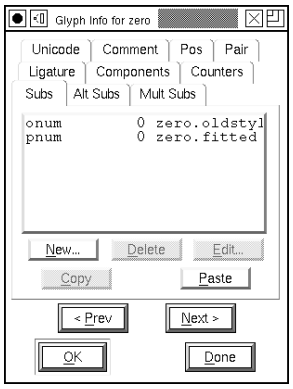
First we should name each glyph appropriately (proportional digits should be named "zero.fitted", "one.fitted", and so forth, while oldstyle digits should be named "zero.oldstyle", "one.oldstyle", and so forth. Use the `Element->Glyph Info` command to name them.

Fig. 34: Naming a glyph



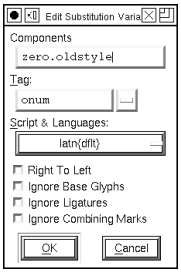
To link the glyphs together we invoke `Element->Glyph Info` again, this time on the default glyph and select the `Subs` tab. This provides a list of all simple substitutions defined for this glyph.

Fig. 35: Providing substitutions



Pressing the `[New]` button will allow you to add a substitution:

Fig. 36: Editing substitutions



Each substitution must contain: the name of a glyph to which it is to be mapped, a four-character OpenType tag used to identify this mapping, and a script and language in which this substitution is active. There is a pulldown menu which you can use to find standard tags for some common substitutions (the tag for oldstyle digits is 'onum'). This substitution is for use in the Latin script and for any language; again there is a pulldown menu to help choose this correctly.

Contextual substitutions OpenType and Apple also provide contextual substitutions. These are substitutions which only take place in a given context and are essential for typesetting Indic and Arabic scripts.

In OpenType a context is specified by a set of patterns that are tested against the glyph stream of a document. If a pattern matches, then any substitutions it defines will be applied.

Instead of an Indic example, let us take something I'm more familiar with, the problem of typesetting a

2. The final stage of font creation would be installing the font. This depends on what type of computer you use and I shan't attempt to describe all the possibilities here.

Latin script font where the letters “b”, “o”, “v”, and “w” join their following letter near the x-height, while all other letters join near the baseline.

Thus we need two variants for each glyph, one that joins (on the left) at the baseline (the default variant) and one which joins at the x-height. Let us call this second set of letters the “high” letters and name them “a.high”, “b.high”, and so forth.

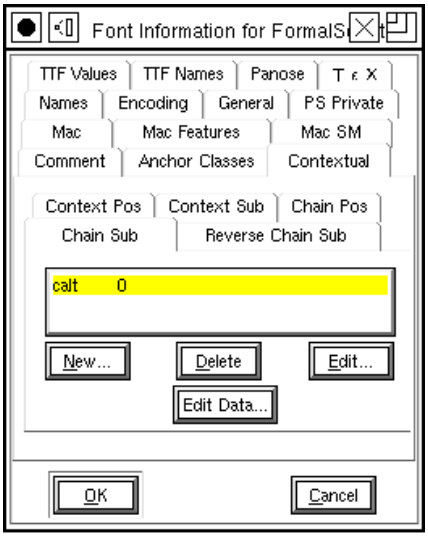
Fig. 37: Incorrect & correct script joins

We divide the set of possible glyphs into three classes: the letters “bovw”, all other letters, and all other glyphs. We need to create two patterns; the first will match a glyph in the “bovw” class followed by a glyph in the “bovw” class, while the second will match a glyph in the “bovw” class followed by any other letter. If either of these matches the second glyph should be transformed into its high variant.

The first thing we must do is create a simple substitution mapping for each low letter to its high variant. Let us call this substitution by the four character OpenType tag “high”. We use `Element -> Glyph Info` as before except that here we use the special “script/language” called “ — Nested — ” (an option in the pull-down menu).

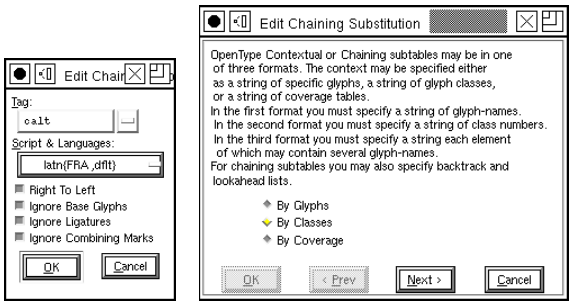
The tricky part is defining the context. This is done with the `Contextual` tab in the `Element -> Font Info` dialog, revealing five different types of contextual behavior. We are interested in contextual chaining substitutions.

Fig. 38: Font Info showing Contextual Subs



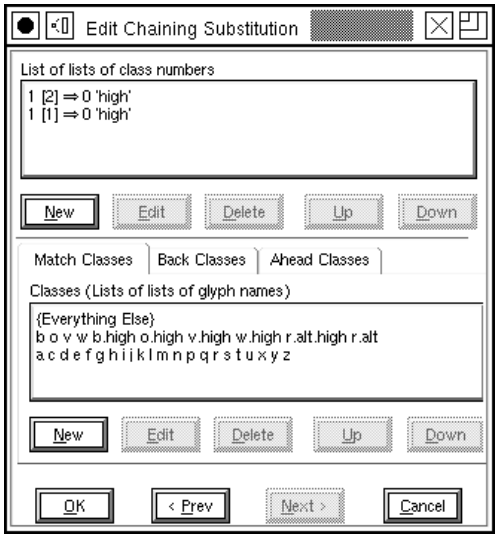
You can add a new entry by pressing the `[New]` button. This brings up a series of dialogs. The first requests a four character OpenType tag and a script/language, much as we saw earlier. The next dialog allows you to specify the overall format of the substitution.

Fig. 39: Tag & Script dialog and format of contextual chaining substitution



The next dialog finally shows something interesting. At the top are a series of patterns to match and substitutions that will be applied if the string matches. Underneath that are the glyph classes that this substitution uses.

Fig. 40: Overview of the contextual chaining substitution



A contextual chaining dialog divides the glyph stream into three categories: those glyphs before the current glyph (these are called backtracking glyphs), the current glyph itself (you may specify more than one, and this (these) glyph(s) may have simple substitutions applied to them), and finally glyphs after the current glyph (these are called lookahead glyphs).

Each category of glyphs may divide glyphs into a different set of classes, but in this example we use the

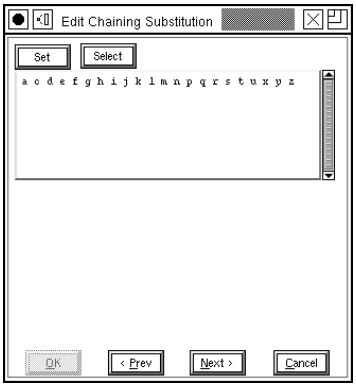
same classes for all categories (this makes it easier to convert the substitution to Apple's format).

The first line (in the "Lists of lists" field) should be read thus: If a backtracking glyph in class 1 is followed by a match glyph in class 2, then location 0 in the match string (that is, the first glyph) should have the simple substitution 'high' applied to it. If you look at the glyph class definitions you will see that class 1 includes those glyphs which must be followed by a high variant, so this seems reasonable.

The second line is similar except that it matches glyphs in class 1. Looking at the class definitions we see that classes 1 & 2 include all the letters, so these two lines mean that if any letter follows one of "bovw" then that letter should be converted to its 'high' variant.

To edit a glyph class, double click on it. To create a new one press the [New] button (under the class list).

Fig. 41: Editing glyph classes



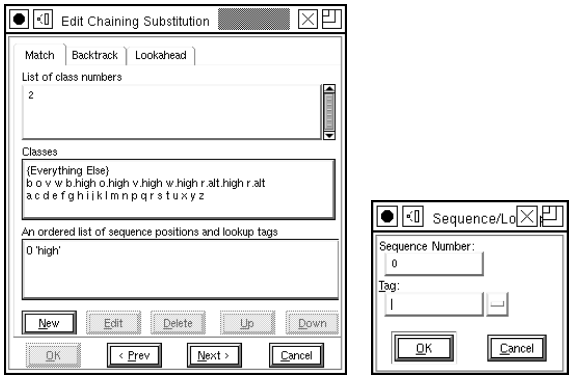
This produces another dialog showing all the names of all the glyphs in the current class. Pressing the [Select] button will set the selection in the font window to match the glyphs in the class, while the [Set] button will do the reverse and set the class to the selection in the font window. These provide a shortcut to typing in a lot of glyph names. Pressing the [Next] button defines the class and returns to the overview dialog.

To edit a pattern double click on it, or to create a new one press the [New] button (under "Lists of lists").

Again the pattern string is divided into three categories, those glyphs before the current one, the current one itself, and any glyphs after the current one. You choose which category of the pattern you are editing with the tabs at the top of the dialog. Underneath these is the subset of the pattern that falls within the current category, the classes defined for this category, and finally the substitutions for the current glyph(s). Clicking on one of the classes will add the class number to the pattern.

To edit a substitution double click on it, or to create a new one press the [New] button (under "An ordered list ..."). The sequence number specifies which glyph among the current glyphs should be modified, and the tag specifies a four character substitution name.

Fig. 42: Adding matches and substitutions



Apple advanced typography

Some of Apple's typographic features can be readily converted into equivalent OpenType features, while others cannot be.

Non-contextual ligatures, kerning and substitutions can generally be converted from one format to another. Apple uses a different naming convention and defines a different set of features, but as long as a feature of these types is named in both systems, conversion is possible.

Contextual substitutions Apple specifies a context with a finite state machine, which is essentially a tiny program that looks at the glyph stream and decides what substitutions to apply.

Each state machine has a set of glyph class definitions (just as in the OpenType example), and a set of states. The process begins in state 0 at the start of the glyph stream. The computer determines what class the current glyph is in and then looks at the current state to see how it will behave when given input from that class. The behavior includes the ability to change to a different state, advancing the input to the next glyph, applying a substitution to either the current glyph or a previous one (the "marked" glyph).

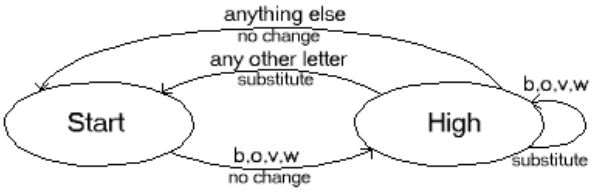
Using the same example of a Latin script font ...

We again need a simple substitution to convert each letter into its high alternate. The process is the same as it was for OpenType, and indeed we can use the same substitution.

Again we divide the glyphs into three classes (Apple gives us some extra classes whether we want them or not, but conceptually we use the same three classes as in the OpenType example). We want a state machine with two states (again Apple gives us an extra state for free, but we shall ignore that); one is the start state (the base state,

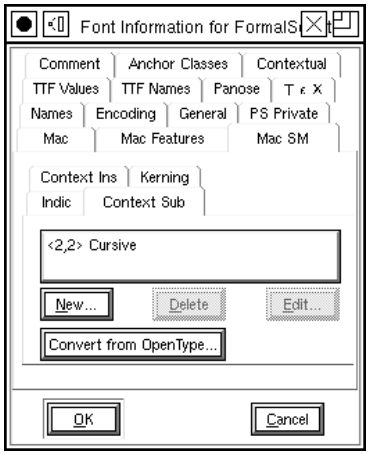
where nothing changes), and the other is the state where we've just read a glyph from the "bovw" class.

Fig. 43: State machine



Again we use the `Element -> Font Info` dialog and the `Mac SM` tag to look at the contextual substitutions available. Again there are several types of contextual behavior, and we are interested in contextual substitutions.

Fig. 44: Font Info showing State Machines



Double clicking on a state machine, or pressing the `[New]` button provides an overview of the given state machine. At the top of the dialog we see a field specifying the feature/setting of the machine; this is Apple's equivalent of the OpenType four-character tag. Under this is a set of class definitions, and at the bottom is a representation of the state machine itself. See fig. 45.

Double clicking on a class brings up a dialog similar to that used in OpenType:

Fig. 46: Editing Apple glyph classes

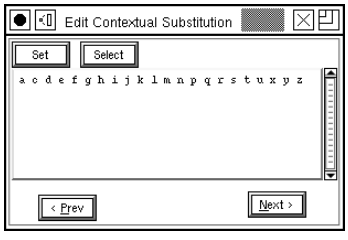
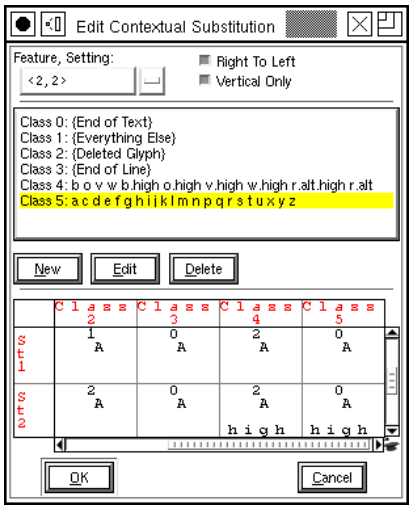


Fig. 45: Overview of a State Machine

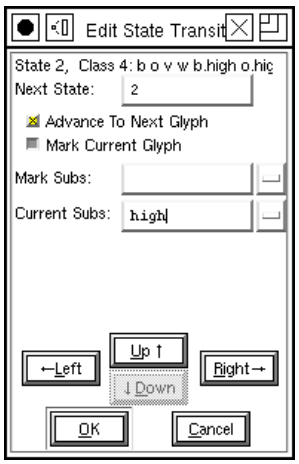


Clicking on a transition in the state machine (there is a transition for each state/class combination) produces a transition dialog.

This controls how the state machine behaves when it is in a given state and receives a glyph in a given class. In this example it is in state 2 (which means it has already read a "bovw" glyph), and it has received a glyph in class 4 (which is another "bovw" glyph). In this case the next state will be state 2 again (we will have just read a new "bovw" glyph), read another glyph and apply the "high" substitution to the current glyph.

At the bottom of the dialog is a series of buttons that allow you to navigate through the transitions of the state machine.

Fig. 47: Transition dialog



Pressing `[OK]` many times will extract you from this chain of dialogs and add a new state machine to your font.

Appendix: Additional features

FontForge provides many more features, further descriptions of which may be found at

`http://fontforge.sf.net/overview.html`

Here is a list of some of the more useful of them:

- Users may edit characters composed of either third order Bézier splines (for PostScript fonts) or second order Béziers (for TrueType fonts) and may convert from one format to another.
- FontForge will retain both PostScript and TrueType hints, and can automatically hint PostScript fonts.
- FontForge allows you to modify most features of OpenType's GSUB, GPOS and GDEF tables, and most features of Apple's morx, kern, lcar and prop tables. Moreover it can often convert from one format to another.
- FontForge has support for Apple's font formats. It can read and generate Apple font files both on and off a Macintosh. It can generate the FOND resource needed for the Mac to place a set of fonts together as one family.
- FontForge allows you to manipulate bitmap fonts as well as outline fonts. It has support for many formats of bitmap fonts (including TrueType's embedded bitmaps — both the format prescribed by Apple and that specified by Microsoft).
- FontForge can interpolate between two fonts (subject to certain constraints) to yield a third font between the two (or even beyond). For instance given a "Regular" and a "Bold" variant it could produce a "DemiBold" or even a "Black" variant.
- FontForge also has a command (which often fails miserably) which attempts to change the weight of a font.
- FontForge can automatically guess at widths for characters, and even produce kerning pairs automatically.
- FontForge has some support for fonts with vertical metrics (in Japanese, Chinese and Korean fonts), and some support for right to left fonts (Arabic, Hebrew, Linear-B, etc.).
- FontForge has some support for PostScript (and pdf) Type 3 fonts, allowing for glyphs with different strokes and fill and images.
- FontForge has a scripting language which allows batch processing of many fonts at once.

The Free UCS Outline Fonts Project — an Attempt to Create a Global Font

Primož Peterlin
University of Ljubljana
Faculty of Medicine, Institute of Biophysics
Lipičeva 2, SI-1000 Ljubljana, Slovenia
primoz.peterlin@biofiz.mf.uni-lj.si
http://biofiz.mf.uni-lj.si/~peterlin/

Abstract

In February 2002, the Free UCS (Universal Character Set) Outline Fonts project (http://savannah.gnu.org/projects/freefont/) was started. Exercising the open-source approach, its aim is to provide a set of free Times-, Helvetica- and Courier-lookalikes available in the OpenType format, and progressively cover the complete ISO 10646/Unicode range. In this stage of the project, we focus mainly on two areas: collecting existing fonts that are both typographically and license-wise (i.e., GNU GPL) compatible and can be included to cover certain parts of the character set, and patching up smaller areas that are not yet covered. Planned future activities involve typographic refinement, extending kerning information beyond the basic Latin area, including TrueType hinting instructions, and facilitating the usage of fonts with various applications, including the TEX/Ω typesetting system.

Résumé

Le projet de *fontes vectorielles libres pour UCS* (http://savannah.gnu.org/projects/freefont/) a démarré en février 2002. À travers l'*Open Source*, le but de ce projet est de fournir un ensemble de clones libres de Times, Helvetica et Courier, disponibles dans le format OpenType, et couvrant progressivement l'ensemble des caractères d'ISO 10646/Unicode. À ce stage du projet nous nous concentrons sur deux domaines : la collection de fontes existantes qui soient compatibles avec notre projet, aussi bien du point de vue typographique que du point de vue de la licence (GNU GPL), qui puissent être intégrées pour couvrir certaines parties de l'ensemble de caractères ; et d'autre part, faire des ajouts de petites régions qui n'ont pas encore été couvertes. Nos activités futures prévoient le perfectionnement typographique, le crénage au-delà de l'alphabet latin, l'incorporation de *hints* TrueType, et le support de plusieurs applications, dont aussi les systèmes TEX et Ω.

Introduction

The aim of the Free UCS Outline Font project is to provide a standardised set of glyphs which make a harmonised design despite including glyphs from different scripts (Latin, Cyrillic, Greek, Armenian, etc.). It is clear that this requires compromises at the cost of typographic finesse (see [11] for a discussion of typographic compromises regarding the Greek alphabet), yet the end result must look acceptable for general use, including electronic mail, world-wide web, and text editors.

While this is clearly not the first attempt to create a typeface covering glyphs beyond Latin, previous attempts are not as numerous as one might imagine. Bigelow [5] quotes *Romulus* by Jan van Krimpen from 1931 as one of the first examples. Another example is Николай Николаевич Кудряшов [Nikolaĭ Nikolaevich Kudryashov]'s *Encyclopaedia* (Кудряшовская енциклопедическая [Kudryashovskaya entsiklopedicheskaya]) family (1960–1974), originally designed for the third edition of the Great Soviet Encyclopaedia which contains thousands of glyphs, including Cyrillic, Latin, and Greek letters in serif and sans-serif styles, as well as other special signs and symbols. Most popular typefaces such as Stanley Morison's Times Roman, Max Miedinger's Helvetica, Adrian Frutiger's Univers and Eric Gill's Gill Sans were originally designed for the Latin alphabet and later, as they gained popularity, extended to Greek and Cyrillic alphabets.

A typeface covering the whole ISO 10646/Unicode range has to be designed flexibly enough to allow addition of scripts not belonging to the Western typographic tradition. The task is not easy, and it is not surprising that so far there are very few aesthetically satisfying typographic solutions. Among those one has to mention Bigelow and Holmes's *Lucida Sans Unicode* [5] and Haralambous and Plaice's Ω [12]. Other solutions like *Arial Unicode* [16] and James Kass's compendium *Code2000* [14] are too varied in style to form a unified typeface.

Design issues

Historical and cultural context. It is clear that a typeface aiming to cover "all the scripts of the world" can not rely on historical styles (e.g., Renaissance, Baroque, etc.) known from Western typography, as most of the world has not experienced these periods in the evolution of typography. Trying to extend them to non-European scripts[1] is as inappropriate as, say, trying to design a Latin alphabet in Kufi style.

Even the apparently logical division into seriffed and sans-serif typefaces is eurocentric. To see why, one only has to think of the origin of Latin capital letters. Serifs are an invention of Roman stone-carvers — a smaller stroke perpendicular to the main stroke was added to provide a uniform smooth finishing of the stroke. Here, the visual effect followed the available technology. The modulated stroke — another ancient Roman typographic invention — is an opposite example, where the technology followed the visual effect [13]. The need to strengthen vertical strokes arose once the Romans started to erect monuments of monumental proportions, where the inscriptions were no longer in the eye-height. While the rain took away the paint from the vertical strokes, dirt and dust accumulated in the horizontal strokes, which thus appeared optically heavier. *Physical* broadening of vertical strokes was introduced as a compensation, in order for the horizontal and vertical strokes to look *optically* equivalent. The Roman technological innovations survived for 2 000 years in European typography. However, in countries where the letters were painted with brush onto silk or carved with a needle onto a palm leaf rather than carved into the stone with mallet and chisel, such typographic development never happened.

Similar concerns about eurocentrism are valid for differing the upright and the italic forms, or for that matter, even between *majuscules* and *minuscules*, capital and small letters. Even differing between upright and slanted forms is questionable, as slanted forms make no sense in, say, most native Asian scripts.

The least questionable seems to be the differences based on the weight of the typefaces, which appears to be almost universal. The only exception known to the author is Ethiopic,[2] where the words were traditionally emphasised by printing them in another colour (red in religious texts, blue in imperial decrees) or by underlining them or enclosing them in ovals.

Legibility. A typeface which includes non-Latin scripts offers an opportunity for exploring the "universal" parameters of legibility. While there has been a wealth of publication on legibility centred on Latin script [20, 22, 9, 15], exploring the factors like weight, serif vs. sans-serif faces, x-height, capitalisation etc., it is clear that such differences are minor compared to the differences in shape between Latin and, say, Hebrew, Arabic, Devanagari or Tamil, which nevertheless provide roughly the same level of legibility. Lacking comparative cross-script studies, the experiments in legibility remain in the realm of the typographer's intuition. On the brighter side, making multi-script typefaces available, albeit not perfect, is a step towards the world where such cross-cultural studies will be easier to achieve.

Methodology

The basic idea behind the free UCS outline fonts project was to collect various available free outline fonts, covering single national scripts, and to compile them into a large font using the ISO 10646/Unicode coded character set [21], taking into consideration typographic and legal compatibility, and filling in the missing areas on the way. The whole development was planned to be carried in the open-source manner, with many developers using a central repository.

The general requirement for technical realisation was that typefaces need to be available as scalable vector fonts. The actual technical realisation (PostScript Type 1 [1] uses cubic Bézier splines, while TrueType [3] uses quadratic ones) was considered secondary, because at least in principle it is possible to transform the fonts from one form to another. In reality, though, no known transformations is completely lossless — kerning and hints are usually the most volatile.

Licenses of used sources. We anticipated that our result will contain glyphs from many different sources. Thus, special attention was given to the license under which a font is released. The license we looked for should allow redistribution, modification and distribution of modified font files. Many free and open-source licenses fulfil this requirement. As the URW++ core PostScript fonts, the Ω-Serif and some other major sources were released under the GNU General Public License [10], we adopted it for our project as well. The license itself is suited for programs rather than typefaces and its application to fonts may be legally dubious, even though, say, PostScript Type 1 fonts are perfectly legal programs written in PostScript. We were not able to find any license in the same spirit pertaining specifically to fonts, though. It may be worth seeing the final license of the Bitstream Vera fonts, which were announced in January 2003 to be soon available under a license in the open-source spirit.

We thus limited our search to fonts which were not only typographically compatible, but also specifically released under the GNU GPL. In some cases, where

1. Even though historical styles differ throughout Europe, Europe is nevertheless treated here as a historical and cultural unity when contrasted with the rest of the world.

2. Daniel Yacob, personal communication.

the fonts were released under less clearly defined terms ("free", "public domain" etc.), we contacted the authors and asked for permission to use their work under the terms of the GNU GPL. Most authors agreed, and while none disagreed, some of the emails remained unanswered. In such cases we were unable to confirm whether the recipient received our email at all. These fonts still wait to be included in the free UCS outline font collection.

Choice of typefaces. Even though Bigelow and Holmes claim that their primary reason for choosing a sans-serif font is because it carries least historical and cultural associations [5, p. 1003], and despite the expressed concerns about eurocentrism, we decided to develop in parallel three different families, modelled after Times Roman, Helvetica and Courier, and specifically derived from URW++ typefaces Nimbus Roman No. 9, Nimbus Sans and Nimbus Mono. Reflecting the nature of the project, we dubbed them as Free Serif, Free Sans and Free Mono.

Aside from covering three different letterforms — one monospaced and two proportional, one with modulated strokes and another with unmodulated strokes — the primary reason for choosing these three typefaces was their ubiquity. Since they have been extremely popular for many decades, and present in the desktop publishing world-wide for over twenty years, they inspired numerous local designs around the world, where non-Latin glyphs were designed specifically to blend with one of the above-named typefaces. While we realise that many of these designs introduce typographic practice alien to a traditional local design, we also recognise that the designs done by native speakers reflect the local typographic knowledge and its evolving typographic rules.

Tools. While we paid special attention to the licensing of the used sources, no particular attention was given to the licensing issues of the tools used, as the font tools generally don't imply the licenses under which the fonts are released. Still, we ended up using exclusively open-sourced tools. The choice was clearly influenced by the fact that most participants of the project use Linux, and while none of the numerous popular font editors from other platforms have been ported to this platform, we have some excellent free tools available.

PfaEdit Without George Williams's excellent font editor PfaEdit,[3] this project would not have been possible at all. PfaEdit is a visual font editor that allows copying and modifying glyphs and most other things expected from a font editor, reads and writes most popular outline font formats, and has proved to be stable enough to allow working on large glyph sets. Even though the program was mature enough to be used as a tool back in late 2001, the pace of its development has not diminished over the past year, and its author is very responsive to bug reports.

Choosing PfaEdit as our main tool also influenced the decision of the native font format used in the free UCS outline fonts project. Since we use a CVS repository for the bookkeeping of font additions and modifications, we were looking for a format suitable to differential changes. A format was considered suitable if local changes in typeface design remained localised in the font file rather than propagating themselves across the file. This requirement ruled out compiled binary formats, as well as PostScript Type 1 `eexec` encoding. PfaEdit's native format SFD, similar to Adobe's BDF format in its structure, proved to be a suitable choice. The fonts are thus archived in the SFD format, and PfaEdit is used to create fonts in other outline formats.

PfaEdit's only major drawback may be its weak support for TrueType instructions or "hints". Since TrueType is not the native format used by the free UCS outline font project, we are looking either for automated generation of TrueType hints, or for separate files with TrueType instructions which could be compiled together with the glyph outlines in the SFD format to produce a hinted TrueType font. Any solution requiring manual modification of binary TrueType fonts is not acceptable, as TrueType fonts are created automatically each time from the SFD sources. While PfaEdit will probably eventually produce acceptable TrueType hints in an automated way, other possibilities have to be investigated in the meantime.

TrueType tools Rogier van Dalen's TrueType tools[4] might be the proper solution for the TrueType hinting problem. Van Dalen's approach uses a separate file with TrueType instructions written in a high-level language not unlike C, which can be compiled together with an unhinted TrueType font file to produce a hinted TrueType font file. If this approach is adopted, each TrueType font file will be automatically produced from two source files: the SFD file containing the glyph outlines and the TTI file containing the TrueType instructions for hinting. At the moment of this writing, the main concern is the estimated amount of work needed for producing TrueType hinting instructions in a manual way.

TTX While the SFD format fits present needs just fine, the future might require distributing font sources in some other open format. For this purpose we

3. Since renamed to FontForge: `http://fontforge.sourceforge.net/`

4. `http://home.kabelfoon.nl/~slam/fonts/`

are considering TTX, the TrueType to XML converter.[5] Given the pace at which PfaEdit currently evolves, it may well be that PfaEdit itself will support XML before we will actually feel the need for it.

Towards ISO 10646/Unicode compliance. As of version 3.2, Unicode defines 95 156 encoded characters. Taking into account three families (Free Serif, Free Sans and Free Mono), two weights (normal and bold) and two shapes (regular and italic/oblique), this means designing almost 1.2 million glyphs for the free UCS font project. Being a volunteer project, free UCS outline fonts project grows mainly in a non-systematic way — when a suitable set of glyphs is found or donated, we add it to the font. Often, a wider community can benefit from scratching one's own itch, e.g. creating a set of APL glyphs [7, 8].

Still, we felt the need to add the glyphs in a more systematic way in parallel with the spontaneous growth of the project. For that purpose, we use the multilingual European subsets as defined by Comité Européen de Normalisation [6] as a guideline:

- MES-1, a Latin repertoire based on ISO/IEC 6937:1994 (335 characters)
- MES-2, a Latin, Cyrillic, and Greek repertoire based on ENV 1973:1996 (1062 characters)
- MES-3, a repertoire needed to write all the languages of Europe and transliterate between them. MES-3A is a script-based, non-fixed collection, while MES-3B is a fixed subset of 2819 characters.

Character-to-glyph translation. As the ISO 10646/Unicode standard encodes *characters* rather than *glyphs*, it is clear that for acceptable rendering of many languages more glyphs than those corresponding to Unicode characters are needed. This is particularly important for rendering Indic languages and Syriac which contain numerous ligatures not encoded in the ISO 10646/Unicode standard.

During the 1990's, several initiatives for creating "smart fonts" were started, i.e., fonts being not only containers of glyphs but also incorporating some logical rules for glyph substitution, glyph position, contextual rendering, etc. Among those were Apple Advanced Typography [2], SIL Graphite [18] and Adobe/Microsoft OpenType [17]. From these three, OpenType seems to be grabbing the largest market share.

Results

The project was conceived in December 2001 and was approved for hosting on the Savannah Web site[6] in February 2002. As of March 2003, a total of 17 794 characters are encoded. We are just a couple of dozen glyphs short of MES-1 compliance in Free Serif and Free Sans, while Free Mono is already MES-1 compliant, and approximately 3 500 glyphs short of MES-2 compliance.

Kerning. Aside from the character set compliance and the already mentioned TrueType hinting problem, there remain some other tasks. One of them is kerning, which is currently present only in the Latin portions of the font. Kerning non-Latin scripts requires typographic knowledge which we currently do not possess. We believe that the open nature of the project will eventually attract somebody with the necessary expertise who is willing to contribute.

Discussion

The open-source development model has not been previously tested for font development. While it may be arguable whether such a development model can be applied at all to creation of works of art, it is also worth noting that most contributors have a technical and scientific background, and probably none of them perceives him- or herself as an artist.

While there is no doubt that a skilled typographer would be able to design a multi-script font vastly exceeding what we have done in typographic beauty and consistency, we must not overlook the fact that designing 1 200 000 glyphs is probably beyond the capability of a single person, even if we neglect the financial part of such an endeavour.

Free UCS outline fonts project and TEX. So far, the Free UCS outline fonts project has taken from the TEX community more than it has repaid. Thanks to Péter Szabó and his TEXtrace program (`http://www.inf.bme.hu/~pts/textrace/`) [19], it is easy to transform Metafonts into PostScript Type 1 fonts which can be edited by PfaEdit. Karel Piška demonstrated the technique for Indian Metafonts during the TUG 2002 conference in Thiruvananthapuram, India.

Even though the free UCS outline fonts were designed primarily for screen use, we can certainly raise the question of using the fonts for typesetting with TEX or Ω. As of March 2003, this would require extensive work, such as splitting the fonts into smaller fonts containing no more than 256 glyphs. However, the current development [4] promises that there might be a more direct way of using OpenType fonts with Ω in the future.

Acknowledgments

I would like to thank the following authors who contributed to the project by allowing their work to be included in the free UCS outline fonts project. In alphabetical order: Berhanu Beyene, Daniel Shurovich Chirkov,

5. `http://www.letterror.com/code/ttx/`
6. `http://savannah.gnu.org/projects/freefont`

Prasad A. Chodavarapu, Vyacheslav Dikonov, DMS Electronics, Valek Filippov, Shaheed R. Haque, Yannis Haralambous, Angelo Haritsis, Jeroen Hellingman, Maxim Iorsh, Mohamed Ishan, Manfred Kudlek, Harsh Kumar, Sushant Kumar Dash, Olaf Kummer, Noah Levitt, Jochen Metzinger, Anshuman Pandey, Hardip Singh Pannu, Thomas Ridgeway, Young U. Ryu, Virach Sornlertlamvanich, M.S. Sridhar, Sam Stepanyan, URW++ Design & Development GmbH, Frans Velthuis, and the Wadalab Kanji Committee.

References

[1] Adobe Systems Incorporated. *Adobe Type 1 Font Format*. Addison-Wesley, 1990. Also available on the Web: `http://partners.adobe.com/asn/developer/PDFS/TN/T1_SPEC.PDF`.

[2] Apple Computer, Inc. Apple advanced typography. Available on the Web: `http://developer.apple.com/fonts/`, 1994.

[3] Apple Computer, Inc. TrueType reference manual. Available on the Web: `http://developer.apple.com/fonts/TTRefMan/`, 1999.

[4] Gábor Bella, Anish Mehta, and Yannis Haralambous. Adapting Omega to OpenType fonts. In *EuroTEX 2003* (this volume), 2003.

[5] Charles Bigelow and Kris Holmes. The design of a Unicode font. *Electronic Publishing*, 6(3):999–1015, 1993.

[6] CEN. Information technology – multilingual European subsets in ISO/IEC 10646-1. Technical Report CWA 13873:2000, Comité Européen de Normalisation, Technical Committee 304, 2000. Also available on the Web: `http://www.evertype.com/standards/iso10646/pdf/cwa13873.pdf`.

[7] Phil Chastney. An APL Unicode font. *Vector*, 16(1):75–85, 1999.

[8] Phil Chastney. APL Unicode font – extended. *Vector*, 17(3):139–142, 2001.

[9] Rudi W. de Lange, Hendry L. Esterhuizen, and Derek Beatty. Performance differences between Times and Helvetica in a reading task. *Electronic Publishing*, 6(3):241–248, 1993.

[10] Free Software Foundation. GNU General Public License, version 2. Available on the Web: `http://www.gnu.org/copyleft/gpl.html`, 1991.

[11] Yannis Haralambous. From Unicode to typography, a case study: the Greek script. In *Proceedings of 14th International Unicode Conference*, pages b.10.1–b.10.36, Boston, 1999.

[12] Yannis Haralambous and John Plaice. Ω, a TEX extension including Unicode and featuring Lex-like filtering processes. In *EuroTEX Proceedings*, pages 154–167, Gdańsk, 1994.

[13] Georges Jean. *L'écriture, mémoire des hommes*. Gallimard, Paris, 1987.

[14] James Kass. Code2000. Available on the Web: `http://home.att.net/~jameskass/`, 1998.

[15] Ole Lund. *Knowledge construction in typography: the case of legibility research and the legibility of sans serif typefaces*. PhD thesis, The University of Reading, Department of Typography & Graphic Communication, 1999.

[16] Microsoft Corporation. Arial Unicode MS. Available on the Web: `http://office.microsoft.com/downloads/2000/aruniupd.aspx`, 2000.

[17] Microsoft Corporation. OpenType specification. Available on the Web: `http://www.microsoft.com/typography/otspec/`, 2001.

[18] SIL International. Graphite. Available on the Web: `http://graphite.sil.org/`, 1997.

[19] Péter Szabó. Conversion of TEX fonts into Type 1 format. In *Proceedings of EuroTEX 2001 conference, Kerkade, September 24–27*, pages 192–206, 2001.

[20] Miles A. Tinker. *Legibility of Print*. Iowa State University Press, Ames, 1963.

[21] The Unicode Consortium. *The Unicode Standard Version 3.0*. Addison-Wesley, Reading, Massachusetts, 2000.

[22] Bror Zachrisson. *Studies in the legibility of printed text*. Almqvist & Wiksell, Stockholm, 1965.

Adapting Ω to OpenType Fonts

Anish Mehta
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
anish_mca@yahoo.com

Gábor Bella
Gabor.Bella@enst-bretagne.fr

Yannis Haralambous
Yannis.Haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

Abstract

Nowadays, TEX and its successors still use METAFONT or PostScript Type 1 as their primary font formats, while the "outside world" is moving on to OpenType, a likely candidate for *the* future font standard in computer-based typography. The project we are going to present in this article represents the first step of a long-term plan of adapting the full Ω system to OpenType. As a result of our work, the first version of a new, OpenType-compatible *odvips* is available. This article presents the basic concepts of our solution as well as our further development plans for the project.

Résumé

De nos jours, TEX et ses successeurs utilisent toujours METAFONT ou type 1 comme principaux formats de fonte alors que le «monde réel» se tourne de plus en plus vers OpenType, un candidat très probable pour être *le* format de fonte standard du futur. Le projet présenté dans cet article est la première étape vers une adaptation d'Ω à OpenType. Notre résultat est une nouvelle version d'*odvips*, compatible OpenType. Cet article présente les concepts de base de notre solution ainsi que nos projets de développement futurs.

Introduction

OpenType is a relatively new font format developed jointly by Adobe and Microsoft. The goal of the two companies was to replace the widely used PostScript and TrueType fonts with a single, backward compatible format that also provides better support for international scripts and advanced typography. In fact, OpenType is but a wrapper format that can either embed PostScript (CFF, Compact Font Format) or TrueType fonts, plus some additional tables that provide the extra features.

OpenType's possible advantages or drawbacks put aside, one must realise that, due to the strong technological and economic support of its creators, it is very likely to become a de facto digital font standard. Until now, the market's reaction to this new technology has been rather slow, mainly because of the amount of investment a new font format requires from foundries and application developers, not to mention customers. Nevertheless, newly designed fonts come out more and more often in OpenType format, and for non-Latin scripts OpenType's impact is even greater. So, if the TEX community would like to avoid isolation as far as new high-quality fonts are concerned, it is time to jump on the bandwagon and embrace the format as soon as possible.[1]

Our work aims to provide OpenType support for the Ω typesetting system. Among TEX-based systems, Ω is known for its support of non-Latin scripts such as Arabic, Hebrew or various Indic scripts. Since OpenType is the first format to offer proper typesetting for these scripts, implementing OpenType support in Ω is a logical and desirable step forward.

Design Principles

Ideally, *OpenType compatibility* should mean that OpenType fonts are recognised by every link in the document production chain, from the typesetting phase right up to printing. As of today, Ω, just like all TEX-based systems, needs to load a plethora of files at each step of the typesetting process: font metrics and kerning from OFM and OVF files, script-specific typographic features from ΩTP's (Ω Translation Processes), and finally the glyphs themselves from the actual font file, either in PostScript or bitmap format. With OpenType fonts, this would not be necessary: all information mentioned above could be directly retrieved from the OpenType font file itself. The virtual font concept could also be eliminated, as both

1. We are not the first to do this: pdfTEX supports OpenType.

Ω and OpenType are Unicode-compatible. Finally, the same OpenType font could be integrated, partially or in whole, into the final document to be sent to the printer.

However elegant and visionary (not to mention utopian) this scenario might be, it would certainly require a major rewrite of the whole Ω system, ΩTP handling included, as well as of the *odvips* utility. Support of OpenType's advanced typographic tables (GPOS, GSUB, etc.) as well as access to metric information would have to be implemented in Ω internally, thereby short-circuiting OFM and OVF files. Such a solution, however, would mean abandoning Knuth's basic approach, which separates metrics (information needed for text layout) from glyph shapes (only needed at the final step of typesetting). Although this approach was partly inspired by limitations of memory and processing power at the time TEX was designed, it could be argued that the TFM/OFM concept has the advantage of acting as a common interface to the layout engine, independently of the actual font technology used. It also has to be added that extraction of TEX- and Ω-compatible metric information from OpenType tables is far from a trivial operation, as it will be shown later in this article.

Ω's input processing will also need to be extended so that users can turn on or off certain OpenType features by hand. Most importantly, user control is crucial over discretionary features like glyph alternates.

The real problem, however, comes at the final processing step: the sad fact is that there are virtually no printers today with OpenType support built in. Whatever our document format, the OpenType fonts it uses will necessarily have to be converted into a format the printer will recognise, either by the printer driver (available only for Windows and Macintosh) or at the application level. TEX-based systems, and more precisely, *(o)dvips*, produce PostScript documents directly, without using any drivers. Consequently, the only possible solution is to convert OpenType fonts, right before inclusion in the PostScript document, into a common font format recognised by most printers.

In summary, four major areas of the Ω system need changes:

1. extension of Ω's control sequences to allow user control over OpenType features;
2. extraction of metric and kerning information from the OpenType font;
3. support for OpenType's advanced typographic tables, including GPOS, GSUB and BASE;
4. automatic conversion of OpenType fonts to a format supported by a vast majority of today's printers.

Due to the fact that full implementation of these functionalities will require a major rewrite of the whole Ω-*odvips* system, we have decided to approach the problem in three steps. As a response to the needs of the Ω user community, a basic but working solution is going to be published as soon as possible, providing access to OpenType fonts, even if the advanced feature support is far from complete and not as user-friendly as it should be. This transitional solution is still based on OFM and OVF files, and only includes GSUB support. As a second step, in the new Ω_2 system (coming soon to a TEX Live near you) metrics will be read directly from the OpenType font. Finally, full support of OpenType features as well as their corresponding input control sequences will be implemented in the long run.

The rest of the article deals with the improvements we have made so far to provide basic OpenType support. This includes metric extraction, GSUB support and font conversion and subsetting. In order to read the OpenType font and produce the necessary OFM, OTP, etc., files, we have developed a handful of Python utilities, based on Just van Rossum's great *ttx* program and *ttf2pt1* from Mark Heath et al. Also, *odvips* has been patched to use and subset OpenType-originated fonts.

Conversion of Metric and Kerning Information

In its present state, Ω needs font metric files (OFM) in order to typeset glyphs. As long as direct OpenType access from Ω is not possible, the corresponding OFM file will have to be created automatically during installation of the OpenType font. Whatever our input method, conversion of OpenType metric data into Ω-compatible values is not a trivial process: it is not always possible to find an equivalence relation between the two approaches.

Information found in a TFM/OFM file can be divided into the following three categories:

- global information;
- metric information on an individual glyph level;
- information on glyph pairs.

Global information corresponds to font-wide parameters including *checksum*, *design size* of the font, *character coding scheme*, *font family*, *font size*, *font face* and *font dimension* parameters.

Checksum information could be kept as a trace of the original OFM file, but obviously, there is no standard OpenType table to host it.

Design size is usually a reference to the point size at which glyph outlines and spacing display the best. Design size information can be obtained from OpenType's GPOS table via the `size` feature tag, which contains both the design size and a suggested range of sizes.

OpenType fonts use the Unicode encoding to allow handling of large glyph sets. *Font family* information is available in the *naming table* (`name`) with the help of the predefined `nameID` of 1. However, this information is not of much interest as it is not used consistently.

Global information also includes font dimension parameters such as *slant*, *space*, *stretch*/*shrink*, *xheight* and *extraspace*. Slant in OFM corresponds to the *italicAngle* entry in OpenType's `post` table, which gives the angle in counterclockwise degrees from the vertical, zero for upright text and negative for text leaning right.

The equivalent of TEX's global *space* parameter in OpenType can be taken as the width of the space glyph.

Stretch/shrink information are not available in standard OpenType tables.[2]

Information regarding *xheight* is available as the *sxHeight* field in the OpenType `OS/2` table. If it is not present then we can take the height of the non-ascending lowercase glyph 'x'. This metric specifies the approximate distance between the baseline and the height of non-ascending lowercase letters.

Finally, no equivalent of the *extraspace* parameter exists in standard OpenType tables.

Metric information on an individual glyph level in an OFM file corresponds to *width*, *height*, *depth* and *italic correction* parameters of individual glyphs. The *width* parameter corresponds to the *advanceWidth* value in OpenType's `hmtx` (horizontal metrics) table.

Height and *depth* information are not explicitly available in OpenType as there is some difference in the way TEX/Ω and OpenType deal with character dimensions. Figures 1–2 compare the two approaches, which we now describe further.

TEX and Ω have access to *abstract* height, depth and width parameters, in the sense that these values may be independent of the bounding box itself, of which they have no direct information. OpenType, on the other hand, provides bounding box, advance width and left and right sidebearing values so that both the width of the *abstract box* used for typesetting *and* the position of the glyph inside that box are known.

For OpenType fonts with TrueType outlines, the bounding box information is stored in the `glyf` table, while for CFF/OpenType, it needs to be extracted from the CFF data. Advance values and sidebearings come from either the `hmtx` or `vmtx` table, depending on the text direction. Nevertheless, the abstract height and depth parameters of TFM/OFM files are not present in OpenType in any form; vertical metric data such as advance height, top sidebearing, etc., are only used when typesetting vertically, they bear no relation to TEX's height and depth fields. Whether bounding box size parameters are acceptable as a substitute is also not evident. Nevertheless, this substitution has been used when adapting PostScript Type 1 fonts to TEX for many years already.

2. The JSTF table in OpenType contains prioritised suggestions where each suggestion defines the action that can be used to adjust a given line of text. When spaces are too large or too narrow, JSTF substitutions can help producing better lines. But this is by no means comparable to the global stretch and shrink parameters which give the maximum and minimum values of interword spaces.

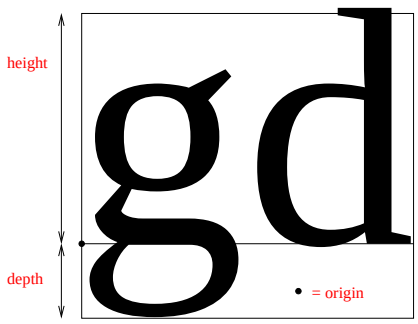


Fig. 1: TEX's notion of width, height and depth.

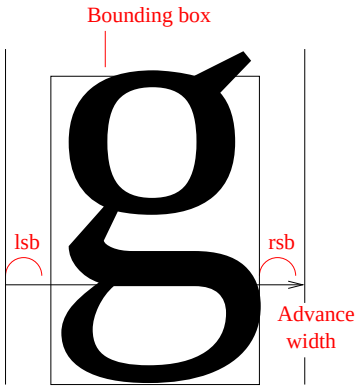


Fig. 2: OpenType's notion of *advance width* and *bounding box*.

Information regarding italic correction is not explicitly available in OpenType. However, it is possible to calculate it by subtracting the advance width value from *xMax*.

Information on glyph pairs refers to *ligatures* and *kerning*. Ligature substitution is handled by OpenType's `GSUB` table. It also supports contextual substitutions, which is similar to TEX's smart ligatures. Application of `GSUB` features will be discussed later in the article.

In OpenType, kerning information is available via both the `GPOS` and `kern` tables. In the former, a *Pair Positioning Subtable* specifies kerning data for glyph pairs, offering more flexibility than a typical kerning table: glyph pairs can be adjusted in both horizontal and vertical directions. It can also use the device table to adjust the positions of glyphs at each font and device resolution. The `kern` table, inherited from TrueType, provides a more classical approach to kerning; its use is discouraged.

OFM and OVF Creation

As explained above, OFM file creation for OpenType fonts is solely a temporary solution until direct OpenType access becomes available. Conversions described in the previous section are done by a Python utility called *makeovp*. The resulting OVP file can in turn be converted into OVF (virtual font) and OFM (virtual and real font metric) files by using the standard `ovp2ovf` utility.

In the virtual font, Unicode positions are mapped to glyph ID's according to the `cmap` table. Unencoded glyphs (swash alternates, contextual forms etc.) are automatically mapped to the private use area (PUA) of the Basic Multilingual Plane (U+E000...U+F8FF). The DVI file will use this "mixed" encoding scheme, by which *odvips* will later look up glyphs in the converted OpenType font (see section on PFC tables). It has to be stated again that this is only a temporary solution: our long-term plan is to eliminate the need for virtual fonts, as far as OpenType is concerned.

Advanced Typographic Table Support in Ω

This section presents how Ω makes use of OpenType's so-called *advanced typographic tables* such as `GSUB` and `GPOS`. As explained earlier, full support for these tables will certainly require a major rewrite of the Ω source code, which will most likely be a longer-term project. At the moment, only pair kerning information is used from the `GPOS` table (lookup type `2`). If no `GPOS` table is present in the font, the `kern` table will be used instead, but only for TrueType outlines: CFF/OpenType fonts can only use the former.

The `GSUB` table is supported to a much greater extent: all kinds of lookups are handled with the exception of *Extension Substitution*. This is a major breakthrough that makes available such OpenType features as contextual ligature substitutions for Indic scripts, old style forms or small capitals in Latin, etc.

As a first step, OpenType substitution lookups need to be converted into ΩTP's (Ω Translation Processes, [3]) by a Python utility named *makeotp*. This program will extract every feature and lookup type of every script from the OpenType font file given as input and generate, for each one of them, a separate OTP file.[3] The naming convention for these ΩTP files is the following:

```
FontName-FeatName-LkupType-ScriptTag.otp
```

`FontName` is the name of the font. `FeatName` is the name of the feature that the ΩTP provides; feature names conform to Microsoft's feature tag registry. `LkupType` is the type of lookup the given feature can perform. An example ΩTP filename using the `dlig` (discretionary ligatures) feature for the *Palatino* (`pala.ttf`) font is `pala-dlig-lkup4-latn.otp`. Inside the newly generated ΩTP file, one finds expressions such as:

```
@"0051 @"0075 => @"E010;
@"0073 @"0070 => @"E026;
@"0073 @"0074 => @"FB06;
```

where multiple glyphs are being replaced by a single ligature glyph. The hexadecimal codes represent Unicode character positions, coming from the `cmap` table. Here, the third expression replaces 'st' with a single ligature glyph 'ﬆ' (mapped onto the PUA). Behold, Dear Reader, as this is the world premiere of an Ω document using OpenType fonts, with `GSUB` features applied!

`pala-frac-lkup5-latn.otp` is another example of a `GSUB`-originated ΩTP performing contextual substitution. This ΩTP contains the expressions of the form:

```
@"0034 @"002F @"0035 => @"2074 @"2044 @"2085;
@"0034 @"002F @"0036 => @"2074 @"2044 @"2086;
@"0034 @"002F @"0037 => @"2074 @"2044 @"2087;
```

where the first one replaces '4/5' by '⁴⁄₅' through a contextual substitution: `four`, `slash` and `five` are replaced by `foursuperior`, `fraction` and `fiveinferior` respectively.

Using `GSUB` *Features in Ω Documents*

While converting an OpenType font file, the *makeotp* utility will automatically generate ΩTP's for the `GSUB` features present in the font. In order to access these features, one must include a *style* file corresponding to the script and font. For a *(font, script)* pair, the `Fontname-LangTag.sty` style file is generated automatically by *makeotp*. For example, for the *CourierStd* (`CourierStd.otf`) font and *Latin*, the file created will be `CourierStd-latn.sty`. At this moment, the style file does not turn on `GSUB` features automatically, it is up to users to activate the features they need. One includes the style file in the Ω document header, i.e.,

```
\usepackage{CourierStd-latn.sty}
```

Then, to apply a specific feature, include

```
\pushocplist\CourierStddliglatn
```

at the point where the `dlig` (discretionary ligatures) feature of the Latin script is to be activated.

Devanagari with Ω: A Case Study

This section explains how to use ΩTP's for typesetting in the *Devanagari* script (especially in the Hindi language).

The following two input methods may be used:

- Velthuis Transliteration Scheme;
- a Unicode compliant editor.

3. In some cases, this operation will result in dozens of files per font. Do not forget: this temporary solution has been conceived to bring some key OpenType support to Ω as quickly and easily as possible.

The Velthuis scheme needs two ΩTP's to convert the input transcription into Unicode: `velthuis2-unicode.otp` and `hindi-uni2cuni.otp`, used to deal with *virama* and dependent vowels. In the case of the Hindi language, the final *virama* is removed by this ΩTP.

Things are easier using a Unicode (UTF-8) compliant editor: Ω can read Unicode-based text directly. In this case, the two additional ΩTP's used for the transliteration scheme are not needed. Only these two lines need be added to the header of the Ω document:

```
\DefaultInputMode UTF8
\InputMode currentfile UTF8
```

To turn on GSUB features inside Ω, the following line needs to be included in the header, supposing that we are using the `raghu.ttf` font:

```
\usepackage{raghu-deva.sty}
```

The *makeotp* program is intelligent enough to assemble this style file in such a way that OpenType features are applied in the correct order. For reference, the correct order for the Devanagari script is:

- `nukt`-Nukta form.
- `akhn`-Akhand Ligature.
- `rphf`-Reph form.[4]
- `blwf`-Below-base form.
- `half`-Half-form (pre-base form).
- `vatu`-Vattu variants.
- `pres`-Pre-base substitution.
- `abvs`-Above-base substitution.
- `psts`-Post-base substitution.
- `haln`-Halant form substitution.

The example below has been created using the *Raghu* OpenType font, with all GSUB features enabled. Note that, due to the lack of support for GPOS features, glyph positions may not always be correct.

भारत और रूस ने महत्वपूण समझौते किये । भारत और रूस ने बुधवार को महत्वपूण समझौतों पर हस्ताक्षर किये हैं जिनमें आतंकवाद से मुकाबला करने का एक संयुक्त समझौता भी शामिल है ।

Font and Glyph Outline Conversion

The following sections of the article deal with outline conversion issues. As it has already been pointed out, there are virtually no printers today with built-in OpenType font support. OpenType fonts will thus in all cases be converted before printing, either by a printer driver or by the application itself, to an appropriate format. In our case, no printer driver is available (native PostScript being produced), so we had to come up with our own font conversion procedure.

Theoretically, several destination formats are possible: PostScript Type 1, CFF (Type 2), Type 3, and even Type 42, for TrueType-flavoured OpenType. Each of these formats has its advantages and drawbacks:

- Type 1 enjoys complete support by all kinds of PostScript printers, including Level 1 printers. Its disadvantage is that conversion of TrueType-based OpenType fonts into PostScript is not trivial at all, and converting TrueType hints seems to be especially difficult;

- CFF is the deluxe edition of Type 1; it would be an ideal solution for Type 2 charstrings in CFF-based OpenType (no conversion is needed), but for TrueType outlines we face the same problems as with the Type 1 format. In addition, CFF needs Level 3 printers.

- although Type 3 is also universally supported, its major problem is that it is not suitable for PDF: Acrobat may render it very poorly. The PDF format has little support for Type 3 fonts, so that their glyphs end up being displayed as bitmap images. This not only affects rendering quality, but also obstructs glyph-to-character mapping mechanisms, so that Type 3 rendered glyphs cannot be searched, copied or indexed as characters;

- Type 42 could be a possible candidate for TrueType-flavoured OpenType: OpenType to TrueType conversion should not be difficult,[5] and the resulting TrueType font can easily be embedded in the Type 42 wrapper and then sent to the printer directly. However, a Level 2 printer is needed to interpret the Type 42 format, and furthermore, we know little about Type 42 compatibility with PostScript font operators such as `glyphshow` and `charpath`. Failure of these PostScript operators to work would mean that PostScript code using Type 42 fonts is of low quality.

After careful consideration of the advantages and disadvantages of each format, the two most promising choices seem to be Type 1 and Type 42. As our final decision, we opted for Type 1, mainly because of its simplicity and absolute support by printers — but this choice does not necessarily mean that Type 42 conversion will not be added later. If we consider discarding TrueType

4. The `rphf` and `blwf` features should not be used simultaneously. Two macros (in the form of ocplists) will be created, one with the `rphf` feature and the other with `blwf`, so that the user can switch between the two forms when necessary.

5. Speaking here of converting outlines and other basic data necessary for rendering, not about OpenType's advanced typographic tables.

hints acceptable (it should not matter for printed documents), OpenType to Type 1 conversion becomes a feasible, not too painful process.[6]

The solution we were looking for had to be intelligent enough to meet two important requirements: first, the same OpenType font should not be converted every single time it is used in a document, but *only once*, when it is first used. Secondly, to decrease the number of the resulting Type 1 fonts (and thus the size of the PostScript document), glyphs not referenced in the DVI file (not used in the document) should not be included in the Type 1 fonts. In other words, *odvips* should be able to do *subsetting*. These two points are justified by the following arguments:

- If the OpenType font is large (CJKV fonts can occupy tens of megabytes), converting can take an excessive amount of time;
- Type 1 fonts can encode up to 256 glyphs, while for OpenType this limit is 65,536. A CJKV OpenType font containing 10,000 glyphs would need to be converted into no less than forty Type 1 fonts.
- the download time of large fonts to the printer is slow, and printer memory is filled up quickly;
- we should keep the size of the final PostScript document as small as possible (Internet downloads, etc.);
- even if the OpenType file is small, why do the same conversion over and over again?

The method we propose is to convert OpenType fonts only at the time of their first use, and to store the converted Type 1 outlines in an intermediate format called *PFC* (abbreviation of "PostScript Font Container", also a logical continuation of PFA and PFB). The PFC format can be regarded as a Type 1 glyph directory that provides one-by-one access to charstrings (as opposed to monolithic Type 1 fonts), allowing *odvips* to assemble Type 1 fonts on the fly, only including glyphs that are used in the given document (see fig. 3). If, say, a Chinese text contains only 500 different ideographs, *odvips* will retrieve the corresponding glyphs from the PFC font and create two Type 1 fonts (instead of forty!) to be included in the PostScript document. We will call these partial Type 1 fonts *mini-fonts*. The advantage of partial font creation is that both document sizes and printer overhead will be greatly reduced.

A Brief Description of the PFC format

We have chosen to design the PFC format to be compatible with the modular, table-based file structure of TrueType and OpenType (sometimes called *sfnt*)[7] so that all

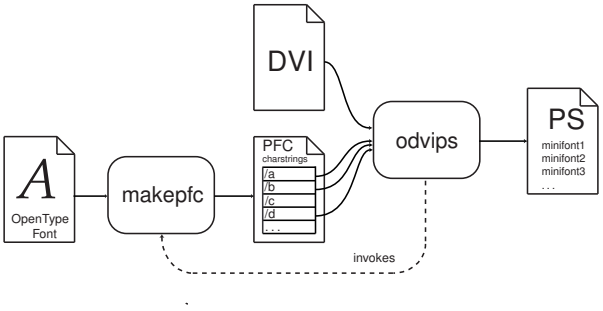


Fig. 3: Conversion and integration of OpenType font data into a PostScript document.

PFC data may be included, as tables, into the original OpenType font itself, instead of storing them in a separate file. However, if the font's licence does not allow modification of the font file, we have no choice but to create a stand-alone PFC file.

Our PFC format defines the following tables:

- the header and table directory entries comply with OpenType's header format, although the checksum is not used (at least for now);
- the `oMAP` table contains mapping information: for all glyphs in the PFC, their glyph code (the same code is used in the DVI file), the corresponding Unicode value, PostScript name, and the position of the glyph's charstring length in the `oCHR` table;
- the `oCHR` table contains the length of charstrings as well as the charstrings themselves, already encrypted with *charstring encryption*;[8]
- `oGFD` (Global Font Data) includes general information needed in Type 1 minifonts, such as italic angle and bounding box values;
- `oPRI` and `oSUB` contain the beginning of the *Private dictionary* and the subroutines of the Type 1 minifonts, respectively. At the moment, these tables are unstructured and their contents, including the entire subroutine set, are embedded into each minifont without change. In the future, we plan to replace this dumb temporary solution with intelligent subroutine handling (partial subroutine download).

It is important to point out that the PFC charstring directory can be generated on demand, at the first use of a given OpenType font. If *odvips*, while reading the DVI file, finds a reference to an OpenType font, it will first check whether the corresponding PFC file already exists. If it does not, the OpenType-to-PFC conversion starts automatically.

Since the two steps of OpenType-to-PFC conversion and assembly of minifonts are more or less independent,

6. According to unofficial statements of Adobe people in the OpenType discussion forum, even Adobe's "official" OpenType-ready applications like InDesign do an OpenType-to-Type1 conversion before printing.

7. See [1] for an overview of the OpenType file format.

8. See [2] for details.

there is no need to implement both inside *odvips*. In our solution, the converter is a separate utility called *makepfc* that is called by *odvips* when the font is used for the first time.

OpenType Outline Conversion

Our work[9] related to the *makepfc* utility revolves around the conversion process of TrueType quadratic splines to cubic Bézier curves of the PostScript Type 1 format. We have decided to build our work on Just van Rossum's *ttx* utility as the parser, mainly because of its ability to read OpenType tables, as well as its easy extendability to new functionalities. The conversion itself, as far as TrueType outlines are concerned, is based on the *ttf2pt1* program.

The conversion algorithm for TrueType fonts goes as follows: suppose we are given a TrueType (i.e., quadratic) curve whose starting and ending points are `startx` and `endx` respectively, with `ctrlx` as the single control point. To split the single control point into two, as needed for the Bézier cubic, the following calculations need to be done (fig. 4):

- starting and ending points, i.e., `startx` and `endx`, remain the same;
- the two control points are given by $(2 * \texttt{ctrlx} + \texttt{startx})/3$ and $(2 * \texttt{ctrlx} + \texttt{endx})/3$.

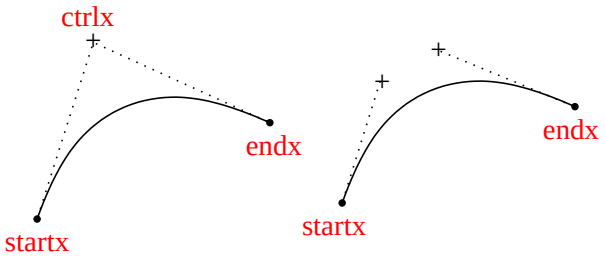


Fig. 4: Quadratic TrueType and cubic Bézier curves.

For CFF/OpenType fonts, no outline conversion is needed, except for Type 2-specific charstrings like `flex` or `random`, where conversion can be difficult: the `random` operator, for example, has no Type 1 equivalent at all. Most information, however, can be directly copied from OpenType's CFF table.

Configuration of odvips

To understand exactly how *odvips* deals with OpenType fonts, let us suppose that we are using `verdana.ttf` as our OpenType font, and that a DVI file using Verdana's original OpenType glyph ID's has already been created by Ω (either by means of a virtual font or by accessing OpenType tables directly). *odvips* will thus find a `verdana` font referenced inside the DVI, where `rverdana` is the TEX name (the name of the corresponding OFM file) of the OpenType font. It will therefore look through `psfonts.map`, trying to find an entry with the same fontname. The following line would be added to `psfonts.map`:

```
rverdana verdana <verdana.ttf
```

odvips will then realise that `verdana` is a TrueType-flavoured OpenType font and that it needs to find its PFC charstrings to be able to continue. First, it will try to find the PFC tables inside `verdana.ttf`. If there are no PFC tables in `verdana.ttf` (which is very probable as—due to Microsoft's restrictive licence—we are not allowed to modify the font) or if *odvips* cannot even find the file, it will look for a separate file called `verdana.pfc`, supposing that it must have already been created. If the PFC is not found either, *odvips* will invoke the *makepfc* program to produce the PFC tables on the fly. If both files are missing, *odvips* falls back to a default font.

A Final Remark

The process described above (including metric, style and OTP file creation, font conversion, modification of configuration files, etc.) may seem complicated. And, to be honest, it is, even if the individual steps are easy to carry out. We therefore plan to write a single script that would call the individual *make** and other programs automatically and thus let the user do the whole conversion process in a single step.

References

[1] The OpenType Specification v1.4. `http://www.microsoft.com/typography/otspec/default.htm`

[2] Adobe Type 1 Font Format. `http://partners.adobe.com/asn/developer/pdfs/tn/T1Format.pdf`

[3] Draft Documentation for the Ω system. 7 March 1998. `http://www.loria.fr/services/tex/moteurs/omega7mar1998.pdf`

[4] *Haralambous, Yannis, Plaice, John*: Low-level Devanagari Support for Omega—Adapting `devnag`. *TUGboat* 23(1), 2002, Proceedings of the TUG Annual Meeting, pp. 50–56. `http://omega.enstb.org/yannis/pdf/tug2002.pdf`

9. Anish Mehta (TTF) and Ghislain Putois (CFF) are the two developers of the *makepfc* utility.

Experience with OpenType Font Production

Sivan Toledo
School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
stoledo@tau.ac.il
http://www.tau.ac.il/~stoledo

Zvika Rosenberg
MasterFont Studio Rosenberg, 159 Yigal Alon Street, Tel-Aviv 67443, Israel
http://www.masterfont.co.il

Abstract

The article describes the production of about 200 Hebrew OpenType fonts with advanced typographic layout features. The production project was conducted for MasterFont, a large commercial font foundry in Israel, which produces and sells over 1400 Hebrew fonts. The production project had two main objectives: to convert Hebrew fonts with complex diacritic support from an older format to OpenType, and to streamline the font production process at the foundry. The article describes font technology background, the objectives and constraints of the project, the tools that are available, and the final conversion and production process. The article also describes our conclusions from the project, and in particular our conclusions concerning font production processes and tools, and the OpenType format.

Résumé

Cet article décrit la production d'environ 200 fontes OpenTypes hébraïques, munies de propriétés typographiques avancées. Ce projet a été mené par MasterFont, une importante fonderie israëlienne, qui produit et commercialise plus de 1 400 fontes hébraïques. Le projet avec deux objectifs principaux : la conversion de fontes hébraïques avec support de diacritiques complexes à partir d'un format de fonte plus d'ancien en OpenType, et l'automatisation du processus de production de fontes à la fonderie. L'article décrit les technologies de fontes impliquées, les objectifs et les contraintes du projet, les outils disponibles, et le processus de conversion et de production final. Il décrit également les conclusions que nous avons tirées du projet et en particulier nos conclusions concernant les processus et les outils de production de fonte, ainsi que concernant le format OpenType.

Introduction

This article describes the production of a large number of Hebrew OpenType fonts. More specifically, we describe the production of OpenType fonts with TrueType glyph descriptions and with advanced typographic layout features. The project was conducted by MasterFont Studio Rosenberg, a large Hebrew digital font foundry in Tel-Aviv, with assistance from Sivan Toledo from the School of Computer Science in Tel-Aviv University.

Hebrew is a right-to-left script that is used in one of two ways: with or without diacritics. Most Hebrew texts are printed almost without any diacritics, but children's books, poetry, and bibles are printed with diacritics. Even texts that are printed without diacritics use them occasionally to disambiguate pronounciation and meaning. The diacritics in children's books and poetry are vowels and consonant modifiers, which are called *nikud* in Hebrew (meaning "to add points"). Bibles also use a third kind, cantillation marks, which are not treated in this article (see [4, 5] for a thorough discussion). Hebrew diacritics are quite hard to support in fonts, since there are 15 of them and 27 letters, with up to 3 diacritics per letter. Even though not all the combinations are grammatically possible, there are too many combinations to support them conveniently using precomposed glyphs. In addition, diacritics must be positioned relative to the base letter visually, rather than mathematically. For example, the below-baseline diacritics must be set below the visual axis of the letter [15], not below its mathematical center. This means that ligatures, pair kerning, and simple mathematical centering alone are insufficient for correct placement of diacritics.

In addition to diacritics, Hebrew fonts can also benefit from pair kerning, although until recently, pair kerning was not particularly common in Hebrew fonts. For

further information about the Hebrew script, see [1, 14, 15].

The initial objective of the project was to convert older fonts into the new OpenType format. This was motivated by the concurrent development of Adobe InDesign 2.0 ME (Middle Eastern), the first high-end page-layout program to support Hebrew OpenType features. Microsoft's Office XP also supports these features, so InDesign was not the only motivator. But it quickly became evident that the same tools that would be necessary to convert the old fonts to the new format could also be used to streamline the entire font production process at MasterFont. Therefore, the project had two objectives: to convert the old fonts to the new format, and to streamline the production process.

The rest of the paper is organized as follows. The next section provides background on font formats. Section "Diacritic-Positioning" describes how diacritic positioning information is represented in MasterFont's old fonts. Section "Objectives" describes the objectives of the project, and "Tools" describes the software tools that we used to achieve these objectives. The overall production process is described in "The Production Process", and our conclusions from this development and production experience are described in "Conclusions".

Font Formats

This section describes the OpenType font format and other relevant font formats. Unfortunately, the term OpenType is somewhat vague, so this section also establishes a more precise nomenclature.

Type 1, TrueType, and Early Extensions The first widely-accepted device-independent font format was the PostScript Type 1 format [10]. Adobe started selling retail Type 1 fonts in 1986, and published the specification in the late 1980's, after Bitstream figured out how to produce Type 1 fonts (prior to that the specification was proprietary and only Adobe could produce Type 1 fonts). Type 1 fonts consist of two or three main parts: scalable hinted glyph descriptions, metric information, which includes glyph dimensions, advance widths, and pair kerning, and possibly bitmaps for screen previewing. Today most systems can rasterize the scalable glyph descriptions adequately, so the bitmaps are rarely necessary; Adobe distributes the Adobe Type Manager program without cost; it is a Type 1 rasterizer for older Windows and Macintosh computers without a built-in rasterizer. Type 1 fonts are relatively easy to design and produce, and today there are tens of thousands of commercial Type 1 fonts.

The next widely-accepted device-independent font format was the TrueType format [7, 2], which was invented by Apple in the late 1980's, as part of a collaboration between Apple and Microsoft, and became the native scalable font format for both Apple Macintosh computers and Windows computers. In a TrueType font, all the data associated with the font, such as glyph descriptions, metric information, administrative information (font name, for example), and possibly bitmaps, are placed in a single file, which is organized as a collection of tables. For example, one table contains glyph descriptions, another character-to-glyph mappings, yet another table contains kerning pairs, and so on.

The kinds of data in a TrueType font are almost identical to the kinds of data in a Type 1 font, but they are organized differently and sometimes represented differently. Most importantly, glyph descriptions and hints for low-resolution rasterization are represented differently. In a Type 1 font, glyphs are represented using cubic-spline outlines with declarative hints. In a TrueType font, glyphs are represented using quadratic-spline outlines and a low-level programmable hinting language. Type 1 outlines are easier to design and hint, and most designers use cubic splines when designing fonts. Well-hinted TrueType fonts are hard to produce, but they can achieve pixel-perfect rasterization at low resolutions, something that Type 1 fonts cannot achieve. Virtually all font editors can convert one type to the other. The conversion of TrueType outlines to Type 1 outlines is lossless, but the other direction only produces an approximation. However, the approximation of a Type 1 outline by a TrueType outline can be made arbitrarily accurate and font editors produce approximations that are visually indistinguishable from the original. Hinting information is usually also converted, but not as well.

Both Adobe and Apple introduced enhancements to these font formats during the 1990s, but none became widely used. Adobe introduced the Multiple Master font technology [11, 12], which allows users to interpolate between fonts in a family. This enhancement transforms the font family from a discrete set to a continuous spectrum. For example, there aren't just medium and bold fonts, but a continuous weight axis. Apple introduced GX fonts [7] (now called AAT fonts), which are TrueType fonts with additional tables that allow continuous interpolation, like Multiple Master fonts, and which enable high-end typographic features. In particular, GX fonts can support multiple glyphs for a single character, such as swash, small caps, or old-style figures, they can reorder glyphs, and so on. Neither format became widely accepted by font vendors, probably because the fonts were too hard to produce. No major font foundry besides Adobe produced Multiple Master fonts, and only a few GX fonts were produced by Bitstream and Linotype.

OpenType OpenType is a new font format [3] that Microsoft and Adobe specified collaboratively. The format was first used, under the name TrueType Open, in the Arabic version of Windows 95, which was introduced in 1996. Apple later joined the specification effort and the name was changed to OpenType.

OpenType adds three capabilities to the TrueType format: advanced layout features, support for Type 1 glyph descriptions, and digital signatures. The advanced layout features are supported by five new tables:

- GDEF, which defines glyph properties and glyph sets;
- GSUB, which defines glyph substitution rules;
- GPOS, which specifies positioning information;
- JSTF, which provides justification information; and
- BASE, which provides baseline-adjustment information.

These features were designed to support layout of text in complex scripts such as Arabic, Hebrew, Indic, and East Asian scripts (Hebrew is by no means the most complex), and to support high-end typographic features such as swash letters and decorative ligatures, old-syle and proportional figures, and small capitals.

The support for Type 1 glyph descriptions is designed to allow lossless conversion of Type 1 fonts to a TrueType-like table-based file format. Such fonts use two new tables, CFF and VORG, to store a losslessly-compressed Type 1 font. Technically, the format of the CFF table corresponds to that of PostScript Type 2 fonts [13], also known as the compact font format, but the essence of the format is a lossless representation of Type 1 glyph descriptions (outlines and hints). The need to support two kinds of glyph descriptions and hints constrained the design of the layout-related tables, in the sense that positioning information in the GPOS table could not be hinted using the normal TrueType or hinting language; a different hinting mechanism is used, which supports both Type 1 and TrueType glyphs.

The support for digitally signing fonts, which uses the DSIG table, is meant to allow users and font-using software to verify the authenticity of fonts. For example, future operating systems might require that certain system fonts are digitally signed by the operating-system's vendor, thereby preventing modified versions of these fonts from being used instead. For independent font vendors, there is currently little benefit in digitally signing fonts, since users currently do not have means to verify these signatures, and since fonts rarely pose security or stability problems to systems.

OpenType fonts with TrueType outlines can be used by any TrueType-capable software, since except for extra tables (which are allowed by the TrueType spefication), the fonts are also valid TrueType fonts. Software such as rasterizers, layout engines, and printer drivers can simply ignore the extra tables. OpenType fonts with Type 1 outlines require special support in rasterizers and printer drivers, support beyond that required to support Type 1 and TrueType fonts. In principle any TrueType-capable layout engine can use any OpenType font, since the metric and layout information are compatible (an exception is pair-kerning information, which can appear in either the 'kern' table or the GPOS table or both in a font with TrueType glyphs, but only in the GPOS table in a font with Type 1 glyphs). To emphasize the fact that OpenType fonts with TrueType glyphs can be used by any TrueType-capable software, whereas special support is required for fonts with Type 1 fonts, files containing the first kind use the `ttf` or `ttc` extension, whereas files containing the second kind use the `otf` extension.

So what is an OpenType font? According to the Microsoft/Adobe specification, any font that conforms to the specification, including both TrueType and Type 1 glyphs, with or without advanced layout tables and digital signatures. This is somewhat confusing, especially since the file icons in Windows do not correspond to the glyph descriptions (Windows 2000 and XP use the 'O' icon for `otf` files and for `ttf` or `ttc` files if the fonts have a DSIG table, and the 'TT' icon for all other `ttf` files). We will use the acronyms OTF for OpenType fonts with Type 1 glyphs, TTF-OTL for fonts with TrueType glyphs and with advanced layout tables, and TTF for fonts with TrueType glyphs but no advanced layout tables.

Microsoft's main interests in the OpenType format lies in supporting complex scripts in Windows, with initial emphasis on Arabic and later on Indic scripts, and in tighter operating system/font integration using digital signatures. Adobe's main interests in the format lies in exploiting TrueType's advantages, mainly the support for non-Latin character encodings and the convenient single-file format, and in providing its fonts and applications easier access to so-called expert glyphs, such as old-style figures and small capitals [9].

Windows and Macintosh Font Files Differences in the way font files are stored on different platforms cause a few additional difficulties during font production.

On Windows and Unix/Linux systems, a TrueType or OpenType font is stored in a single file. A Type 1 font is represented by up to four files: a `pfb` or `pfa` file that stores the glyph descriptions and a character-to-glyph encoding, `afm` and/or `pfm` files that store metric information, as well as the encoding, but no glyphs, and an `inf` file that stores administrative infomation about the font. On Windows system, only the `pfb` and `pfm` files are used.

On Macintosh systems, there are multiple ways to store fonts. The Macintosh operating system supports

two byte streams per file. One is called the data fork and the other the resource fork, which was meant to store application resources such as localized strings, icons, and so on. The format of the resource fork allows multiple resources to be stored in a single file. Before Mac OS X (that is, before version 10), fonts were always stored in the resource fork of files whose data fork was empty. Fonts and font families are represented by several kinds of resources, such as the FOND resource that describes an entire family (regular, bold, italic, etc.), SFNT for storing TrueType fonts, FONT and NFNT for storing bitmap fonts, and so on. The fonts of a family are usually stored in a single file called a *suitcase*, whose resource fork contains multiple resources. The Macintosh also associates a file type (separate from the extension) and a creator code with each file. Font files need to have a specific type to be recognized as such by the Macintosh's operating system.

Storing fonts in multiple resources within a single file, and storing the font data in the resource fork rather than the data fork, creates problems when fonts are moved between platforms. To address this issue, Apple introduced new ways to store fonts in Mac OS X. First, suitcases can now be placed in the data fork, using a format called a `dfont` file (after its extension). This format is essentially a traditional Mac font file, but in the data rather than the resource fork. Second, Mac OS X also allows Windows-style packaging of TrueType and OTF files in a single file in which the font data is stored in the data fork with no header or trailer.

Diacritic-Positioning Information in Older Fonts

MasterFont's existing fonts used three mechanisms for positioning diacritics. These mechanisms were developed over a long period of time — each additional mechanism was designed to correct deficiencies in previous ones.

The first mechanism uses a number of precomposed glyphs. These are used for *dagesh*, a dot that appears inside letters, and which must be carefully positioned to prevent overstriking the letter itself. Such combinations are also used for *shin*/*sin* dots, for diacritics attached to a final letter (only two combinations are used in modern Hebrew), and a few other cases. There are Unicode code points for these glyphs, as well as code points in the MacHebrew 8-bit encoding, and they are used by both Macintosh and Windows systems.

On the Macintosh, precomposed glyphs are also used to place diacritics on particularly difficult letter: *resh*, *dalet*, and *qof*. The first two are wide letters that have a single vertical stem at the right, under which below-baseline diacritics should be placed. Incorrect placement under these letters looks awful. The *qof* is the only non-final letter with a descender, so special attention is required to prevent the below-baseline diacritics from colliding with the descender. These glyphs do not have Unicode code points, but they are used in all the Mac's Hebrew fonts, including the fonts bundled by Apple with the operating system. These glyphs appear to have been added by Apple quite early on. They do fix the most serious placement cases, but they do not fix all the placement problems. Furthermore, this glyph set appears to have been designed by somebody without much expertise in Hebrew, since some of the precomposed glyphs can never appear in a text (e.g., *dalet* with a *hataf-patah*).

Since the precomposed glyphs are insufficient for correctly positioning all the combinations, MasterFont developed a few years ago a positioning aid for the Macintosh. This software, called *Mapik-Ve-Day*, is a Macintosh system extension that selects a diacritic glyph according to the base letter. Fonts designed to work with this software have three sets of diacritics: one for narrow letters, one for wide letters, and one for medium-width letters. The diacritics in all sets usually have the same shapes, and all have zero advance widths, but each set has different sidebearings. The three *qamats* glyphs, for example, all have the same shape and zero width, and all print to the right of the insertion point (so they appear below the preceding letter in a right-to-left text run), but the amount of right shifting is different. The software essentially divides the letters into three categories, each of which has its own set of diacritics.

These two mechanisms, the precomposed glyphs and the three sets of diacritics of different widths, essentially mimic hot-lead diacritics. They allow fairly good control over positioning, provided the font is designed with this system in mind. In particular, special attention must be given to the left sidebearings of base letters, to ensure that the diacritics from the appropriate category are well placed under them.

To allow more precise control over diacritic positioning, MasterFont began to use explicit positioning information in the fonts. These are stored in the kerning table of the font, even though they are not real kern pairs, since the insertion point should not move after the positioning adjustment. For example, a kerning pair *alef*-*qamats* with value 35 means that the *qamats* glyph must be shifted relative to the *alef* glyph by 35 font units. The insertion point remains exactly after (to the left of) the *alef*, as if no adjustment took place. This information is not used by the Macintosh operating system, but a some Adobe applications can use it.

MasterFont currently has 86 fonts with this level of diacritic positioing, which is essentially the set of fonts designed for book work. In addition, there are many more display fonts with pair kerning information but without elaborate diacritic support, a few of which are shown in Figure 6.

Objectives

Now that the setting has been described, we can enumerate the objectives of this project in more detail.

First and foremost, we wanted to convert the diacritic positioning information in the existing fonts to OpenType advanced-layout tables. We wanted to do this without specifying any additional positioning information. This restriction was motivated by several reasons:

- Efficiency; there was no point in doing extra design work on fonts that already produce perfectly positioned texts. For example, there is no point in specifying positioning information for a combination represented by a carefully designed-precomposed glyph.
- Correctness; by not specifying new positioning information, the chances of introducing new positioning bugs are minimized.
- Continued support for older applications; we wanted to be able to produce additional new fonts that would be able to work not only in OpenType applications, but also in older applications. By building new fonts according to the old specification and automatically converting them to OpenType, the foundry would be able to offer customers either an OpenType font or an old-syle font for legacy applications.

Second, we wanted to automatically produce fonts for both the Windows and Macintosh platforms (Windows fonts also work on Unix and Linux). More specifically, we wanted to produce the fonts for the two platforms using the same source fonts, rather than generate a Windows font and convert it, and then generate a Mac font and convert it. This meant that the glyph sets for Windows and Mac fonts had to be merged, since previously the foundry used different glyph sets for each platform. Furthermore, if possible, we wanted to have identical *finished* fonts, except for the suitcase wrappers. It turned out to be possible.

Two remaining choices had to be made, concerning the glyph-description format and the Mac-packaging format. After some deliberation, we decided to produce TTF-OTL fonts rather than OTF fonts. Producing OTF fonts has one advantage, namely that the original Type 1 outlines and hints remain intact, whereas automatic conversion is used when producing TTF fonts from the same source files. We felt that given the accuracy of the conversion, this had essentially no impact on high resolution output from printers or imagesetters. We also felt that modern on-screen rasterizers that use antialiasing (font smoothing) produce acceptable output from automatically-hinted and unhinted TrueType outlines. Therefore, we felt that OTF's advantage over TTF-OTL is not particularly significant. On the other hand, on pre-2000 Windows and pre-X Macs, OTF fonts require the Adobe Type Manager utility, which is free but nonetheless requires downloading and installing by the end user. Also, some applications, most importantly Microsoft's PowerPoint, cannot use OTF fonts even on systems that do support them (this is true for both PowerPoint 2000 and PowerPoint 2002). Finally, one of the tools we used in the production of the fonts, Microsoft's VOLT, provides much better support for TTF-OTL fonts than for OTF fonts; this is described more fully in the next section. Given the clear disadvantages and the only slight advantage of OTF fonts, we decided to produce TTF-OTL fonts.

Tools

The production of the fonts uses a number of software tools, which we describe in this section. We also describe alternatives to some of the tools and explain our choice of tools.

Assembly of the Advanced Typographic Tables The tool that we used to construct the advanced typographic tables in the fonts is Microsoft's Visual OpenType Layout Tool (VOLT). Given a font, we automatically construct input files for VOLT, and use it to construct a derivative font with appropriate GDEF, GPOS, and GSUB tables.

Adobe's OpenType Font Development Kit (FDK) can perform a similar function. However, when we started the project, the FDK had incomplete support for the GPOS table, which meant that we could not use it to convert our Hebrew fonts. The FDK is also OTF-oriented, in that it cannot accept TrueType glyphs as input, where as we wanted a TTF-to-TTF-OTL conversion, in order not to modify the glyphs or their hints. These restrictions appear to still hold.

In addition to VOLT, Microsoft also distributes command-line tools that can construct advanced typographic tables, but they are quite old and we did not experiment with them.

Versions 4.0 and 4.5 of FontLab can also produce OpenType fonts with advanced typographic tables. But our understanding has been that the OpenType support in FontLab is based on Adobe's FDK and hence does not support the required GPOS mechanisms, so we did not explore this approach.

Since we used VOLT, we would like to comment on its advantages and disadvantages. VOLT is a visual tool, designed to allow a font designer to specify visually glyph substitutions and positionings. Together with the sample fonts, it is a superb tool for understanding how OpenType layout features work and for prototyping and developing OpenType layout support. In addition to visual design, VOLT allows the user to export and import the structure of the layout tables or parts of the tables into text files

with special syntax. Since we had to convert many fonts, we opted for automatically generating these files, which are called VOLT project files (`vtp`) rather than visually constructing each font. On the other hand, VOLT does not have command-line options that can be used to open a font and a VOLT project file and ship an assembled TTF-OTL font. This means that even if the project files are constructed automatically en masse, they cannot be processed in batch mode — each font must be opened using a menu command, the corresponding project file must be imported using another menu command, and the output font must be shipped using yet another command.

Data Extraction and Project File Construction To produce the input file for VOLT, we had to convert the data in the initial TTF fonts into VOLT project files. The format of the VOLT project files is complex and undocumented, but quite simple to figure out by comparing the file to the font structure in the graphical user interface.

We performed the conversion using two custom-written programs. The first program, written in C, reads a TTF font and outputs an `afm`-like text file, which maps glyph indices to PostScript glyph names, and which documents all the kerning pairs (glyphs in TrueType fonts are stored in an array and are refered to by VOLT using their indices).

The second program, written in the AWK language, transforms this information into a VOLT project file. The transformation consists of:

- Mapping each glyph index to a glyph name and Unicode code point(s).
- Defining glyph groups.
- Defining glyph substitution lookups, for example to map normal diacritics to wide or narrow ones, to map glyph sequences to precomposed glyphs, and in one case to decompose a precomposed sequence depending on the preceding glyphs (this supports two meanings and vocalizations of the Unicode *vav+holam* sequence, which can stand for a single long vowel or a consonant followed by a short vowel; there is a single Unicode sequence for both cases, but they should be printed and vocalized differently).
- Defining diacritic positioning information.
- Defining kerning pairs.

We wrote the AWK program by developing a prototype font visually in VOLT, exporting the project file, and then writing the program so that given the input file, it generates a similar project file. We then loaded the resulting project file into VOLT and inspected the resulting OpenType structure. When we discovered problems, we modified the AWK program to correct them and so on. This methodology exploits the two strengths of VOLT: the ability to design and inspect OpenType layout tables visually, and the ability to accept automatically-generated input.

Minor Modifications of Other TrueType Tables We gradually discovered that other TrueType tables had to be modified to ensure that the fonts perform properly. Here are the main modifications that we perform on the fonts:

- We add code-page-support and Unicode-range information to the fonts. This information is used by Windows and Windows applications to determine which scripts a font supports.
- While we use VOLT to construct the Unicode character-to-glyph mapping table (a subtable of the 'cmap' table), we later add an 8-bit Hebrew encoding table to support older Mac applications. This encoding subtable must be added even if the original TrueType font has one, since VOLT regenerates the 'cmap' table, and to the best of our knowledge, cannot emit this particular subtable.
- We remove the 'kern' table, which contains diacritic positioning data that are not true kerning pairs, as described above.
- We add a 'gasp' table to specify that the fonts should to be antialiased at all sizes.
- We make minor modifications to the 'name' table, which contains copyright, version, URL, and other textual information for the end user.

Some of the modifications are done with a specially-written C program. The other modifications are performed by converting the appropriate table to XML format using Apple's `ftxdumperfuser` command-line tool for Mac OS X, modifying the XML file using AWK and SED programs, and then fusing the resulting XML-formatted table to the font file, again using `ftxdumperfuser`.

We perform some of the modifications using a C program and some using the Apple tool because when we started the project, the Apple Font Tools for Mac OS X had not yet been released. We therefore wrote a custom C program to perform the modifications we found necessary. As testing progressed and we discovered more required or desired modifications, we switched to using the Apple tool, which was easier than extending the C program. One problem with the XML format that Apple uses to describe font tables is that simple text processing tools like AWK and SED do not understand XML's hierarchical structure. But for relatively simple modifications, the combination of `ftxdumperfuser` and AWK or SED scripts works.

There is another tool that can perform TrueType-to-XML and XML-to-TrueType conversions, Just van Rossum's TTX.[1] We experimented with this tool, which

1. `http://www.letterror.com/code/ttx/`

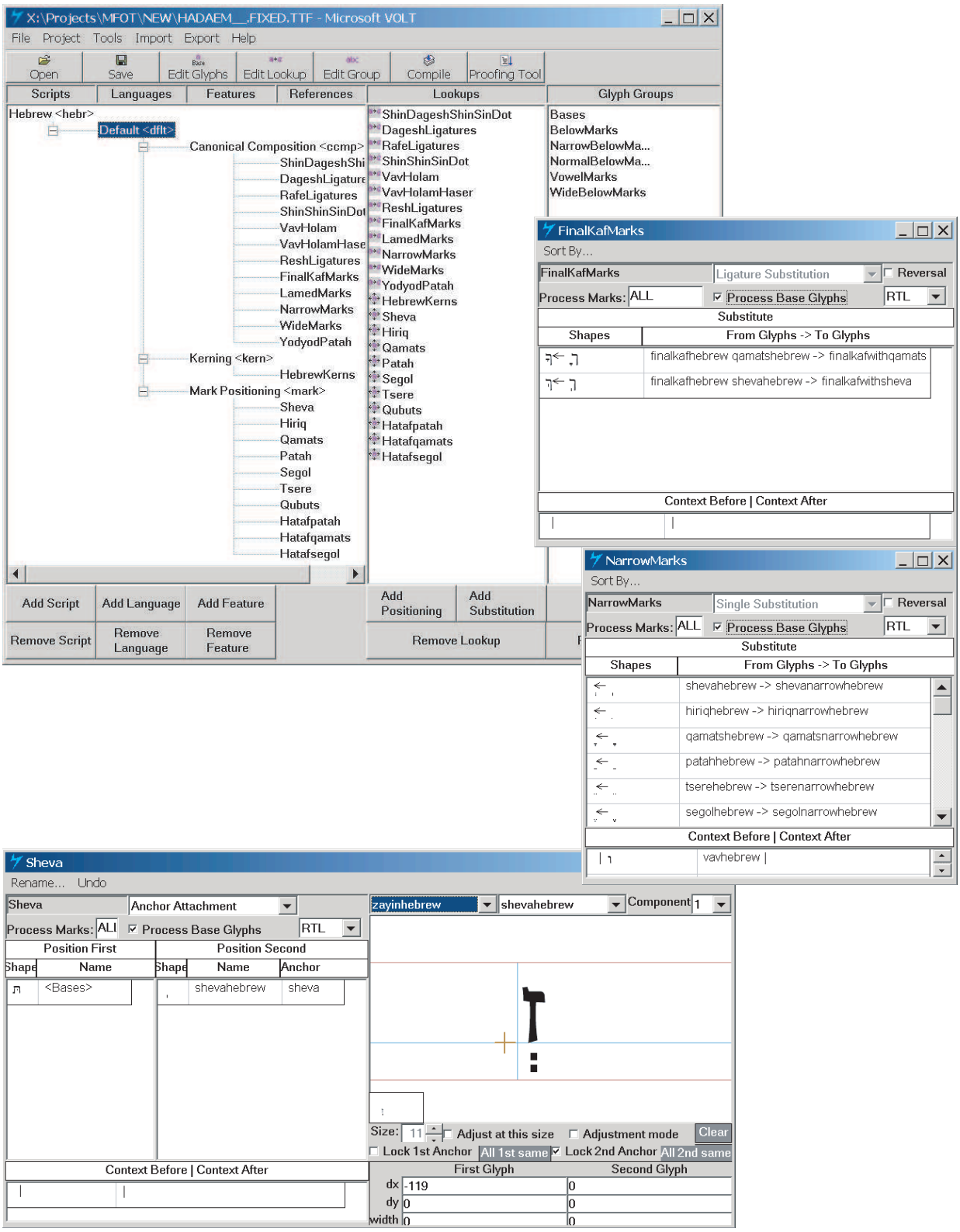


FIG. 1: One of the OpenType fonts in VOLT along with three types of OpenType lookups that we use to position diacritics: anchor positioning, single contextual substitution, and ligature substitution. Our fonts also use pair-kerning adjustments and more complex contextual substitutions.

is written in the Python language, but we could not install it successfully. The trouble was that TTX uses a number of other Python packages, for example, to perform numerical calculations, which did not install properly on our systems. But assuming that these issues in TTX are easily solvable, it is certainly an alternative to the Apple tools, and at least in principle, it is a multiplatform tool that runs on Windows, MacOS, and Linux.

General-Purpose Font Editors Although no general-purpose font editor was used for the actual conversion, we did use two font editors to produce the input font files and to inspect fonts that seemed to have problems. We used Fontographer to produce the input files to the conversion process, and we used PfaEdit to inspect fonts.

Macintosh Suitcase Generation We used the command-line program `ufond` from George Williams' FONDU[2] package to package TTF-OTL fonts into suitcases, and we used command-line tools from the Mac OS X developer kit to tag the suitcases with the required file type and creator code.

We also used a script to automatically scan an entire directory with TTF-OTL fonts and classify them into families. The classification is done using the family name field of the font. Once the classification is made, the script invokes `ufond` to package the fonts of the family into a suitcase.

The Production Process

The final production process consists of four stages. The *prevolt* stage is performed by a script that processes all the input TTF fonts in a given directory. For each input file, the script generates a modified TTF file and a VOLT project file. The modifications to the input files that we perform at this stage involve fixing the 'post' table to overcome an apparent Fontographer bug, upgrading the version of the 'OS/2' table and adding to it Unicode-range and code-page information, and removing the 'kern' table. The VOLT project file is produced by invoking the C program that generates an `afm`-like text file with glyph information and kerning and diacritic-positioning information, and converting that file to a VOLT project file using a custom AWK program.

In the second stage, each modified TTF file is opened in VOLT, the corresponding project file is imported, and the resulting font is shipped out. This requires only three menu commands in VOLT and clicking 'okay' a couple of times, but it is still the most time consuming stage of the production process.

The next stage, the *postvolt* stage, processes all the files shipped from VOLT in a given directory. A script invokes `ftxdumperfuser` a few times on each font, to add a MacHebrew encoding subtable, to fix the 'name' table, and to add a 'gasp' table. Although some of the processing can be moved from the *prevolt* to the *postvolt* or vice versa, some processing must be done before VOLT can process the font and some processing must be done after VOLT ships the font. Hence, both prevolt and postvolt stages are necessary.

The final *suitcases* stage again uses a script to process an entire directory. The script first extracts the family name from each font file. Next, the script finds

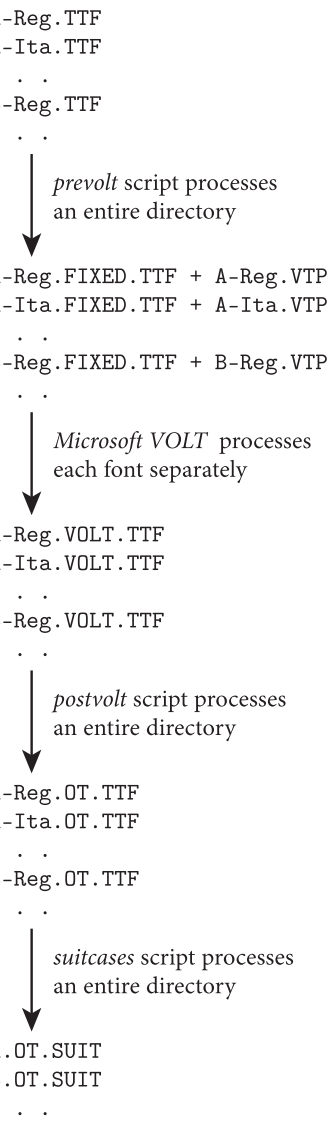


Fig. 2: An overview of the production process. The figures shows the input and output files for each stage and how that stage operates. The *postvolt* stage produces the final font files for Windows and Mac OS X, and the *suitcases* stage produces the final font files for any Mac operating system, X or older.

2. `http://fondu.sourceforge.net`

Ligature substitution:	שְׁלָךְ
Anchor positioning:	זְמָן
Contextual substitution:	וְגָם
	מָצוֹת vs. מִצוֹת
Insensitivity to mark orderings:	
gimel+dagesh+qamats	גָּל
gimel+qamats+dagesh	גָּל

Fig. 3: Hebrew diacritic positioning using OpenType features. The first three lines show how OpenType features utilize the mechanisms of older fonts. These examples correspond exactly to the examples shown in Figure 1. The fourth line shows a new application of contextual substitution that yields correctly-positioned glyphs for two identical Unicode sequences with different grammatical meanings and different vocalizations. The last two lines show that the OpenType features have been encoded in a way that prints all valid diacritic sequences in the same way, as mandated by Unicode. The sample was prepared using Adobe InDesign 2.0 ME.

בְּרֵאשִׁית, בָּרָא אֱלֹהִים, אֵת הַשָּׁמַיִם, וְאֵת
הָאָרֶץ. וְהָאָרֶץ, הָיְתָה תֹהוּ וָבֹהוּ, וְחֹשֶׁךְ, עַל פְּנֵי
תְהוֹם; וְרוּחַ אֱלֹהִים, מְרַחֶפֶת עַל פְּנֵי הַמָּיִם.
וַיֹּאמֶר אֱלֹהִים, יְהִי אוֹר; וַיְהִי אוֹר. וַיַּרְא אֱלֹהִים
אֶת הָאוֹר, כִּי טוֹב; וַיַּבְדֵּל אֱלֹהִים, בֵּין הָאוֹר וּבֵין
הַחֹשֶׁךְ. וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם, וְלַחֹשֶׁךְ קָרָא
לָיְלָה; וַיְהִי עֶרֶב וַיְהִי בֹקֶר, יוֹם אֶחָד.

Fig. 4: The first few phrases of Genesis, typeset in Adobe InDesign 2.0 ME using an OpenType version of Henri Fridlaender's *Hadassah* typeface.

בְּרֵאשִׁית, בָּרָא אֱלֹהִים, אֵת הַשָּׁמַיִם, וְאֵת הָאָרֶץ.
וְהָאָרֶץ, הָיְתָה תֹהוּ וָבֹהוּ, וְחֹשֶׁךְ, עַל פְּנֵי תְהוֹם;
וְרוּחַ אֱלֹהִים, מְרַחֶפֶת עַל פְּנֵי הַמָּיִם. וַיֹּאמֶר
אֱלֹהִים, יְהִי אוֹר; וַיְהִי אוֹר. וַיַּרְא אֱלֹהִים אֶת
הָאוֹר, כִּי טוֹב; וַיַּבְדֵּל אֱלֹהִים, בֵּין הָאוֹר וּבֵין
הַחֹשֶׁךְ. וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם, וְלַחֹשֶׁךְ קָרָא
לָיְלָה; וַיְהִי עֶרֶב וַיְהִי בֹקֶר, יוֹם אֶחָד.

Fig. 5: The first few phrases of Genesis again, typeset in Adobe InDesign 2.0 ME using an OpenType version of Zvi Narkiss's *Narkiss Classic* typeface. The typeface is a new version of *Narkiss Linotype*, which came out in 1968. The sample shows four weights: light, regular, medium, and bold.

all the font files that belong to that family, and invokes `ufond` on them to create a suitcase. The suitcase is created by `ufond` in the data fork of a file. We then move the font data to the resource fork of the font file, and tag it with appropriate type and creator codes. The automatic recognition of the font files that belong to each family minimizes the chances for errors that could occur if a family-configuration file was written by hand.

Due to the use of `ftxdumperfuser`, the *prevolt* and *postvolt* stages must be performed on a Mac OS X machine. The VOLT stage must be performed on a Windows machine. The *suitcases* stage is also performed under Mac OS X, in order to move the font data to the resource fork and in order to assign type and creator tags; non-Macintosh operating systems do not support these file-system features.

Conclusions

This project led us to several conclusions concerning OpenType and font production in general.

Reflections on Font Production Fonts are atypical objects that present special challenges to their manufacturers. The uniqueness of fonts is caused by a combination of factors that is not present in similar products.

First, fonts are highly complex data objects. While Type 1 fonts are fairly simple, TrueType, OpenType, and other advanced font formats are complex, with hundreds of individual fields that client software uses.

Second, fonts, especially those produced by independent foundries, are supposed to work with a variety of operating systems and software applications. Since different clients access different parts of font data, a font that tests perfectly with a given set of clients might fail on another client, as we have often discovered during the production process. To compound the problem, font-file specifications specify the format and intent of data fields, but they specify only loosely the behavior of client data. This is often done on purpose, to accommodate the behavior of older existing clients, but it makes it almost impossible to create working fonts without extensive testing. Let us give an example: are the glyph names in a TrueType or OpenType font significant, or are they present only to allow printer drivers to download the font into a printer in a consistent manner? Presumably, applications should use the encoding table in the 'cmap' table to find glyphs, not the PostScript names of the glyphs. But if some application uses glyph names instead, a font with variant names would not work in that application (we note that even Microsoft- and Apple-supplied fonts

use names for the Hebrew glyphs that are inconsistent with the Adobe Glyph List).

Third, font data, such as glyph outlines, hints, and metric information, has a very long life, but font files have a shorter life span. The outlines and metrics of widely-used fonts by foundries such as Linotype, Bitstream, or Agfa-Monotype are sometimes more than 20 years old, clearly an old age in the digital world. On the other hand, font files need to be recreated from this data once in a while. Font data that might have been used to create pre-PostScript fonts, were used again in the mid 1980s to create PostScript Type 3 fonts, then PostScript Type 1 fonts, then TrueType fonts. The TrueType font might have been upgraded once with better hints, then again to add the Euro symbol, then again to add non-Latin glyphs for supporting additional scripts. In between, special versions might have been produced for bundling with operating systems or software packages. In the non-Latin font market, special versions might have been produced to work with various layout programs. And today, the same font data is used to produce TTF-OTL and/or OTF fonts. Even a relatively young foundry like MasterFont has font data going back about 15 years.

Fourth, font foundries are typically small businesses or small business units [8]. This limits their ability to invest in custom programming to automate font production. This again is a particular problem in the non-Latin font market, where fonts are more specialized.

The longevity of font data from which new font files must be occasionally produced, together with the fact that foundries easily accumulate hundreds or thousands of fonts, implies that automation should be widely employed in font production. That is, producing a new font by launching a font editor, loading an older version, making the required changes, and generating a new font, is inefficient when a large number of fonts must be produced. Manual production is also prone to errors and hence requires more testing than an automatic conversion that processes all the fonts in exactly the same way. However, the size of most independent foundries precludes large investments in custom automation (that is, paying programmers to automate font production).

This problem has been addressed recently in three ways:

- Microsoft, Adobe, and Apple, who need to encourage font production, release free font-production tools (Adobe's interest in independent font production stems from its role as a vendor of application, not from its role as a font foundry). Microsoft has released VOLT and a large number of other tools for hinting, adding digital signatures, editing strings in a font, and so on. Adobe has released the FDK, and Apple has released a number of font tools. To a large extent, the three companies release the tools that they use themselves to produce fonts. Still, the effort in publicly releasing and supporting these tools is significant, and the wide availability of these tools does help independent foundries. In some cases the tools are not ideal for automated production. For example, the inability of VOLT to process a font without invoking the graphical user interface slows down production.
- General-purpose font editors now include scripting languages. Specifically, FontLab uses Python and PfaEdit its own scripting language. These tools will reduce the cost of automation and hence of font production, especially if scripts can be applied to a font from the command line (this is true for PfaEdit, and perhaps also for FontLab). Obviously, this approach requires programming, but at least the programming environment is the familiar font editor. One remaining problem in this approach is that these font editors read the font data into their own in-memory data structures and then generate a new font, which can have side effects on tables that are not supposed to be modified by a script. FontLab is careful not to modify data it does not need to (for example, not to modify TrueType glyphs unless glyphs are edited), but PfaEdit regenerates TrueType glyphs whenever it outputs a font file.
- Individuals produce powerful free tools that can assist in automating font production. These include TTX and PfaEdit. One problem with using these tools in commercial font production is that they tend to be less stable. One might expect that they would not be as well supported as commercial tools, but that is not always true: George Williams, for example, the creator of PfaEdit, provides excellent and prompt support.

Font production requires careful management over time of source files, deliverables, and software. Source files are used to produce fonts over many years, and they need to be updated by adding glyphs, metric infomation, and so on. When the source files are updates or when deliverables in new formats are needed, deliverables are produced again. It is beneficial to automate new production cycles as much as possible, especially when many fonts are involved. Since foundries often cannot invest significant funds in custom programming, standard font-production tools must support automation and reproduction. The scripting support in FontLab is a step in this direction, and essentially any command-line tool is usable in script-driven batch processing. But other tools, like VOLT, do not support automated production and reproduction. The importance of automation and the requirements of multiple production cycles may be evident to large font foundries, but are usually not evident to smaller foundries.

Reflections on OpenType Will the OpenType format, and in particular the advanced typographic layout, succeed, or will it fail like Multiple Master fonts and GX fonts? This question is important for font vendors, application developers, and font consumers, who all need to decide whether or not to invest in OpenType. There are several reasons to believe that OpenType layout will become successful and widespead.

First, both Adobe and Microsoft are pushing this technology. In particular, both companies provide font producers with free production tools, Adobe has converted almost its entire typeface library to OpenType (which was not the case with Multiple Masters), and Microsoft has provided application developers with both tools to utilize OpenType layout features (through the Uniscribe and OTLS libraries) and with assistance with developing layout engines. Both companies already produce major applications that exploit OpenType's layout features, namely Office XP and InDesign. Other companies are also developing OpenType layout engines, such as IBM's ICU open-source engine.

Second, the technology is essential for presenting text in some scripts, such as Indic scripts, and essential for presenting Arabic and Hebrew text with diacritics (Arabic and Hebrew without diacritics can be set without OpenType layout features). In particular, Microsoft has used OpenType layout in its Arabic fonts and operating systems for years now.

Third, the OpenType format has been designed to make font design easy, which was not the case for Multiple Masters and GX. The structure of OpenType's layout support is declarative. For example, the font designer can declare that a certain glyph substitution only occurs in a certain glyph context (preceding and following glyphs). The algorithmic question of how to recognize the context is left to the layout engine. In contrast, GX's structure is procedural/programmable, and the font designer must specify a finite-state machine that the layout engine uses to detect the context. In general, declarative font technologies like outline fonts and Type 1 hinting have been more successful than programmable technologies like TrueType hinting, GX state machines, or METAFONT. (TrueType hinting is unsuccessful in the sense that the cost of hinting causes most fonts to remain unhinted or automatically hinted.) The obvious reason is that most font designers are graphic designers by training, rather than programmers.

One must acknowledge, however, a few problems related to OpenType layout. First, the main benefit that OpenType brings to the Latin-script market are alternate glyphs, especially small caps, old-style figures, proportional figures, and swashes. But these glyphs have been available for years, although in separate small-caps, old-style-figures, and expert fonts. Using these separate fonts is not much harder than using OpenType layout features: instead of marking a run of text or a character style with OpenType features, the document designer or graphic designer marks the text or style with an alternate font. This makes it a bit harder to switch font families, but the difference is not that dramatic.

Second, a single OpenType font may behave in different ways in different applications, which may be confusing or disappointing to users. This happens due to two reasons: (1) The OpenType specification does not specify formally and in detail how a layout engine should behave, and (2) the desire to allow application developers to increase the level of OpenType support gradually. Microsoft published some specifications regarding layout engines, but these specifications basically specify only the minimal actions that a layout engine must perform to support OpenType fonts in a particular script. Many other aspects of layout-engine behavior are not spelled out explicitly. The desire to allow application developers to add OpenType support gradually means that some applications can rasterize OpenType fonts but completely ignore the advanced layout tables, other applications support the advanced layout tables but give users access to only a few layout features, and so on. For example, Microsoft Office applications appear to ignore the layout tables in fonts with CFF outlines, and both Adobe and Microsoft application seem to ignore the association between layout features and specific scripts and languages.

OpenType and TEX The aim of the project described in this article was to produce OpenType fonts for existing OpenType-capable applications, such as Microsoft Word and Adobe InDesign. Getting these fonts (or any other OpenType fonts) to work with TEX or Omega was not one of the project's goals. Still, how would one use OpenType fonts like these in TEX or Omega?

The familiarity with OpenType's advanced layout features that we gained from this project leads us to think that TEX's mechanisms are too weak to support many useful OpenType layout features. Several factors restrict TEX from exploiting OpenType. First, most OpenType fonts represent more than 256 characters and have more than 256 glyphs, so the 8-bit character/glyph restriction in TEX is restrictive. Second, many OpenType layout features, such as mark-to-mark diacritic positioning, mark-to-base diacritic positioning, and contextual substitution simply cannot be represented by TEX's ligature and kerning mechanisms. Third, OpenType's justification mechanisms appear to be incompatible with TEX's assumptions about boxes and words. OpenType fonts can suggest glyph substitutions and positioning adjustments to be done when text needs to be expanded or condensed to justify a line. The adjustments can include changing the sidebearings of glyphs (tracking), replacing glyphs by

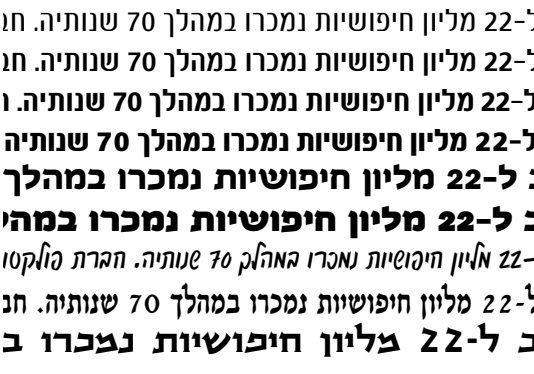


Fig. 6: A few display OpenType fonts produced by MasterFont using the method described in the paper. Display fonts do not have extensive diacritic-positioning support. The first four lines show *Rosenbert Naot*, the next two *Taxi*, and the 7th line shows *Hafgana*, all by Zvika Rosenberg. The 8th line shows *Daphna* by Ada Yardeni, and the 9th shows *Avney-Gad Hakuk* by Gad Ullman.

wider/narrower variants, forming or breaking ligatures, and so on. TEX's paragraph-breaking algorithm assumes that words have a fixed width, but this is not necessarily true in OpenType fonts: words can have several possible widths that the layout engine can choose from.

At least the first two issues should be addressed in order to use OpenType fonts in an effective way in a TEX system, at least in complex non-latin scripts. Omega [6] already addresses the first issue, so it seems to be the ideal candidate for OpenType support. It appears that the Omega team is already working on the second issue, and we hope that it will get resolved. We believe that it would be best to support OpenType in Omega directly, with no intermediate metric files (i.e., without `ofm` files), but that is something for the Omega team to decide. We plan to explore these issues ourselves at Tel-Aviv University using NTS, the new implementation of TEX in Java.

Acknowledgements Thanks to an anonymous referee for helpful suggestions and corrections. Thanks to Microsoft's Paul Nelson for answering numerous OpenType-related questions on the VOLT-community's message board, on the OpenType mailing list, and via private email exchanges. Thanks to WinSoft's Pascal Rubini for answering questions regarding the behavior of InDesign 2.0's Hebrew layout engine, and regarding the Hebrew-diacritic-handling mechanisms of older applications. Thanks to George Williams for making PfaEdit (now named FontForge) available. Some of the historical information concerning font formats was gathered from replies to a question that Adam Twardoch posted on the OpenType mailing list.

References

[1] The Unicode Consortium. *The Unicode Standard Version* 3.0, 2000. Parts of the standard are available online at `http://www.unicode.org`.

[2] Microsoft Corporation. *TrueType* 1.0 *Font Files, Technical Specification Revision* 1.66, 1995. `http://www.microsoft.com/typography`.

[3] Microsoft Corporation. *OpenType Specifications Version* 1.4, 2002. `http://www.microsoft.com/typography`.

[4] Yannis Haralambous. Typesetting the Holy Bible in Hebrew, with TEX. *TUGboat*, 15(3):174–191, 1994. Also appeared in the *Proceedings of EuroTEX* 1994, Gdańsk.

[5] Yannis Haralambous. Tiqwah: a typesetting system for biblical Hebrew, based on TEX. *Bible et Informatique*, 4:445–470, 1995.

[6] Yannis Haralambous and John Plaice. Omega, a TEX extension including Unicode and featuring lex-like filter processes. In *Proceedings of the* 1994 *EuroTEX Conference*, Gdańsk, 1994.

[7] Apple Computer Inc. *TrueType Reference Manual*, 1999. `http://fonts.apple.com`.

[8] Emily King. *New Faces: Type Design in the First Decade of Device-Independent Digital Typesetting*. PhD thesis, Kingston University, 1999. `http://www.typotheque.com` under 'Articles'.

[9] Thomas W. Phinney. TrueType, PostScript Type 1, & OpenType: What's the difference? `http://www.font.to/downloads/TT_PS_OT.pdf`, 2001.

[10] Adobe Systems. *Adobe Type* 1 *Font Format*, 1990. `http://partners.adobe.com/asn/developer/technotes/main.html`.

[11] Adobe Systems. *Type* 1 *Font Format Supplement*, 1994. Technical Specification #5015, `http://partners.adobe.com/asn/developer/technotes/main.html`.

[12] Adobe Systems. *Designing Multiple Master Typefaces*, 1997. Technical Specification #5091, `http://partners.adobe.com/asn/developer/technotes/main.html`.

[13] Adobe Systems. *The Type* 2 *Charstring Format*, 2000. Technical Specification #5177, `http://partners.adobe.com/asn/developer/technotes/main.html`.

[14] Sivan Toledo. A simple technique for typesetting Hebrew with vowel points. *TUGboat*, 20(1):15–19, 1999.

[15] Ada Yardeni. *The Book of the Hebrew Script: History, Palaeography, Script Styles, Calligraphy and Design*. Carta, Jerusalem, 1997.

The METATYPE Project: Creating TrueType Fonts Based on METAFONT

Serge Vakulenko
Cronyx Engineering, Moscow, Russia
vak@cronyx.ru
http://www.vak.ru/proj/metatype

Abstract

The purpose of the METATYPE project is the development of freeware TrueType fonts for the general user community. The METAFONT language was chosen for creating character glyphs. Currently, glyph images are converted to cubic outlines using a bitmap tracing algorithm, and then to conic outlines.

Two font families are currently under development as part of the METATYPE project: the TeX font family, based on Computer Modern fonts by Donald E. Knuth, retaining their look and feel. A rich set of typefaces is available, including Roman, Sans Serif, Monowidth and Math faces with Normal, Bold, Italic, Bold Italic, Narrow and Wide variations; and second, the Maestro font family, which is designed as a Times look-alike. It also uses Computer Modern sources, but with numerous modifications.

Résumé

Le but du projet METATYPE est le développement de fontes TrueType libres pour la communauté générale d'utilisateurs. Le langage de programmation METAFONT est utilisé pour créer les glyphes. Actuellement, les glyphes sont convertis d'abord en contours cubiques, en utilisant un algorithme d'auto-traçage de bitmap, et ensuite en contours coniques.

Deux familles de fontes sont actuellement en développement, dans le cadre du projet METATYPE : la famille «TeX font», basée sur les fontes Computer Modern de Donald E. Knuth, et proposant du romain, du sans empattements, du mono-chasse, et les fontes mathématiques, dans les styles : régulier, gras, italique, gras italique, étroitisé et élargi ; la famille «Maestro», qui est un clone du Times. Cette dernière utilise également le code de Computer Modern avec un grand nombre de modifications.

Why TrueType?

The world is getting smaller. We can see how the Internet and globalization cause people of different cultures and races to come into contact with each other more and more. The world is thus becoming more multilingual, something the computer world has known for some time: the first draft of the Unicode standard is dated 1989 [1].

At the same time, free software is gaining in popularity. Linux is more and more widespread, and it needs fonts; lots of fonts; a variety of good fonts with Unicode support. We already have a standard format in TrueType. (In a super-format, combined with Type 1, it is called OpenType [2].)

The TrueType format has several advantages:

- It is an open standard; the specification is available online free-of-charge.
- High-quality (commercial) fonts are abundant.
- There exists a free implementation named FreeType [3], of very high quality.
- The XFree86 graphic environment fully supports TrueType fonts, including antialiasing.

The author is bold enough to assert that Unicode and TrueType are the future. "Young" programming languages, such as Java and C#, use Unicode for the basic representation of text strings. Visionary operating systems, such as Windows NT, support Unicode at the kernel level. Word processors use Unicode for storage of documents (XML, RTF and other formats). The number of web sites with Unicode encoding grows every day. Cellular telephones use Unicode for web browsing (WAP). The world is slowly moving toward Unicode. And TrueType seems to be a natural result of font technology development.

The problem is that currently available systems for font development do not support Unicode and/or TrueType, or are not free. The METATYPE project is intended to fill this gap.

METAFONT as a glyph source language

METAFONT was chosen as the method for glyph development for the following reasons:

- It is a powerful, advanced language for glyph representation, and importantly, it is well documented.
- Glyph parametrization provides a straightforward means of creating entire families of fonts: normal-bold, normal-slanted, narrow-wide, serif-sans serif, etc.
- There are a number of very high-quality fonts (Computer Modern, AMS, etc.), whose sources can be studied.

It is also important that METAFONT is implemented on the majority of modern platforms and is available free of charge.

For the METATYPE project, an existing METAFONT/Web2C implementation with no additional modifications was used.

What are the problems?

So, we have METAFONT, and an implementation of METAFONT in Web2C [4]. What prevents us from taking, say, the source for Computer Modern [5] and transforming it easily to TrueType format? We see several problems here.

Problem 1: 16-bit encoding. METAFONT supports only an 8-bit encoding of symbols. But we need 16-bit Unicode encoding. The situation is aggravated because Computer Modern, as well as the majority of other METAFONT fonts, is optimized for use with TEX. Thus, they have non-standard encodings, which, moreover, is different for different fonts of the family.

For the transition to Unicode, the decision was made to store every glyph source in a separate file. Glyphs are grouped by 16 symbols into subblocks, and subblocks are grouped into blocks. Each block NN contains up to 256 symbols in the range NN00-NNFF.

Definitions common to all glyphs of a font are collected in a separate file, `base.mf`. For certain subsets of symbols there may be separate definition files, for example, `cyrbase.mf` for Cyrillic and `greekbase.mf` for Greek characters. Parameters for various styles are located in separate `param.mf` files.

The existing Computer Modern source was manually processed and split into two fonts, which the author has named "TeX" [6] (381 symbols) and "TeX Math" [7] (104 symbols).

Problem 2: splines. For TrueType, glyphs must be represented as contours consisting of splines. The best solution would be to derive splines directly from METAFONT. Another possibility is to use METAPOST — the implementation of METAFONT with PostScript output — since it is possible to extract the required splines from PostScript output. There are several utilities that use this method to generate PostScript Type 3 fonts, for example mf2pt3 [8].

The first difficulty here is that the contour generated by METAPOST must be converted to "canonical form", i.e., without self-crossings. This is a separate interesting and difficult mathematical task, the solution for which must be a task for later. For now we use a simpler method — generating a contour by tracing the glyph raster image.

Autotrace [9] is used to transform the raster image of a glyph into a contour. Autotrace has been extended to allow direct input of files in GF format.

The second difficulty lies in the transformation of splines. Autotrace generates cubic splines (third-order Bézier curves), while TrueType requires conic splines (second-order Bézier curves). The solution to this mathematical problem is to cut a cubic spline into two "close enough" conic splines. Thus, another new feature was added to Autotrace to convert traced contours to conic splines and to output a special UGS (Unicode Glyph Source) format.

Problem 3: hints. For use on low resolution devices such as monitors, TrueType fonts are equipped with so-called *hints*. Hints are programs written in the pseudo-code of a special interpreter, which for every symbol define some changes of the shape of a symbol to enhance glyph rasterization. Creating good hints is a very hard manual task.

Hints are not currently implemented in METATYPE. Support for hints will likely be added in the future.

Some modern rasterizers, such as FreeType 2 [10], do not use hints (because of patent problems [11]). Instead, they use an "auto-hinting" technology. In this case real hints are not necessary.

Problem 4: low resolution bitmap glyphs. You can also use raster glyphs for low-resolution environments like monitors, embedding the raster glyphs into the TrueType font in addition to the contour. Such rasters can be created by means of METAFONT, commonly in 13, 14, 15, 16, 17, 18, 20 and 23 pixels. Thus the most frequently used point sizes are covered (see table 1).

METAFONT generates rasters in GF format. To convert them to UGS format, a special utility `gf2ugs` was developed.

Problem 5: building TTF files. A TrueType file has a very complex structure. Fortunately, there is a Python library (Fonttools [12]) that makes it possible to work with TTF files. All complexities of the organization of a font file are hidden. To build a font, it is enough to make a dozen XML data files, and then to call an appropriate library procedure.

Font height	METATYPE font size	96 dpi — X Windows	100 dpi — MS Windows, Small fonts	120 dpi — MS Windows, Large fonts
13 pixels	96gf	10 pt	9 pt	8 pt
14 pixels	102gf		10 pt	
15 pixels	108gf	11 pt	11 pt	9 pt
16 pixels	114gf	12 pt		
17 pixels	120gf	13 pt	12 pt	10 pt
18 pixels	132gf	14 pt	13 pt	11 pt
20 pixels	144gf	15 pt	14 pt	12 pt
23 pixels	168gf	17 pt	17 pt	14 pt

Table 1: Bitmap font height in pixels and covered point sizes

How it all works

The source code for every glyph is stored in a separate file in a METAFONT format. The scheme of glyph compilation is shown in figure 1. As a result, the file in UGS format is created, containing a contour and rasters for several different pixel sizes.

When UGS files for all symbols are ready, the TTF and BDF fonts are assembled (shown in figure 1).

This work is carried out by the Python scripts `mk_db.py`, `mk_ttx.py` and `mk_bdf.py`. The `mk_db.py` script collects all UGS glyphs into a single DBM database to speed up working with the data. The `mk_ttx.py` script builds XML font data files and, using the Fonttools library, creates a TTF file. Lastly, the `mk_bdf.py` script transforms raster data to the BDF format for use under the X Window System.

UGS file format

For intermediate storage of the glyph data, a special format named UGS (Unicode Glyph Source) was developed. It is an ASCII-encoded text file where each line contains a single statement. An example of a UGS file for the symbol `FULL_STOP` is given below:

```
symbol 0x2e design-size 2048
      advance-width 569
      contour
            path
                  dot-on 273 216
                  dot-off 258 213
                  dot-on 245 209
                  dot-off 231 203
                  ...
                  dot-on 273 216
            end path
      end contour
      left-bearing 176
      right-bearing 177
      ascend 218
      descend -1
end symbol

symbol 0x2e design-size 33
      advance-width 9
      bitmap width 4 height 4
             . * * .
             * * * *
             * * * *
             . * * .
      end bitmap
      left-bearing 2
      right-bearing 6
      ascend 4
      descend 0
end symbol
```

Installing METATYPE

The METATYPE distribution can be downloaded from SourceForge [14]. The current version is available by CVS [15].

METATYPE depends on the following 6 items:

1. Python 2.2.2 [16]
2. Fonttools 2.0b1 [17], with PyXML 0.8.2 [18], and Numeric Python 23.0 [19]
3. Netpbm 10.15 [20], with libpng 1.2.5 [21]
4. Freetype 2.1.4 [22]
5. Web2C 7.3.1, including METAFONT 2.7182 and METAPOST 0.641 (we used teTEX 1.0.7 [23])
6. Autotrace 0.31.1 [24], patched

Before installing Autotrace, you must apply the patch `autotrace.pch` to it:

1. Unpack Autotrace 0.31.1
2. Enter directory autotrace-0.31.1
3. Apply patch:

   ```
   patch -p1 < autotrace.pch
   ```

4. Execute `automake`
5. Run script `./configure`
6. Execute `make` and `make install`.

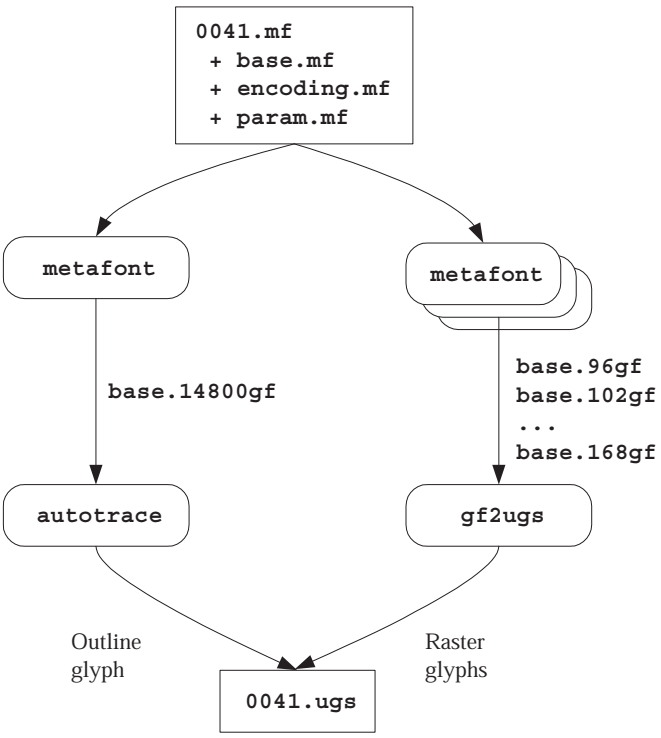


Fig. 1: Translating MF to UGS

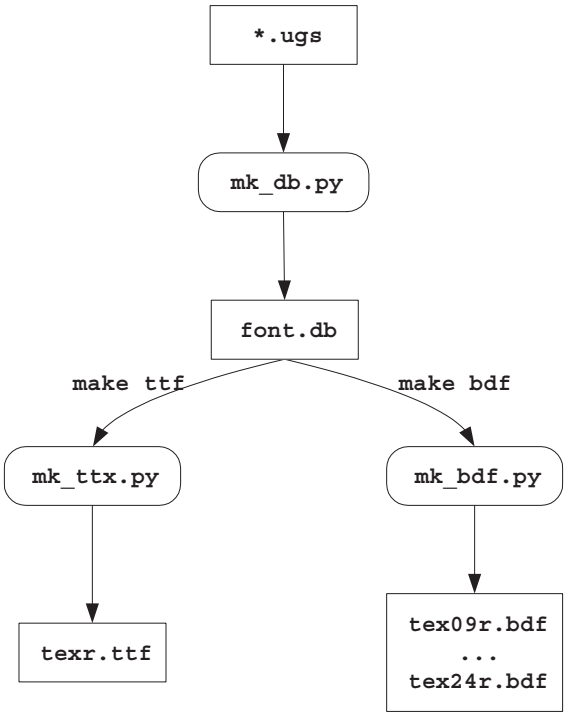


Fig. 2: Translating UGS to TTF and BDF

The top directory of the METATYPE package contains several utilities, scripts and makefiles necessary for processing fonts. The source code of fonts is contained in subdirectories, one directory per font family. For example, the family of fonts named "TeX" is placed in subdirectory `cm/`.

To prepare all the METATYPE utilities, execute `make` in the top directory of the METATYPE project.

Font source code is organized into three levels. All Unicode space is divided into blocks of 256 symbols, and each block is placed in a separate directory. Blocks are further divided into 16 subblocks with 16 characters each. An example of the file structure of a single font family is shown in figure 3.

The `compile` directory is used when compiling a font. Each font style is compiled in a separate subdirectory with the appropriate name: `compile/roman`, `compile/roman-italic`, etc. During compilation, subdirectories for all blocks and subblocks are created automatically.

- To compile all glyphs for all styles, enter the `compile` directory and execute `make`.
- To build TrueType fonts, execute `make ttf`.
- To build fonts in BDF format (sizes 9, 10, 11, 12, 13, 14, 18 and 24 points at 100 dpi), run `make bdf`.

The result

Here is an example of the TeX Roman and Maestro Roman fonts created by METATYPE. These figures were made using Adobe Illustrator and imported into the article as EPS graphics.

TeX:

"Мой дядя самых честных правил,
Когда не в шутку занемог,
Он уважать себя заставил
И лучше выдумать не мог."
"My uncle was a man of virtue,
When he became quite old and sick,
He sought respect and tried to teach me,
His only heir, verte and weak."

Maestro Roman:

"Мой дядя самых честных правил,
Когда не в шутку занемог,
Он уважать себя заставил
И лучше выдумать не мог."
"My uncle was a man of virtue,
When he became quite old and sick,
He sought respect and tried to teach me,
His only heir, verte and weak."

(A. S. Pushkin. *Yevgeny Onegin*; English translation by Dennis Litoshick.)

Unsolved problems

At present the Fonttools library does not support adding raster data to TrueType fonts. Extending this library by adding support for the tables EBDT, EBLC and EBSC is required.

The problem of kerning is not yet solved. We plan on adding an algorithm for automatic kerning.

Further, the character set of the TeX and Maestro font families is incomplete. To cover even the simplest of needs, adding support for Latin-1, Latin Extended-A and Latin Extended-B is highly desirable.

Lastly, the contours created by tracing rasters are not optimal. It would be nice to build an optimizer to reduce the number of spline segments without distorting the glyph's appearance.

Acknowledgments

The author wishes to thank Cronyx Engineering Company for making available the necessary resources for the project.

References

[1] http://www.unicode.org/history/
[2] http://www.microsoft.com/typography/specs/
[3] http://www.freetype.org/
[4] http://www.tug.org/web2c/
[5] ftp://cam.ctan.org/tex-archive/fonts/cm/mf/
[6] http://www.vak.ru/proj/metatype/cm/roman/
[7] http://www.vak.ru/proj/metatype/cm-math/roman/
[8] http://obelix.ee.duth.gr/~apostolo/mf2pt3.html
[9] http://autotrace.sourceforge.net/
[10] http://www.freetype.org/freetype2/index.html
[11] http://www.freetype.org/patents.html
[12] http://sourceforge.net/projects/fonttools/
[13] http://www.python.org/
[14] http://sourceforge.net/project/showfiles.php?group_id=41605
[15] cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/metatype checkout metatype
[16] http://python.org/2.2.2/

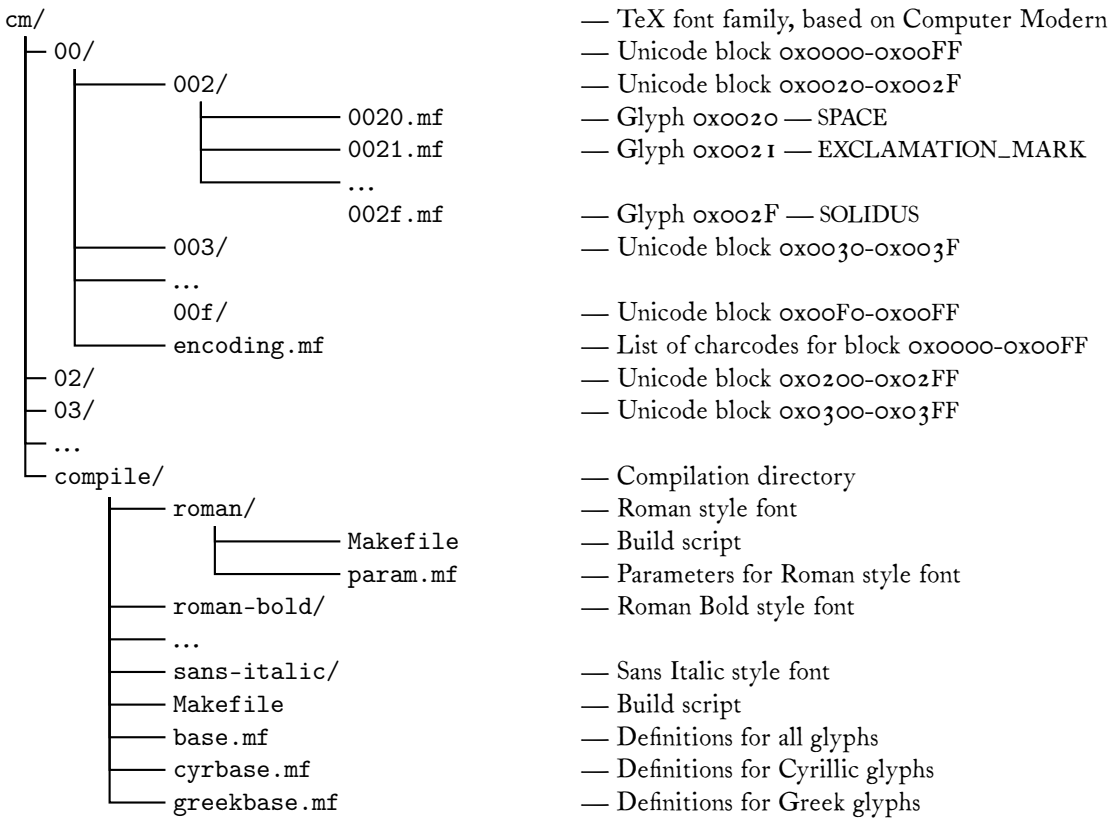


Fig. 3: Structure of the METATYPE font source directory

[17] http://prdownloads.sourceforge.net/fonttools/fonttools-2.0b1.tgz
[18] http://prdownloads.sourceforge.net/pyxml/PyXML-0.8.2.tar.gz
[19] http://prdownloads.sourceforge.net/numpy/Numeric-23.0.tar.gz
[20] http://prdownloads.sourceforge.net/netpbm/netpbm-10.15.tgz
[21] http://prdownloads.sourceforge.net/libpng/libpng-1.2.5.tar.gz
[22] http://prdownloads.sourceforge.net/freetype/freetype-2.1.4.tar.gz
[23] http://www.tug.org/teTeX/
[24] http://prdownloads.sourceforge.net/autotrace/autotrace-0.31.1.tar.gz

Programming PostScript Type 1 Fonts Using METATYPE1: Auditing, Enhancing, Creating

Bogusław Jackowski
ul. Tatrzańska 6 m. 1, 80-331 Gdańsk, Poland
B_Jackowski@gust.org.pl

Janusz M. Nowacki
ul. Śniadeckich 82 m. 46, 86-300 Grudziądz, Poland
J.Nowacki@gust.org.pl

Piotr Strzelczyk
ul. Subisława 14, 80-354 Gdańsk, Poland
P.Strzelczyk@gust.org.pl

Abstract

METATYPE1 is a METAPOST-based package for producing outline PostScript fonts in the Type 1 format. Since its first unofficial release a few years ago, it has been applied to several tasks. We would like to share our experience with other fans of computer typography.

Résumé

METATYPE1 est un ensemble d'outils basé sur METAPOST pour produire des fontes PostScript de type 1. Depuis sa première sortie, il y a quelques années, il a eu un grand nombre d'applications. Nous souhaitons partager notre expérience avec d'autres amateurs de typographie informatique.

Introduction

We started the work on METATYPE1 some five years ago. Outline fonts had already been in vogue then, especially in electronic publishing. Being adherents of both computer typography and programming, we urgently needed a programming tool for generating outline fonts. Fortunately, neither TEXtrace [7] nor FontForge [9] existed at that time. We highly esteem both of these tools and therefore we would perhaps have given up — which would be a pity, because they do not comprise all possible application areas. Sometimes, full programmability is preferable. Anyway, we had no choice but to develop a home-grown engine.

In this paper, we describe a few sample tasks that we had the opportunity to accomplish using METATYPE1, thus demonstrating the pros and cons of the programming approach to generating of outline fonts.

A brief overview of METATYPE1

As the name (and logo) suggests, METAPOST is the kernel of METATYPE1. In other words, METATYPE1 sources are written in the METAPOST language. METATYPE1 provides a task-oriented library of METAPOST macros. It also makes use of popular open source tools: Gawk and T1Utils. The METAPOST output is processed by Gawk and then assembled by T1Utils (figure 1, left). The details of the process of generating of fonts using METATYPE1 can be found in [4].

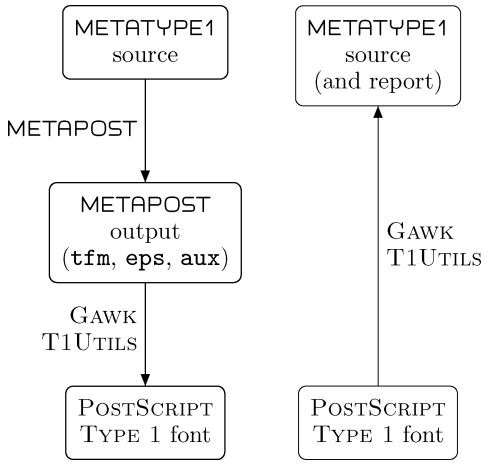


Fig. 1: A simplified scheme of the METATYPE1 engine (comments in the text).

An important part of the METATYPE1 package is a (freestanding) converter from the PostScript Type 1 format to METATYPE1 source, `pf2mt1` (figure 1, right), which also makes use of Gawk and T1Utils. As we shall see, it proves useful for auditing of fonts.

The installation of the METATYPE1 package is straightforward, once Gawk, T1Utils, and METAPOST have already been installed — the files should be copied to a chosen directory and a system variable `METATYPE1` should point to this directory. Originally, METATYPE1 was developed under DOS; its provisional Linux port is maintained by Włodek Bzyl (see [4] for the url). A simplified version of the Linux scripts is available from the authors; the scripts will be included in the next release of METATYPE1.

Many roads to the creation of fonts

Since we are no font designers whatsoever, we will assume that the design of a typeface is given. Still, the source of a font may be given in various forms — from lead prints to tentatively ready digital form. In the former case, one has to create a digitized version from scratch; in the latter, an audit of a font may prove sufficient. Below, we discuss a few different approaches to coping with fonts, starting from the simplest one, that is, from auditing.

Audit A useful yet a very simple tool for auditing of a font is the already mentioned converter from PostScript Type 1 format to human readable (and thus machine readable) METATYPE1 sources. We assume, of course, that the disassembling of a given font is legal.

Faulty hinting is perhaps most typical in freely available fonts. Even such diligently prepared fonts as the *Computer Modern* family in the PostScript Type 1 format, released as a freely available product by the American Mathematical Society in 1997, contain questionable hints.

Prior to the presentation of an example, let us briefly sketch the idea of the PostScript Type 1 hinting. Hints are special instructions, allowed only in a font program, providing additional information useful during the process of rendering of a glyph. There are two kinds of hints: vertical and horizontal; both convey information concerning the preferred width and position of a stem (which, in general, may have a varying width).

Consider now, as an example, the font `cmsy10` from the AMS *Computer Modern* family. The converter `pf2mt1` produces METATYPE1 sources, and, which is important here, a log file; here is an excerpt from the log (the first number denotes the position of a stem, the second one — width):

```
...
intersectionsq: semi-matching hstem; 558, 45
intersectionsq: non-matching vstem; 55, 40
intersectionsq: semi-matching vstem; 571, 34
```

The log indicates that there is something wrong with hints: why should such a regular character as *intersectionsq* have hints of different width (see figure 2)?

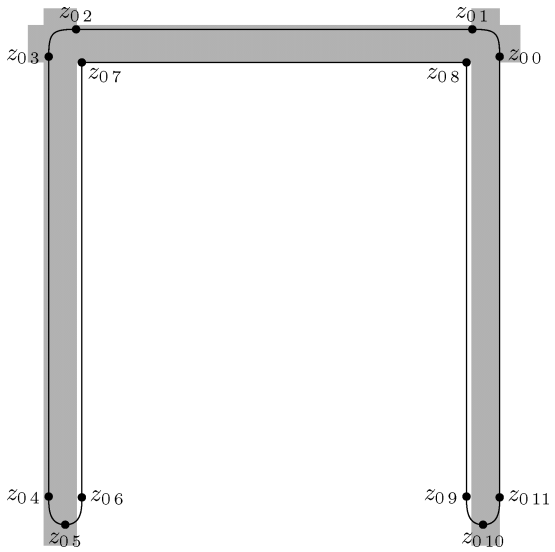


Fig. 2: Character *intersectionsq* from `cmsy10` from the collection provided by the AMS; gray rectangles mark the placement of hints — it can be seen that they do not precisely coincide with the respective stems.

The inspection of the code of the glyph *intersectionsq* reveals, as one would expect, that each stem has the same width (equal to 40; compare the following pairs of nodes: z_{04} and z_{06}, z_{09} and z_{011}, z_{08} and z_{01}):

```
beginglyph(_intersectionsq);
 save p; path p[];
 z0 0=(605,565); z0 0a=(605,594); z0 1b=(601,598);
 z0 1=(572,598);
 z0 2=(94,598); z0 2a=(65,598); z0 3b=(61,594);
 z0 3=(61,565);
 z0 4=(61,34); z0 4a=(61,20); z0 5b=(61,0);
 z0 5=(81,0); z0 5a=(101,0); z0 6b=(101,19);
 z0 6=(101,33);
 z0 7=(101,558);
 z0 8=(565,558);
 z0 9=(565,34); z0 9a=(565,20); z0 10b=(565,0);
 z0 10=(585,0); z0 10a=(605,0); z0 11b=(605,19);
 z0 11=(605,33);
 p0=compose_path.z0(11);
 correct_path_directions(p0)(p);
 if turningnumber p0>0: Fill else: unFill fi \\ p0;
 set_hstem (558,603); % stem width = 603 - 558
 set_vstem (55,95);  % stem width = 95 - 55
 set_vstem (571,605); % stem width = 605 - 571
 ghost_stem bot;
 standard_exact_hsbw("intersectionsq");
```

```
endglyph;
```

Note that the program is fairly legible and that even a superficial knowledge of the METAPOST notation suffices to guess what is wrong (here: with hints).

Among others, we tested our `pf2mt1` converter against a rich collection of fonts provided by Vladimir Volovich, namely, CM-Super [10]. The collection was prepared using TEXtrace [7] and tuned manually. The job Volovich did is really impressive. Still, a conversion to METATYPE1 sources revealed a couple of defects in nearly every font. It is not the proper place to analyse minutely the results of the conversion, we therefore confine ourselves to only one aspect: strange tiny paths ("scraps") that we spotted in about 25% of the fonts, on average in two characters per font. Two examples of such paths are depicted in figure 3. Note that the "scraps" are moderately harmful and that their (rare) occurrences can be expected only in autotraced fonts.

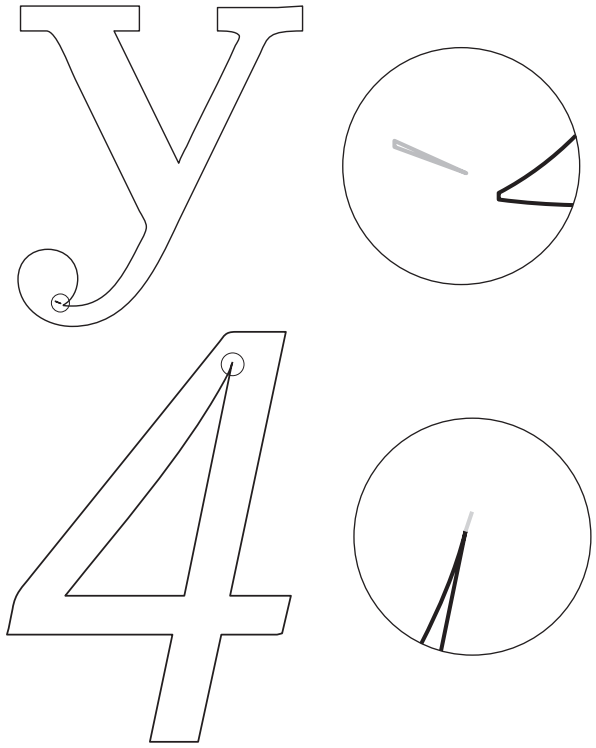


FIG. 3: Two examples of strange "scraps" (coloured gray in the magnified details) occurring in CM-Super fonts: character *y* from `sfbx0900` (top) and character *four* from `sfsi1095` (bottom).

Finding these weird paths was straightforward: we simply wrote a short Awk script that looked for tiny 3-node paths in METATYPE1 sources. For example, the relevant path in the glyph *y* from `sfbx0900` looked as follows:

```
z1 0=(98,-149);
z1 1=(98,-148);
z1 2=(109,-153);
p1=compose_path.z1(2);
```

Again, a superficial knowledge of the METAPOST notation suffices — even not knowing in detail how the operator `compose_path` works — one can easily detect such questionable objects. Note that in the case of the glyph *four* from `sfsi1095`, it would be nearly impossible to notice the superfluous path in the screen or even in a printed proof, while in the METATYPE1 source it is easy.

Let us emphasize that we point out the flaws of the CM-Super family not to deprecate it; we simply want to turn the reader's attention to the fact that METATYPE1, or its parts, can be used as an aid for different font creation software.

One can conceive of more advanced techniques of auditing, for example, using task-oriented METAPOST macros written specially to deal with a given case. But even without employing advanced techniques, many clues can be surmised from the report written during the conversion and from the inspection of the resulting METATYPE1 code.

Elementary enhancement An interesting lesson can be derived from the previous section: some improvements of a font are straighforward. For example, the elimination of the tiny "scraps" is just trivial. The correction of hinting in the program for the glyph *intersectionsq* from the font `cmsy10` is also not difficult. The proper code should read:

```
set_hstem (558,698);
set_vstem (61,101);
set_vstem (565,605);
```

or, equivalently (and more advisably):

```
set_hstem (y0 8,y0 1);
set_vstem (x0 4,x0 6);
set_vstem (x0 9,x0 11);
```

Thus, even a moderately experienced user should be reasonably able to modify METATYPE1 sources and to generate improved fonts in certain simple cases.

Another method of enhancing a font, even simpler than that of requiring the conversion to METATYPE1 sources and the examination of the results, is to use a compressing module from the METATYPE1 package. This module (actually, a short Awk script) performs an analysis of a disassembled PostScript Type 1 font (a PFB file), defines subroutines for repeating fragments of the code, and replaces the occurrences of these fragments by the respective subroutine calls. The method employed is based on an algorithm for finding the longest repeating substring of a given string. The "subroutinization" is one

of a few steps of the whole process of the generation of a font, but it can be easily disentangled.

Usually, the compression is not astoundingly efficient, for example, the fonts from the AMS *Computer Modern* family can be compressed only by 2–3%. The size of the CM-Super family of fonts, however, can be reduced by about 12% (7 MB) with this method. It is not much, but, on the other hand, it is not nothing.

Advanced enhancement The potential improvements discussed so far require only basic knowledge of both typography and METAPOST programming. Therefore, it is not a surprise that the likely results are only moderately significant — in order to achieve more, one has to know more. It is not extremely difficult, however, to obtain really useful results knowing just a little more than the rudiments.

In typical cases, an augmentation of a character set with accented characters is needed. This is a task that can be accomplished relatively easily using METATYPE1; namely, there is an operation `use_accent`, defined in the set of basic METATYPE1 macros, for precisely this purpose.

We have used this technique while preparing the collection of *Latin Modern* fonts [2]. Our starting point was, obviously, the AMS *Computer Modern* family of fonts converted to METATYPE1 format. We programmed most of the accented characters by adding just three lines of code per glyph, for example:

```
beginglyph(_aacute);
 use_accent(_a,_acute);
endglyph;

beginglyph(_abreve);
 use_accent(_a,_breve);
endglyph;

beginglyph(_acircumflex);
 use_accent(_a,_circumflex);
endglyph;
```

etc.; altogether, almost 200 characters of the *Latin Modern* family have been "programmed" using such three-liners.

In some cases, however, additional settings were needed. By default, the `use_accent` operator works similarly to the TEX `\accent` primitive — it just centres an accent over an accented character. This procedure is not adequate for such characters as *Lacute*. METATYPE1, unlike TEX, provides an optional (numeric) parameter for each character, `glyph_axis`, describing the position of the glyph axis to be used with accents. The axis is aligned either with the centre of an accent or with its axis, if it happens to be set.

For example, the program for the letter *L* from the font `lmb10` assigns the value of the x-coordinate of the central node of the top serif to the variable `glyph_axis` (see figure 4):

```
beginglyph(_L);
 save p; path p[];
 glyph_axis=x0 7; % manually added line
 z0 0=(643,274);
 z0 1=(596,274); z0 1a=(588,205); z0 2b=(571,47);
 z0 2=(392,47);
 z0 3=(289,47);
 z0 4=(289,639);
 z0 5=(424,639);
 z0 6=(424,686); z0 6a=(380,683); z0 7b=(271,683);
 z0 7=(222,683); z0 7a=(178,683); z0 8b=(77,683);
 z0 8=(39,686);
 z0 9=(39,639);
 z0 10=(147,639);
 z0 11=(147,47);
 z0 12=(39,47);
 z0 13=(39,0);
 z0 14=(612,0);
 p0=compose_path.z0(14);
 if turningnumber p0>0: Fill else: unFill fi \\ p0;
 set_hstem (0,47);
 set_hstem (639,686);
 set_vstem (596,643);
 set_vstem (147,289);
 standard_hsbw("L");
endglyph;
```

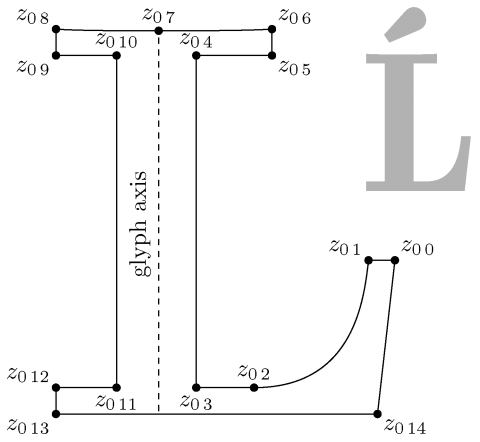


FIG. 4: An example of the setting of a glyph axis in a character *L* from a *Latin Modern* font (`lmb10`); such an axis facilitates the placement of accents in asymmetric characters such as *Lacute*.

Having adequately defined the axis for a given character (and for relevant accents, if needed), the accented characters can be defined using our neat three-liners, for example:

```
beginglyph(_Lacute);
 use_accent(_L,_Acute);
endglyph;
```

(Note that a variant acute accent is used here — accents over capital letters and ascenders are flattened a little bit in *Latin Modern* fonts. It is one of many details that can be conveniently controlled using the programming approach.)

The programming of characters that contain diacritical elements such as *ogonek* or *cedilla* is more arcane, since it requires finding the common outline of the character and diacritic. For this purpose, METATYPE1 provides the operation `find_outlines`. Although fairly handy in practice, it is not as robust as one would wish, because it behaves unstably if the curves to be processed are tangent or nearly tangent. In general, tangency presents an intrinsic problem, because it is impossible to distinguish *numerically* between tangent curves and intersecting ones. Metaphorically speaking, tangency is an infinitesimal property, while computers are discrete, and thus finite.

With METAFONT, things become even worse. For example, the built-in METAFONT operator `intersectiontimes`, which we need, sometimes cheats when disjoint curves are touching each other. Another operator, `turningnumber`, important in dealing with paths, also is not sufficiently reliable (due to, for example, "tiny loops" — see [5]).

METAFONT has been designed to produce bitmaps, not outlines. In such applications, curves that nearly touch each other or tiny loops that change unpredictably the turning number of a curve do not cause too much harm. Its descendant METAPOST, although it is no longer bitmap-oriented, inherited METAFONT arithmetic with all its imprecision; the same problems can be thus likely encountered in METAPOST. One can live with that but the price is the necessity of carefully controlling the situations where nasty tangency-related issues emerge. It is worthy of emphasizing that although such control requires some proficiency in programming in METAPOST, it is usually effortless.

The most complex aspect of a font enhancement is obviously constructing a glyph from scratch. This topic, in fact, has more to do with the creation than with the enhancement of a font. Thus, we postpone it for a while.

From METAFONT to METATYPE1 Although the issue of an automatic conversion of METAFONT font programs into an outline form conforming to PostScript Type 1 standards was recognised relatively long ago, it has still not been fully solved.

A partial solution was announced by Péter Szabó during the EuroTEX 2001 meeting (Kerkrade, The Netherlands [7]).

Szabó's TEXtrace program produces outlines of acceptable quality, thanks to the efficacious Autotrace program [8] by Martin Weber, which is employed for the autotracing of high-resolution bitmaps. Nevertheless, we call Szabó's solution only "partial" since the process of autotracing cannot (yet?) be controlled in detail, while we firmly believe that every detail should be controllable and replicable in the realm of TEX and METAFONT/METAPOST.

Another solution which we prefer, but which might be called "partial" as well, is a manual alteration of METAFONT sources. We applied this approach to generate a METATYPE1 version of Donald E. Knuth's `logo` font. A sample page from the automatically generated documentation of the font is shown in figure 5.

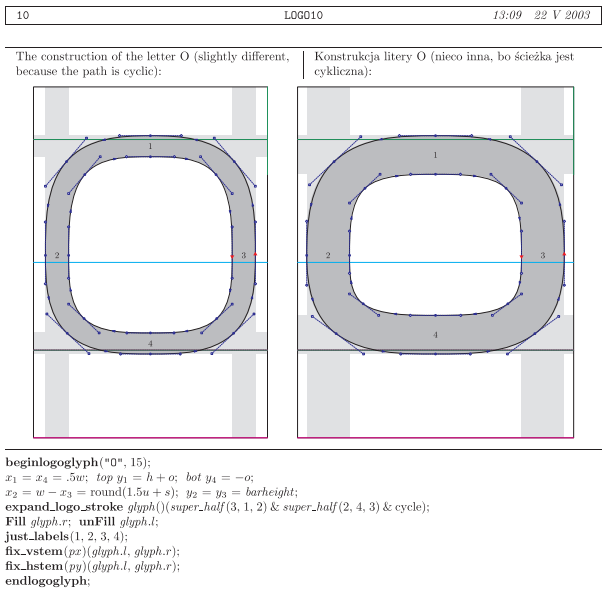


FIG. 5: A sample page from the automatically generated documentation of the METATYPE1 version of the `logo` font; light gray rectangles mark hints; a high-resolution bitmap generated from the original METAFONT sources (painted with a darker gray) is placed in the background for comparison.

We would like to stress that it was a purely experimental effort, because Taco Hoekwater had already prepared a PostScript Type 1 version of the `logo` font using Richard J. Kinch's MetaFog. The differences of glyph shapes between Hoekwater's fonts and ours are negligible. More interesting is perhaps that we managed to obtain metric files (tfm) identical with the original. Those who are interested in technicalities may wish to download both the sources and the results from the same ftp address as the METATYPE1 package. A conversion of Hoekwater's PostScript Type 1 fonts into METATYPE1 sources using `pf2mt1` can be also instructive.

In the *Latin Modern* family, several glyphs were also prepared using this method. In particular, we adapted

the program for the letter *C* from the *Computer Modern* METAFONT sources in order to construct the *Euro* currency symbol (with the addition of a bottom serif, since the *Euro* symbol is philosophically based on a script *E*; thanks to Werner Lemberg).

The point of the story is that, in principle, such a conversion is possible. It remains an open question, however, whether it is worthwhile to attempt to convert all available METAFONT sources. On one hand, such outline-oriented programs would certainly be useful, especially for those who would like to generate new variants of old fonts by playing around with parameters; on the other hand, such a conversion is a laborious task, although in most cases it is routine. In our opinion, however, converting of existing METAFONT sources into an outline form is not as urgent as the problem of the scarcity of new fonts. But the creation of a new font is a challenge, indeed.

Creation The issue of font creation, even limited to the computer-oriented aspects, is a topic rather for a book than for a section in a short article. Therefore, we will not dwell too much on this inexhaustible subject. Instead, we simply sum up the experience we gathered during the work on two replicas of Polish traditional typefaces (figure 6): Antykwa Półtawskiego [3] and Antykwa Toruńska, the latter being intensively developed by Janusz M. Nowacki.

Antykwa Półtawskiego
Antykwa Półtawskiego

Antykwa Toruńska
Antykwa Toruńska
Antykwa Toruńska
Antykwa Toruńska

Fig. 6: Two replicas of Polish traditional typefaces, prepared with METATYPE1.

For both typefaces, only lead prints were available. In many cases, therefore, we had to conjecture as to the details. Surprisingly, this seemingly difficult task of reconstructing and programming of glyph shapes, turned out to be relatively simple. The adjusting of sidebearings and kerns was significantly more difficult. But it is the parameterization of a font that is really a tough problem. We do not want to juggle lots of incoherent parameters. Frankly speaking, we would not be able to. Recall that the *Computer Modern* fonts are governed by more than 60 parameters — this would be certainly too much for us.

After many experiments and discussions, we are still unsatisfied with the current parameterization of Antykwa Półtawskiego. This is the main reason why we have not publicly released its METATYPE1 sources so far. We have not given up yet — we certainly hope that before long we will eventually release tolerably tidy METATYPE1 sources. On demand, however, they can be obtained from the authors for private use.[1]

We have not heard about other book typefaces created using METATYPE1. Our program has rather a small number of users and is mainly employed for auditing and enhancing fonts, although we know about a few fonts containing geometrically regular glyphs (for example, geographical symbols) created using METATYPE1; among them, perhaps the most interesting is a humorous font displayed in figure 7.

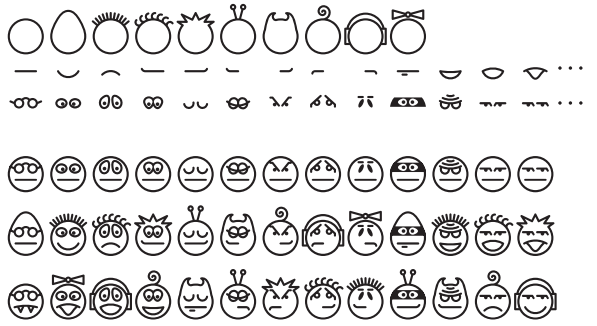


Fig. 7: A dingbats font created with METATYPE1 by Jacek Kmiecik: the top three rows contain characters of the font, the bottom rows present an effect of superimposing of three characters (one from each row); the inspiration for this work was the font `Head-DingMaker`, signed by Nigma Fonts (`kentpw@norwich.net`).

From a TEX user's point of view, it is important that METATYPE1 is suitable for creating TEX-oriented math fonts (note that the popular `afm2tfm` program cannot be used for generating math tfm files). We know about a couple of such endeavours. In one of them we are involved: it is an ongoing project of the Polish TEX Users Group GUST of furnishing Antykwa Półtawskiego with a math extension. So far, however, nobody has announced a release of a math font created with METATYPE1. The bright side of this otherwise disquieting situation is that interesting events are still ahead of us.

1. In the near future, most probably before EuroTEX 2005, the METATYPE1 sources of Latin Modern family of fonts will be released publicly.

All in all, METATYPE1 promises well as far as we can tell. But please feel forewarned that we are incorrigible optimists. Also feel forewarned, as we emphasize repeatedly, that although there are many roads to the creation of fonts, the royal road does not exist.

Concluding remarks

We took for granted the PostScript Type 1 format is exactly what we need. We are aware that the future belongs to a successor of Type 1 and TrueType, namely, to OpenType, but we do not consider it a threat. Let us quote a Microsoft faq [6]: *OpenType* [...] *is an extension of Microsoft's TrueType Open format, adding support for Type* 1 *data.* Moreover, there is an official *Font Development Kit* for OpenType released by Adobe [1] which offers, among several less or more useful functions, a conversion of PostScript Type 1 fonts into their OpenType counterparts. We bumped into some obstacles trying to tame this tool, but finally managed to achieve the desired outcome. The results are not easily verifiable because OpenType, advertised since 1996, actually entered the scene only a few years ago. Nevertheless, it seems that such a conversion may be an appropriate solution for some time to come.

We also have tried FontForge as a converter to OpenType fonts. Again, we were successful, although not without trouble. It is too early, however, to evaluate the existing tools definitively, the more so as other tools of this kind can be expected to become available in the nearest future.

Is METATYPE1 a convenient tool for creating fonts at present and in the future? The answer is up to users. Certainly, it is a tool for creating *open source fonts*, that is, *truly open types.*

Acknowledgements

Very many thanks to Jerzy Ludwichowski for his kind support during the preparation of this paper.

References

[1] *Adobe Font Development Kit for OpenType*, http://partners.adobe.com/asn/developer/type/otfdk/

[2] Bogusław Jackowski, Janusz M. Nowacki, *Enhancing Computer Modern with accents, accents, accents*, *TUGboat* 24(1), Proc. of the 24th Annual Meeting and Conference of the TEX Users Group, p. 64–74.

[3] Bogusław Jackowski, Janusz M. Nowacki, *Antykwa Półtawskiego: a parameterized outline font*, EuroTEX 1999, 20th – 24th September, 1999, Heidelberg, Germany, pp. 109 – 141; the current version of Antykwa Półtawskiego is available from ftp://ftp.GUST.org.pl/pub/TeX/GUST/contrib/fonts/replicas

[4] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *METATYPE1: A METAPOST-based Engine for Generating Type* 1 *Fonts*, Proc. of EuroTEX 2001, 27th – 27th September, 2001, Kerkrade, the Netherlands, pp. 111 – 119; the current version of METATYPE1 is available from ftp://bop.eps.gda.pl/pub/metatype1; METATYPE1 for Linux can be downloaded from ftp://ftp.ctan.org/tex-archive/systems/unix/mtype13/

[5] Donald E. Knuth, *The METAFONTbook*, Addison-Wesley, edition VII, 1992, pp. 152, 228 – 229.

[6] *OpenType initiative FAQ*, http://www.microsoft.com/truetype/faq/faq9.htm

[7] Péter Szabó, *TEXtrace*, http://www.inf.bme.hu/~pts/textrace/

[8] Martin Weber, *Autotrace*, http://autotrace.sourceforge.net/

[9] George Williams, *FontForge — a PostScript Font Editor*, http://fontforge.sourceforge.net/

[10] Vladimir Volovich, *CM-Super Font Package*, ftp://ftp.vsu.ru/pub/tex/font-packs/cm-super/

Typesetting Rare Chinese Characters in LaTeX

Wai Wong, Candy L.K. Yiu, Kelvin C.F. Ng
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Kowloon
Hong Kong
wwong,candyyiu,ncfkelvin@comp.hkbu.edu.hk
http://www.comp.hkbu.edu.hk/~wwong

Abstract

Written Chinese has tens of thousands of characters. But most available fonts contain only around 6 to 12 thousand common characters that can meet the needs of everyday users. However, in publications and information exchange in many professional fields, a number of rare characters that are not in common fonts are needed in each document. This paper describes a method of typesetting such rare characters in LaTeX. The document author describes a rare character in HanGlyph when such need arises. A Chinese character synthesis system renders the glyph according to this description and collects the newly created glyphs into a font so they are available in the body of the LaTeX document.

Résumé

Le chinois écrit possède des dizaines de milliers d'idéogrammes. Mais la plupart des fontes disponibles ne contiennent que quelques 6 et 12 mille idéogrammes standard. Néanmoins, les publications en sciences humaines, comme la littérature classique, l'archéologie, ou, tout simplement, les registres municipaux de noms, nécessitent l'utilisation de grand nombre d'idéogrammes rares.

Cet article décrit une méthode de composition de de tels caractères rares sous LaTeX. L'auteur du document décrit le caractère rare dans une syntaxe spéciale, appelée HanGlyph. Un système de synthèse de caractères chinois génère le glyphe d'après cette description et assemble les glyphes ainsi produits dans une fonte pour être immédiatement utilisés par LaTeX dans le corps du document.

Introduction

The Chinese written script is ideographic. Each ideograph, known as *hanzi*, or more commonly 'Chinese character', has its own visual structure and carries certain meaning. There are tens of thousands of *hanzi* or Chinese characters.

The most notable difference between an ideographic script and a phonographic script, such as the Latin alphabet, Slavonic alphabet, and so on, is the huge number of characters that the former script contains.

Historically, the number of *hanzi* in existence increases with time. This is reflected in many dictionaries published in various times. For example, the first influential book on *hanzi* 說文解字 (shuōwénjiězì[1]) published around 100 AD collected 10,516 characters. By the time of the Qing dynasty, the more famous dictionary 康熙字典 (kāngxīzìdiǎn) (published in 1716) contains 47,043 characters. The largest contemporary dictionary 中華字海 (zhōnghuázìhǎi) documents a staggering 86,000 characters [5].

In order to facilitate the machine processing of the Chinese script, coded character sets have been developed [8]. Commonly used coded Chinese character sets are GB2312-80, CNS11643, Big5, JIS X 0208-1983 and KS X 1001:1992. The number of Chinese characters encoded in these character sets varies from around 4,888 (in KS X 1001:1992) to 48,027 (in CNS 11643-1992).

The main reason behind the selection of characters in these encoded character sets is the frequency with which each character appears in common documents, such as newspapers, textbooks and business correspondence. A statistical study reveals that around 6,600 Chinese characters can cover 99.999% of daily use [5]. It seems that having a large enough encoded character set will solve the problem.

Recent international effort in character set standardization results in the Unicode (version 4.0 released in May 2003) [12] and ISO/IEC 10646 standards. Unicode 3.0 [9] encoded 27,484 Chinese characters. 42,711 characters were added in version 3.2 [11].

Although the new international standards include a

1. The word shuōwénjiězì following the *hanzi* is in *pinyin*, a phonetic transcription of Chinese characters.

huge number of Chinese characters, in practice, several problems remain unsolved. The first is that the design and creation of fonts of a huge character set is very expensive. It is also not economical to process and maintain such large fonts given that many of the characters in them are rarely used. Furthermore, a large number of existing systems may not be able to handle such a large character set. Transferring a document to such older systems will result in missing characters or even crash the system.

One possible solution is to synthesize the characters when needed. We have designed a Chinese character description language called *HanGlyph*. It is a high-level abstract description language which captures the essential features of characters. These features are the topology of the strokes and their relative size and location. We are developing a Chinese character synthesis system (CCSS) to render the characters from the *HanGlyph* description. The *HanGlyph* language and the method of synthesizing Chinese characters will be described in the next section.

Using the *HanGlyph* description and the character synthesis system, we can typeset rarely used Chinese characters in LaTeX documents. We developed a simple macro package to allow the document author to embed *HanGlyph* descriptions in a LaTeX document. During the first time the LaTeX document is formatted, a *HanGlyph* file containing all character descriptions is generated. This file is then processed by the synthesis system to create a font. The next time the LaTeX document is formatted, the newly generated Chinese character font is accessible. Thus, the character can be typeset. Later sections will describe how to use this macro and outline its implementation.

Chinese character synthesis

The HanGlyph language is a Chinese character description language. It provides a means of describing the topological arrangements of strokes in Chinese characters. Each Chinese character can be decomposed into a number of parts called *components*. Each component consists of a number of *strokes*. For example, as illustrated in Figure 1, the character 明 (míng meaning bright) can be decomposed into two components 日 (rì) and 月 (yuè). The first component consists of four strokes: 丨(豎 shù), 乛(橫折 hèngzhé), 一(橫 héng) and 一.

A *HanGlyph* expression describes the arrangement of the strokes in an abstract fashion. This means that only topological information is captured and no geometric information is included. For example, the *HanGlyph* expression describing the character 二 is `h h =` which means the character is composed by two héng strokes, one above the other. We do not need to specify the exact coordinates of the starting point of each stroke. These will be worked out by the character synthesizer.

One of the criteria for writing a good *HanGlyph* expression is to be able to distinguish similar characters. For example, the characters 土 and 士 are composed of the same strokes and in the same arrangements. The only difference is the relative length of the two 一 strokes. The *HanGlyph* expressions to describe these characters are `h h=< s+_` and `h h=> s+_`, respectively. The dimensional relation symbols < and > specify the relative length of the two 一 strokes.

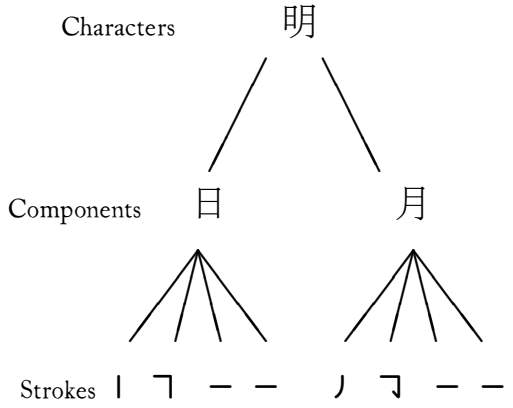


Fig. 1: Composition of a Chinese character

Because the smallest building blocks of a Chinese character are the strokes, the *HanGlyph* language takes the strokes as its primitives. According to many studies of Chinese characters, we selected 41 such primitive strokes. *HanGlyph* composes characters using a small set of five operations which are illustrated in Figure 2:

- top-bottom,
- left-right,
- fully-enclosed,
- partially-enclosed, and
- crossing.

It would be very tedious if every character description contains details down to each single stroke. Using the fact that characters can be decomposed into components and many components appear in a number of characters, *HanGlyph* allows macros to be defined to stand for components. Thus, expressions can be very concise.

In summary, *HanGlyph* provides an abstract way to describe Chinese characters which captures their essential characteristics so that they can be rendered. Details of the *HanGlyph* language can be found in our paper to be presented at *TUG* 2003 [14].

The Chinese character synthesis system (CCSS) is responsible for rendering the character glyphs from their *HanGlyph* descriptions. The core of CCSS is implemented in METAPOST [6]. It consists of a library of METAPOST macros implementing various *HanGlyph* operations. The input to CCSS is a sequence of *HanGlyph*

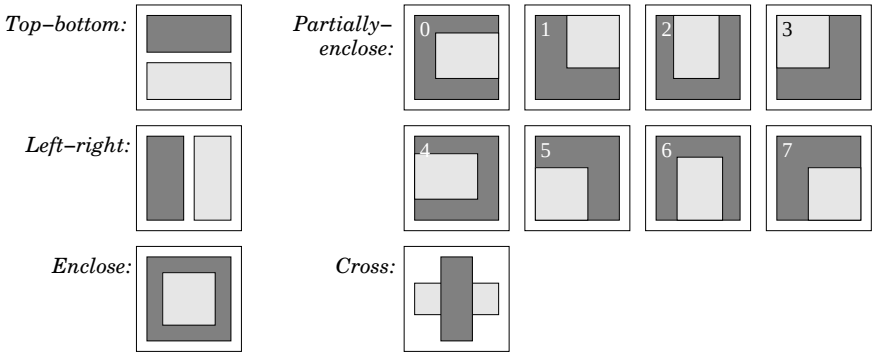


Fig. 2: Graphical representation of operators

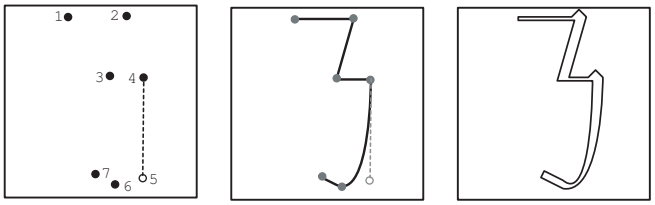


Fig. 3: Basic stroke macros for the stroke 乙

expressions. A front end translator converts these *HanGlyph* expressions into a METAPOST program. Running METAPOST on this program will then generate a set of PostScript files each of which contains a single glyph.

The METAPOST macro library is organized into two major parts: the primitive strokes and the composition operations. Each primitive stroke is implemented as three METAPOST macros, namely a control point macro, a skeleton macro and an outline macro. The control point macro defines the control points, the skeleton macro specifies the skeletal path that connects the control points, and the outline macro draws the outline which are defined relative to the control points and the skeletal path. Figure 3 shows a sample stroke.

The composition operation macros perform the composition. This is done by transforming the control points of each stroke and position them to the correct location within a character bounding box. When all strokes of a character are positioned at the correct location, the skeleton macros are called to define the skeletal strokes. Then, the outline macros are called to draw the outline.

Using HanGlyph in LaTeX

The *HanGlyph* language provides a means of describing Chinese characters, and the CCSS system renders the characters so that we can have visual output. How can we then use them in the context of typesetting professional documents in LaTeX? Figure 4 illustrates the process flow.

First of all, the document author will embed the *HanGlyph* expressions of the required Chinese characters in the LaTeX source files. Typesetting the document will generate a *HanGlyph* file that contains all *HanGlyph* expressions in the source. This file is then processed by the CCSS fontmaker to generate a font in `tfm` and `pk` format so that the next time LaTeX is run it can find the font.

To allow the author to embed *HanGlyph* expressions and to use the synthesized characters in a LaTeX document, we developed a macro package `hanglyph`. This provides a very simple interface for authors to use *HanGlyph*. To define a new character, the author writes the *HanGlyph* expression in the preamble of a LaTeX document as below:

```
\hgchar{tu}{h h=< s+_}
\hgchar{shi}{h h=> s+_}
```

Each call to the macro `\hgchar` defines a new *HanGlyph* character. The macro takes two arguments: The first is a character name which can be used in the document to typeset the character; the second is the *HanGlyph* expression describing the character. For example, the first call to `\hgchar` above defines a new macro `\tu` to represent the character 土.

When processing the LaTeX document, each call to the macro `\hgchar` also writes a *HanGlyph* character definition to a *HanGlyph* file. Each character definition associates a character code to its *HanGlyph* expression.

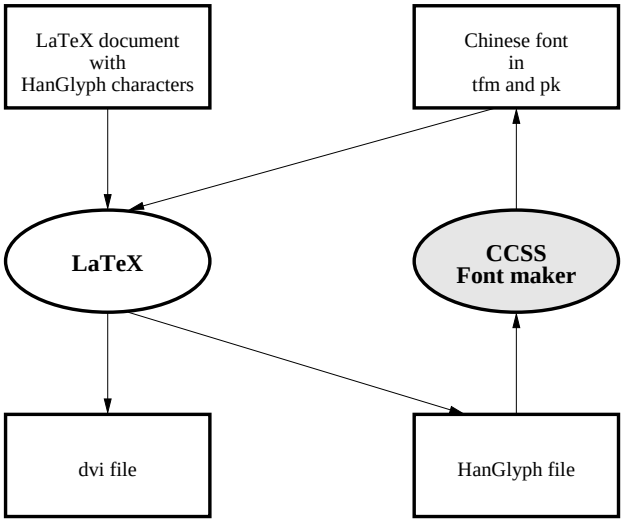


FIG. 4: Typesetting process

The `hanglyph` package automatically keeps track of the character code. By default, the first character, defined by the first call to the macro `\hgchar`, has the code 0.

In the body of the LaTeX document, the character names defined using the macro `\hgchar` can be used to typeset these characters. So the author can type

```
The Chinese characters \tu{} and \shi{}
are composed of the same strokes and in
the same arrangements.
```

to typeset the following:

> The Chinese characters 土 and 士
> are composed of the same strokes and in
> the same arrangements.

To use the `\hgchar` macro, the author should include the `hanglyph` package in the LaTeX source file. After running LaTeX once, a *HanGlyph* file having the suffix `hgc` and the same base name as the LaTeX document is generated. The author should execute the CCSS fontmaker to generate the `tfm` and `pk` files needed by LaTeX and the `dvi` driver. The fontmaker should be executed each time the embedded *HanGlyph* expressions are changed so that the font is up-to-date. Figure 6 shows some sample characters generated by our system.

This approach of typesetting Chinese characters aims at applications where only a small number of rarely used characters appear in a document. By 'rarely used characters', we mean those characters that cannot be found in commonly available fonts, such as those that come with the popular operating systems or typesetting systems. Our method provides a simple and convenient way to solve this missing character problem. Therefore, the current implementation of the `hanglyph` package is able to handle up to 256 characters in a document.

The implementation

The implementation is in two parts: the macro package `hanglyph`, and the CCSS fontmaker implemented as a suite of scripts and C programs.

The `hanglyph` package defines the macro `\hgchar` for the document author to specify the Chinese character and a number of auxiliary macros to manage the Chinese character font.

The package works as follows: All characters defined by calling `\hgchar` will be collected into a font with the default font name `hgfont`. Each character is assigned a character code in the order it is defined. A counter named `hg@charcode` keeps track of the number of characters that have been defined.

The character definition macro `\hgchar` takes two arguments, namely the character name and a *HanGlyph* expression. It performs two tasks: defining a character macro and writing the character description to the *HanGlyph* file.

In order to allow the character name macros to be used in the document body, the definitions must be placed in the preamble. At 'begin document', the *HanGlyph* file will be closed and all character macros have been defined. Each character macro is defined to represent a single character in the default *HanGlyph* font.

By default, the font metric file and the glyph file have the same basename as the document file. The `hanglyph` package provides a macro `\hgfontname` for the author to change the font file name.

The CCSS fontmaker takes a *HanGlyph* file and generates a font to be used in the LaTeX document. The process is slightly long-winded but the philosophy is to use as many

existing tools and file formats as possible, so that the generated files are compatible with existing systems. This process is depicted in Figure 5.

The core of the process is the Chinese character synthesis system. The output of CCSS is a set of encapsulated PostScript (`eps`) files. Each file contains a single glyph. They are then converted to bitmaps in portable bitmap format (`pbm`). This task is accomplished by Ghostscript. The set of `pbm` files is then merged into a large bitmap, which is then converted to a TEX font in a pair of `gf` and `pl` files. This pbm-to-gf conversion is performed by a utility called `pbmtogf` developed by the first author several years ago [13] and since been made available on CTAN. The program to merge a set of pbm files is `pbmmerge`, which is relatively simple to implement. The last two programs, `pltotf` and `gftopk`, are available in all TEX implementations. This process is easily integrated in a single script.

Discussion and conclusion

Using the *HanGlyph* description language and CCSS, one can generate Chinese character glyphs that are not found in commonly available fonts. To typeset such characters requires generating a font in the format understood by the typesetting system. The `hanglyph` package enables users of the LATEX system to typeset rarely used Chinese characters.

At this stage, we have experimented with a small number of characters. The results (some of which are illustrated in Figure 6) show that this approach is feasible, and very promising. We are planning to carry out more experiments to typeset a reasonably large set of characters. The purpose is to study the effect of rasterization and to improve the quality of the glyphs generated by CCSS. We are confident that our method of character synthesis can produce reasonably good quality and readable Chinese character glyphs.

It is obvious that the current font making process is a bit inefficient. Another shortcoming is the loss of scalability by converting the vector representation of character glyphs into a bitmap form. These weaknesses will certainly be eliminated as the development of CCSS progresses.

The current stage of the development of CCSS concentrates on the experiment and improvement of glyph algorithms. It is planned that future versions of CCSS will generate outline fonts for *HanGlyph* input. This will certainly improve quality and usability.

This paper describes only one of many possible applications of *HanGlyph* and Chinese character synthesis. We envisage that CCSS will contribute greatly to easing a major difficulty in Chinese textual information exchange and presentation in a heterogenous environment.

References

[1] 蘇培成 (sūpéichéng).《二十世紀的現代漢字研究》(èrshíshìjìde xiàndài hànzì xiānjiù) 書海出版社 (shūhǎi chūbǎnshè), 2001. [SU Pei Cheng, *20th Century Research on Modern Chinese Characters*, Su Hai Press, 2001.]

[2] 劉連元 (liúliányuán).〈漢字拓扑結構分析〉(hànzì tuōpǔ jiékoù fēnxī). In《漢字》(hànzì). 上海教育出版社 (shànghǎi jiàoyù chūbǎnshè), 1993. [LIU Lian Yuan, *Analysis of the Topological Structure of Chinese Characters*, Shanghai Education Press, 1993.]

[3] 傅永和 (fùyónghé).《漢字結構和構造成分的基楚研究》(hànzì jiékoù he koùzào chéngfènde jī chǔ yānjiù). 上海教育出版社 (shànghǎi jiàoyù chū bǎnshè), 1993. [FU Yong He, *Basic Research on the Structure of Chinese characters and Their Constituents*, Shanghai Education Press, 1993.]

[4] 傅永和 (fùyónghé).《中文信息處理》(zhōngwén xìnxī chǔlǐ). 廣東教育出版社 (guǎngdōng jiàoyù chūbǎnshè), 1999. [FU Yong He, *Chinese Information Processing*, Guangdong Education Press, 1999.]

[5] 馮志偉 (féngzhìwěi).《計算語言學基礎》(jìsuàn yǔyánxué jīchǔ). 商務印書館 (shāngwù yìnshū guǎng), 2001. [FENG Zhi Wei, *Foundations of Computational Linguistics*, The Commercial Press, 2001.]

[6] J. D. Hobby. *A METAFONT-like system with PostScript output*, *TUGboat*, Vol. 10, No. 4, pp. 505–512, December 1989.

[7] John D. Hobby. *A User's Manual for METAPOST*, AT&T Bell Laboratory Computer Science Technical Report, No. 162, 1992.

[8] Ken Lunde. *CJKV information processing*. O'Reilly, 1999.

[9] The Unicode Consortium. *The Unicode Standard, Version* 3.0. Addison-Wesley, 2000.

[10] The Unicode Consortium. *Unicode Version* 3.1, The Unicode Standard Annex No. 27. `http://www.unicode.org/reports/tr27/`

[11] The Unicode Consortium. *Unicode Version* 3.2, The Unicode Standard Annex No. 28. `http://www.unicode.org/reports/tr28/`

[12] The Unicode Consortium. *The Unicode Standard, Version* 4.0. Addison-Wesley, 2003.

[13] Wai Wong. *pbmtogf — converting bitmap to font*. `http://www.comp.hkbu.edu.hk/~wwong/typeset/pbmtogf/`

[14] Candy L.K. Yiu, Wai Wong. *Chinese character synthesis using METAPOST*. *TUGboat*, Vol. 24, No. 1, pp. 85–93, 2003 (TUG 2003 proceedings).

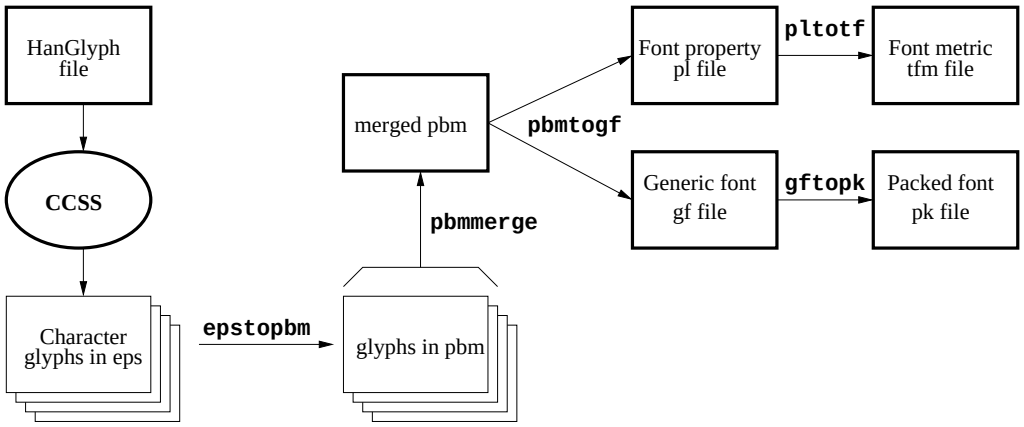


Fig. 5: The fontmaking process

Glyph	HanGlyph expression
八	`p n |`
把	`hSt ba_1 | !~`
班	`wang_b_2 d q | ~ | !~ wang_a_2 | !~`
般	`zhou_1 shu_1 |`
版	`pian_4 /Pq kn ^7 _]/|`
半	`h s+ d Z|!~ h= @ ’`
壁	`qih sih ^7 0.8 0.8 _ xin_1_b | hhs1 =`
避	`div qih sih ^7 0.8 0.8 _ xin_1_b | ^1`
辨	`xin_1_b D q | ~ | xin_1_b |`
昌	`ri_4 ri_4 = <`
长	`p h=0.4<!~n=>]e+#[`
吃	`sih ph m = |!~ 0.5 0.5`
吹	`sih qian_4 |!~ 0.5 0.5`
辞	`she_2 xin_1_b |`
此	`zhi_3_a wp |`
寸	`h S+<’ d^5`
到	`h rd = hst = sS |!~`
敌	`she_2 pu_1_b |`
笛	`zhu_b_2 ri_4 s + _ < =`
第	`zhu_b_2 ihC s + ’ < p ^5 0.5 _ [ =`
独	`pXp chong_2 |!~`

Fig. 6: Some characters generated using *HanGlyph* and CCSS

Formatting Font Formats

Luc Devroye
McGill University, Montréal, Canada H3A 2K6
luc@cs.mcgill.ca

Abstract

Font formats are a tug of war between artists (designers and drawers), programmers (computer scientists), the business world, and users. Each of these four groups has had an influence on the path that font formats have followed. We review the successes and failures, and present a wish list of properties that a good font format should have.

Résumé

Les formats de fontes ont depuis toujours été un tir à la corde entre les artistes (graphistes et dessinateurs de fontes), les programmeurs (informaticiens), le monde des affaires et les utilisateurs. Chacun parmi ces groupes a influencé l'itinéraire historique que les formats de fonte ont suivi ces vingt dernières années.

Nous allons, dans cette présentation, revoir les succès et les échecs des formats de fonte, et nous allons présenter une liste de vœux des propriétés que nous considérons qu'un bon format de fonte devrait avoir.

Introduction

Let us try to imagine what format fonts will be living in several decades from today. That question is very relevant in 2003, as the type world is ready for yet another overhaul. In this paper, we briefly comment on the present situation, in which the TrueType and PostScript font formats are dominant, and the OpenType format, which was proposed about eight years ago, is being promoted. We then take a broader and more long-term view and touch upon various issues related to the design of electronic font formats.

Before we embark on the more technical aspects of electronic font formats, it helps to identify the forces that are helping to shape these formats.

First and foremost, the *users* would like to see simple, useful formats, that are easy to manipulate and edit. They want to have access to the art created by great type artists and the technical refinement provided by digital font experts. In addition, professional users may demand a certain degree of flexibility in a font, in order to incorporate personal choices.

The *artists and typographers* had a lot of influence in pre-electronic font formats. The early typographers were nearly all craftsmen. In the twentieth century, various technological advances were made at companies like Linotype and Monotype, that were driven by the demands of the type designers, and we witnessed a shortening of the time between design on paper and actual glyph production. In the electronic era, the artists and typographers have been largely left out of the decisions on font formats, and this has led to an unfortunate split in the family of typographers: on the one hand, there are those who never adapted to the mouse and the screen, and continued designing typefaces using pen and ink. Perhaps the medium or perhaps the all too mathematical font formats and font editors acted as deterrents for them. On the other hand, we have seen the emergence of digital artists who design glyphs directly on the screen, and do so with extreme efficiency. In this category, we can place prolific artists such as Lucas De Groot, Jean-François Porchez and David Berlow. A few even mastered the bitmap format, and became the ultimate digital technicians. Matthew Carter's Verdana, an outline font designed and tweaked for optimal screen output, is a prime example of the output of a master digital technician. For more on the designer's perspective, read Hermann Zapf's 1991 book [52]. For both groups of designers, however, the font format came first, and they had to adapt to the technology. Perhaps, in the future, we should ask them for some input, and create a medium in which their freedom is undiminished.

The *engineers* have a say in the matter as they report about the limitations of certain media. Screen renderers, printer specifications and other physical facts limit the format in which fonts are presented in those media. There is a movable boundary defined by the partition of the responsibilities between computer and peripheral device. For example, a "lazy" computer may send a raw font to a printer, and the printer must do all the processing internally to put ink on a page [this is the strategy used in native PostScript printers, for example]. Other media expect a device-specific font format, often a bitmap or

pixel font, adapted to the resolution and device specifications. The onus here is on the computer, not the device. In these cases, font formats are sometimes designed by engineers, who have very little typographic training.

Computer scientists and programmers (software artists) are increasingly important players. The creators of PostScript, Geschke and Warnock, who developed PostScript based on the page description language PDL by John Gaffneyin 1976 [1], and the Type 1 and Type 3 font formats [2, 3], were computer scientists with a graphical vision. PostScript succeeded thanks to its simplicity and flexibility. The influence of Adobe today is in fact largely due to the invention of PostScript. In font software and format design, the computer scientists are largely preoccupied with logical organizations of files and with issues like standardization. This endeavour often carries them away, so, just as with the engineers, this group of people should remain dedicated to the users and the font designers, not the other way around.

Going up the ladder, we find the *vendors, foundries and companies*, whose interests are often commercial, and who by definition are concerned with company reputation, sales volume, market share, proprietary formats, and software strategies. Fonts are often developed as part of larger software packages or in conjunction with certain operating systems. This world also revolves around patents, trademarks and copyrights, the various ways in which software and typefaces may be protected. The actual font format itself that is supported by this group is often the result of various market decisions, the prime example being the story of PostScript, TrueType and OpenType that will be recounted a bit further on.

The final force at work in the creation of a font format is *inertia*, driven by tradition and historical models. An electronic font format is often the result of a modification of a previous format or technology. Backward compatibility is often cited as a requirement for a new format, but this been contradicted by the historic record, with dramatic incompatible quantum jumps in the technology.

The typeface repertoire is rich, with many typefaces existing in only one of several possible formats. Many historic faces only exist in print (in specimen books or old manuscripts), while hundreds if not thousands are only available in metal or wood. In the phototypesetting era — the 1950s to 1980s — , typefaces were stored in photographic format. And finally, in the 1980s, electronic font formats were introduced. Among these, the earliest are the bitmap formats such as "fon", "bdf", and "fnt". In 1982, Jim Warnock and Charles Geschke introduced PostScript [1], and suggested storing glyphs by describing their outlines as Bezier curves. This led to the Type 3 and Type 1 font formats. Knuth also used Bezier curves for outlines, but had the idea of describing glyphs by programs in his METAFONT [30], which was introduced and perfected in the period from 1977–1985. In 1987–1989, Apple's Sampo Kaasila developed the TrueType format, which was an economic decision to counteract the stranglehold Adobe had on the type technology market at the time with its proprietary PostScript. Finally, Microsoft and Adobe joined forces in the 1990s to create OpenType in the hope of reconciling TrueType and PostScript. The discussion below will show that this is only a minor technological step. When we look into the future, we must take this varied historical record into account. The electronic era is the first one in which font formats were proprietary — they were designed and "belonged" to one or more companies. In taking the next step in formats, we should steer clear of this trap, and agree on a route that is open to everyone.

It is very likely that the present computer data model, in which the bits are the atoms, and in which bit storage is somehow achieved at the microscopic physical level, will survive for at least a few decades, so we will use words like files and bits in this paper, with the caveat that a future reader may find this vocabulary old-fashioned. Taking a long-term view, we will describe *the hub model* for font storage and manipulation. The details will be described in subsequent sections.

The hub model

A font is an implementation of a typeface: ideally, it contains the full description of that typeface. It is like a complete book — anyone can read it, nothing is missing, the author is clearly identified, and so on. Similarly, a font should thus be implemented in a human-readable "open book" format. None of the previous formats had this. In the metal days, valuable information about the creative process was missing, and only foundries actually owned metal type. TrueType, Type 1 and OpenType fonts are only computer-readable. METAFONT and Type 3 can only be interpreted by programmers and computer scientists. In fact, because of the proliferation of formats, we have TrueType, Type 1 and OpenType versions for hundreds of typefaces, and each version is slightly different from the other one because of technical incompatibilities. In other words, at present, one typeface "lives on" in many fonts, and this is a chaotic situation.

The human-readable mother font for a typeface should exist once, and ideally be frozen forever, just as with a "version" of a piece of software. Additions and modifications of it then yield new fonts. One can swim downstream from the mother font to popular implementations (TrueType, Type 1, et cetera) by filters, but horizontal swimming between OpenType and Type 1, for example, is not recommended, and upstream conversions are to be avoided at all costs.

Font editors at present include Fontographer (owned

by Macromedia, described by Moye [39], FontLab (by Yuri Yarmola), Font Studio (by Letraset), Ikarus (by Peter Karow at URW), FontForge (by George Williams), FontCreator (by Erwin Denissen), Softy (by Dave Emmett), Manutius (by A. Gebert) and Noah (by Yeah Noah). Each operates on one or more formats on one or more computer platforms. New editors should be designed to create or manipulate that mother font, thus leading to a more logical situation. Artists too should be able to directly access that mother font. Printers, screens, applications, and handheld devices can operate on compact electronic formats obtained downstream from the mother font. It should be noted that most serious editors store fonts in an internal human-readable format, and have in fact created models for mother fonts. Most of these do not go beyond a one-to-one translation of the corresponding binary format, however. For surveys on font technology, we refer to the books by André [6], Karow [26, 27] and Knuth [33] and the articles by Gonczarowski [17, 18] and André and Hersch [7].

Each of the sections below treats one of the aspects of the mother font in more detail.

Outline and pre-outline

One of the main contributions to computational geometry and computer-aided geometric design was the development of the Bézier curve by James Ferguson, an airplane designer, Pierre Bézier, an engineer with Renault, and de Casteljau, an engineer at the competing French automobile company, Citroën. Two and three-dimensional objects could be described and approximated rather simply by concatenating sections of curves. This is, in fact, a way of transforming a physical object into a number of bits, and thus, a way of compaction. One can take a 1MB high-detail photograph or scan of a letter, which after compaction by standard methods such as "zip" (which uses a mix of Huffman and Lempel-Ziv coding) may be reduced to 200 kilobytes or so. Yet, by just storing the collection of Bézier curves, the same letter can be locked in memory using under a kilobyte, as the formula for an n-th order Bézier curve requires just the knowledge of $n+1$ control points $x_0, x_1, \ldots, x_n$ in the plane:

$$x(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \cdot x_i \ , \ 0 \le t \le 1.$$

Here $x(t)$ is a parametric curve, a continuous convex combination of the control points (hence, the Bézier curve stays within the polygon formed by the control points), starting at x_0 and ending at x_n. The mathematical properties of Bézier curves and splines in general are described by Farin [15] and Su and Liu [45].

It was only natural that PostScript and METAFONT adopted the Bézier curve: their creators settled on the cubic Bézier curve ($n = 3$). TrueType uses quadratic Bézier curves ($n = 2$), which was an unfortunate decision, as a quadratic Bézier curve can without loss be transformed into a cubic one (given x_0, x_1, x_2, set $y_0 = x_0, y_1 = (2x_1 + x_0)/3, y_2 = (2x_1 + x_2)/3, y_3 = x_2$, to obtain the cubic control points y_i), but not vice versa. So, Type 1 is downstream from TrueType, yet, cubic approximations are usually heralded as being more compact than quadratic approximations. Artists report that cubic curves have a richer palette than quadratic curves.

Bézier curves cannot represent circles without error, no matter how large n is [for the mathematically inclined, this is an excellent exercise]. For example, a 90-degree circle arc is best approximated by a cubic Bézier if we take the control points $(0,1), (a,1), (1,a), (1,0)$, and $a = (4/3)(\sqrt{2}-1) = 0.5522847498\ldots$. This omission could have been rectified if Bézier had allowed parametric descriptions involving either a square root or a trigonometric function.

Type designers who work with type on screen are in fact Bézier point placement artists. Their instrument is the mouse. This is very hard, as many control points are not on the curves, and continuity of derivatives between adjacent Bézier sections is difficult to achive by the naked eye. Some designers still use pen and paper, and rely on scanners for computer input. Yet others, used to software for artists, are good at placing points that are related to Bézier curves indirectly, such as demonstrated by Böhm splines [10], where smooth continuous derivative Bézier sections are implied.

Hobby [23] and Knuth [30] developed an algorithm for constructing a sequence of Bézier curves that is forced to visit the designer's set of points. This algorithm is built in METAFONT [30, p. 131], and can be a great on-line tool for some. We call such ways of describing outlines "pre-outlines".

To summarize, the mother font should be flexible and permit choices between any of a number of outline and pre-outline formats, as long as each format defines a mathematical curve in a unique manner. Concatenations of Bézier curves of any degree (with n being a parameter) should be allowed, as well as several pre-outline formats to accommodate the typographers at large. At least one spline model should be included that stores circle arcs without any error, so that we can finally have exact approximations of those fantastic geometric ruler and compass creations of masters like Philippe Grandjean, the designer of the Romain du roi (1693–1745).

Ink models

The two main formats, TrueType and Type 1, and their derivative, OpenType, are all based on a primitive ink model, based on the principle that a character is defined

by a number of closed outlines, which are then filled with ink according to the non-zero winding number rule. This means that if a point or pixel is in a given region, then its color, black or white, can be determined by drawing a ray from that point to infinity (in any direction!) and keeping a weighted count of the outlines crossed. A weight of one is given to a clockwise turning contour and minus one to a counter-clockwise contour at each crossing point. But this is clearly not how we place ink on paper at home, where overwriting and erasing are two primary operations. Also, one should be able to work with many black/white images, perhaps levels of images, and define a final image as a logical operation on component images, using operators like "or", "and", "exclusive or", and "not". One should be able to mark a region black or white by pointing to it — in other words, the region containing "x" should always be black.

Stroke fonts are distinguished from outline fonts by their ink model: a stroke is defined, perhaps by a collection of splines of Bézier curves, and ink is placed by following the stroke with a brush or nibbed pen, perhaps tilted at an angle or suitably shaped. Japanese and Chinese seem like prime territory for such fonts. But closer to home, we should not forget about the characters that are created by the interaction between a pen and a tablet, as on palm-held devices, or signatures made with a magnetic pen, or input from a computer tablet. A person's handwriting is often better captured by letting the person write on a tablet (so that we obtain the stroke points in chronological order, with dynamic information), as opposed to scanning the individual's handwriting. Tablet input is naturally translated into strokes.

The recommendation to allow many ink models sounds like an extension of the PostScript graphical model, but it can be organized by rasterizers and printers without too much trouble as all can be internally reduced to outlines (out of sight of the font designer!) and to the classical non-zero winding number rule. The extension is suggested, once again, to make type design easier, more universal, more current and more accessible.

Path complexities

Outlines and curve data are not unrestricted in our present electronic formats. For example, paths in PostScript and thus Type 1 are limited to about 750 control points. Such limitations make it impossible to store certain complex characters as are found in ornaments, decorative initial caps, and outlines based on high resolution scans. TrueType has higher limits, but the mother font should in principle have no limit. Limits could be introduced by various formats downstream, and by various viewing media even further downstream, but it should not be introduced at the mother font level.

Accuracy

Outlines in any form require mathematical input. As points need to be represented in a unique manner across all platforms, it is imperative that all mathematical descriptions be in terms of integers. For example, a point can simply be (x, y), where x and y are integers, but it can also be $(x/x', y/y')$ where x, x', y, y' are integers, so that we can attain all rational numbers. At present, assuming that a character occupies the square $[0, 1]^2$, points in that square can be addressed as $(x/1000, y/1000)$ with x, y integer, as is common in Type 1. Type 1 permits higher values than 1000, but not all interpreters of Type 1 fonts are happy with such. In TrueType, the 1000×1000 box is replaced by 2048×2048. The different box sizes shows that there is no lossless horizontal conversion between TrueType and Type 1, as $x/1000 = y/2048$ implies that x must be a multiple of 125 and y a multiple of 256, and any other values imply a loss in accuracy. OpenType inherits the Type 1 restriction for its CFF style implementation, and the TrueType restriction otherwise.

It is incomprehensible that no one has even attempted to increase these limits of accuracy. Picture a complex character consisting of 50 rows and 50 columns of circles that touch other. In a 1000×1000 integer box, this would force the radius of each circle to be 5. In a cubic Bézier implementation of a quarter circle, we need to place the control points at $(0, 5), (a, 5), (5, a), (5, 0)$, and must select the values $1, 2, 3$ or 4 for a, recalling that the ideal value is about 2.75 (see above). By picking $a = 3$, the circles will be far from perfect!

There is an even more compelling reason why the accuracy must be increased: the historical record. As we scan historical designs, in our quest to store everything in some electronic format for the future, we must ensure that as little as possible is lost in the process. Just as the noise in old LPs was due to mechanical limitations, so is the noise introduced by storing valuable designs using less-than-ideal accuracy. Reconstruction and de-noising will be difficult once the damage is done.

In a 1000×1000 box, storing a point (x, y) requires about 20 bits. In a 1 million by 1 million box, the storage increases to about 40 bits, and for an unimaginable 1 billion by 1 billion box, the storage increases to about 60 bits. Thus, by doubling the storage requirements, we can in fact increase the number of point positions by a factor of one million! By tripling, that multiplication factor becomes one trillion. In other words, this is a change that comes relatively cheaply. Furthermore, since the mother font is upstream of everything else, one can always drop down to lower accuracies when moving downstream. For storing points, perhaps the best method is to work with (x, y, n), where n is the accuracy, and x and y are inte-

gers in or near the range $[0 \ldots n]$. The triple then represents $(x/n, y/n)$. The value of n should not a priori be restricted. Accuracy should be a variable parameter, perhaps different from font to font.

It must be mentioned that accuracy is not an issue in a pure PostScript type format such as Type 3, and that theoretically, in a Type 1 font, it can be controlled by the Font Matrix, although, in practice, many applications expect a 1000×1000 matrix.

Programming and fonts

The current crop of electronic font formats are just tables. Just as with their metal counterparts, they are dead objects that require manipulation by an external master or computer program. Even though some companies claim that their fonts are programs, this is false, with the exception of METAFONT and Type 3, which were both major steps forward in font technology. In addition, some TrueType fonts have some bits of code in their hinting sections, but it is debatable whether this should be considered as a program or a table.

The Type 3 format allows the use of the full PostScript language: there are parameters, variables, conditional instructions and loops. It is possible to make randomized fonts, e.g., for the simulation of handwriting, and to create connected context-sensitive glyphs. Characters can be programmed in terms of tunable parameters. Perhaps the simplest tunable fonts are the multiple master fonts that Adobe proposed in the 1990s, in which one can vary one or more parameters to interpolate between extremal fonts. Of course, this can be emulated in Type 3 fonts. METAFONT has similar capabilities, and, in fact, Knuth demonstrated with his Computer Modern family [32] that one program per glyph suffices to create a family of 72 component fonts, ranging from typewriter type to serif and sans serif (see also [19]). Other attempts at parametrization, such as Infinifont (McQueen and Beausoleil, [38]) and LiveType (Shamir and Rappoport, [42, 43]) were short-lived.

The disadvantage of such programmable fonts is the necessity to have at one's fingertips, in printers, and in applications, powerful interpreters or on-the-fly converters to other formats. Furthermore, the danger of a virus lurks in every piece of code — indeed, executing a Type 3 "font" can have as side effect the creation or deletion of one or more files. Finally, interpreters for powerful languages are often legally protected and can only be licensed at enormous fees. With language features wisely restricted to purely mathematical and graphical operations, one should be able to flag mother fonts that contain active code, analogous to the present flagging of multiple master fonts.

Reviving the idea of programmable fonts will have enormous benefits for mathematical typesetting. Knuth's model (METAFONT + TEX, [31]) is now over 20 years old, and has a few shortcomings that require an update. There should be a continuum of optically adjusted symbols like brackets and parentheses, with line thickness and size adapted to the surrounding text. At present, the symbols are selected from a finite set, which often leads to aesthetic mismatches. Improvements should be made in optical size matching of subscripts and superscripts.

Of course, optically and continuously adjusted symbols are only part of my mathematical typesetting wishlist. There should ideally be a symbiosis of figures, formulas and text, all playing and interacting on the page, a bit as with blackboard mathematics in the hands of a master mathematician. This requires a paradigm that transcends TEX.

In the area of randomized fonts for the simulation of handwriting, we refer to Devroye and McDougall [13] for a theoretical development and some crude examples, to Desruisseaux [12] for a thoroughly researched font called MetamorFont, and to André and Borghi [5], Dooijes [14] and van Blokland and van Rossum [50] for earlier attempts in this direction. All these developments used the programming power of Type 3 to create random-looking characters that are either based on a sample of one's handwriting (as in the first reference above) or that are constructed artificially by programming the randomness in the outlines (as in MetamorFont). It would be a shame not to include a random number generator in the specification of the mother font. Of course, one should make sure that the random sequence generated can be "replayed" for debugging purposes.

Ligatures and context sensitivity

Ligatures are combinations of two or more characters. Context sensitive characters are single characters that change shape as a function of their context or neighborhood. The activation of a context sensitive change should always be the responsibility of the application — the font should only contain the various shapes without getting involved in questions related to context.

This separation of form and application should also apply to ligatures. Fonts provide the shapes only. This division has been rigorously supported in the METAFONT + TEX model, with TEX taking care of the actual activation of ligatures. In OpenType, a `GSUB` table was introduced that in combination with software such as InDesign will activate ligatures. However, which letters react in what manner is stored in the `GSUB` table, so that the separation is less clear, forcing the font designers to worry about non-artistic issues, and thus making the design process too hard. Artists can hardly be expected to design `GSUB` tables!

Arabic requires a large number of ligatures for proper typesetting (see, e.g., Smitshuijzen AbiFarès,

[44]). However, a large number may also be required for Latin handwriting. The author has experimented with ligatures in an interesting way, creating glyphs in Type 3 with a tablet for about 1600 ligatures. These consisted of the most popular pairs of letters, with a distinction between starting pairs in words, ending pairs, and mid-word pairs. In addition, triples were added, again by popularity as measured in a large body of text. Finally, single letters came in three forms, starting letters, sentinels, and mid-word letters. Combinations of capitals with one or two trailing lower case characters were also thrown into the collection. Given a text, a small program decided on the optimal composition of a word using these ligatures thanks to a formula based on rewards and penalties. Others can improve the typesetting by changing that parsing program, without touching the font file, keeping the activation of the ligatures away from the fonts.

Bitmaps and images

The preservation and restoration of old typefaces if done in outline format requires an increased accuracy. Nevertheless, at some point, a crucial transformation from bitmap or image to outline is necessary. This process is often called tracing or auto-tracing. Algorithms for this abound (see Avrahami and Pratt [9], Plass and Stone [40], Itoh and Ohno [24], Gonczarowski [16], Schneider [41], Lejun, Hao and Wah [34], or Mazzucato [37]). However, the perfectionists may wish to keep the original image, rather than the possibly polluted outline. The storage may be prohibitive, but one might want to compress the images by clever lossless (or reversible) compression methods that are designed to look for straight edges and smooth outlines. Such dedicated or "smart" compression methods may yield high compression ratios. The mother font should allow for the storage of bitmaps of extremely fine grain.

It is not far-fetched to project that one day, all fonts will be stored in a compressed bitmap format, with storage capabilities expanding at an enormous pace, and with smart compression an active area of research in information theory. The benefit of such a format is that the design of a font editor will be much easier, while the editing process itself will feel more natural to the typographers. In fact, paper and electronic format will converge again.

Standardization and coding

The effort to standardize the naming of symbols and the positioning (or: coding) of symbols by attaching permanent numbers to each of them, should continue. Unicode has changed the typographic scene in this respect, but it is unrealistic to expect each font to be "complete", using whatever definition of "complete" one wants. For one thing, new symbols are invented daily, so that the standardizers will never be able to keep up. Furthermore, special unique and innovative glyphs add to the value of a typeface, especially if no other typeface offers them. It is in the human nature to create and invent, and thus, the mother font should not be tied to one particular coding scheme. It could be flagged as Unicode-compliant or Unicode-subset-compliant, but in the matter of encoding and naming, we cannot the predict the future — who could have predicted the Euro symbol in 1970 —, and therefore have to recommend that mother fonts be unrestricted.

The number of glyphs in one font should not a priori be limited. Each glyph should have a name and an integer-valued position, but the maximal value among those integers should have no obvious bound, not even the seemingly large bound that comes with Unicode.

Font information

One of the key components of the mother font relates to the information and history of the typeface and the font. Each font or typeface has a genealogical history. There is an ancestral tree or dag (directed acyclic graph) that explains the present. The tree should be shown, and each node and link in it explained and if possible, dated. It is a pipedream to think that one can have a permanent font information depository somewhere. The best we can hope for is to make the font information an essential part of the mother font. In many cases, the ancestors can and should be traced back to the days of metal type.

Font names should be unique, perhaps by introducing foundry letters and short version numbers in the font name. In no case should information about the font be separated into a "readme" file, another invention of the eager computer scientists. The font information should explain the absolute and earliest origins of the typeface. It should then report on the changes, revivals, additions, and extensions that have transformed the original typeface into this font. Clearly, this information can be erased and fraudulently altered, but no practical system will prevent this. At present, many foundries such as Adobe and Linotype do not mention the typographer who created the typeface anywhere in or near the font. They offer biographies of their designers on web pages that may one day disappear while their information-starved fonts survive. Thus, the information field should be used to pay a permanent tribute to the creators, typographers and artistic forefathers.

Human-readable format

The mother font has to exist in a simple human-readable form. For TrueType and OpenType, the TTX tool by van Blokland and van Rossum [51] permits a one-to-one transformation between the binary font file and a human-

readable XML file. Other examples of such mapping programs exist for other formats. Non-commercial formats such as METAFONT essentially exist only in text format.

Motivated by the simple requirement that anyone, even a person without appropriate software, or without software attached to a certain decade, can read and interpret the instructions, all font information and all outlines must be readily accessible. Adding an accent or dieresis to a character should be a trivial operation. And importantly, even moderately capable programmers should be able to write simple code to act upon the mother font to achieve a certain effect. For all these reasons, a binary model should be excluded. Those who argue that the storage may be prohibitive should be reminded that fonts can be compressed on the fly when sent over a network or to a device.

Automated operations

In any typeface, metric and kerning information is essential. Kindersley [28] has provided nice ideas on how letters should be spaced. At URW in [47, 48, 49], an attempt was made at automating character spacing through internal programs cryptically called hz and Kq. The choice of spacing around each character requires a certain amount of expertise, and a well-kerned font is out of reach of most typographers. Therefore, the mother font should have flags that indicate the automation of the process of determining the sidebearings of the characters and the kerning between all pairs of glyphs. And if set, another parameter could be used to select an algorithm from a collection of possible algorithms, with further parameters left to the user's choice. The kerning algorithms should be unambiguously defined, but not in a programming language. In this manner, automation and hand-kerning can coexist, and one can override the other.

Hinting is uniquely tied to electronic fonts, as earlier formats were not concerned with discretized media. It too can be dealt with in the way suggested above for kerning, via the setting of a parameter which selects one of the built-in hand-kerned data sets, or one of the automated algorithms. Examples of the latter are described by Karow [25], Andler [4], Hersch and Bitrisey [21] and Herz and Hersch [22]. An argument could be made to exclude hinting altogether from a font, and insist that it is the responsibility of the printing or screening device.

Addendum

Since the paper was first written in 2003, we have become aware of some independent attempts at text-based mother formats for font sources. These include the UFO (unified font object) file format introduced in October 2004 at the ATypI meeting in Prague by the Letterror people, Erik van Blokland and Just van Rossum. Furthermore, George Williams's internal SFD (spline font database font format) for his FontForge font editor also constitutes an attempt in this direction.

References

[1] Adobe Systems, *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1990a.

[2] Adobe Systems, *Adobe Type 1 Font Format*, Addison-Wesley, Reading, MA, 1990b.

[3] Adobe Systems, *Adobe Font Metric Files Specification Version* 3.0, Adobe, 1990c.

[4] S. F. Andler, "Automatic generation of grid-fitting hints for rasterization of outline fonts", in: *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography, Gaithersburg, Maryland, September* 1990 (edited by R. Furuta), pp. 221–234, New York, 1990.

[5] J. André and B. Borghi, "Dynamic fonts", in: *Raster Imaging and Digital Typography* (edited by J. André and R. D. Hersch), pp. 198–204, Cambridge University Press, Cambridge, 1989.

[6] J. André, "Création de fontes et typographie numérique", IRISA, Campus de Beaulieu, Rennes, 1993.

[7] J. André, "An introduction to digital type", in: *Visual and Technical Aspects of Types* (edited by R. D. Hersch), pp. 56–63, Cambridge University Press, Cambridge, UK, 1993.

[8] J. André, "Ligatures & informatique", *Cahiers GUTenberg*, vol. 22, pp. 61–86, 1995.

[9] G. Avrahami and V. Pratt, "Sub-pixel edge detection in character digitization", in: *Raster Imaging and Digital Typography II* (edited by R. A. Morris and J. André), pp. 54–64, Cambridge University Press, Cambridge, 1991.

[10] W. Böhm, "Cubic B-Spline curves and surfaces in computer-aided geometric design", *Computing*, vol. 19, pp. 29–34, 1977.

[11] W. Böhm, G. Farin, and J. Kahmann, "A survey of curve and surface methods in CAGD", *Computer-Aided Geometric Design*, vol. 1, pp. 1–60, 1984.

[12] B. Desruisseaux, "Random dynamic fonts", M.Sc. thesis, School of Computer Science, McGill University, Montreal, Canada, October 1996.

[13] L. Devroye and M. McDougall, "Random fonts for the simulation of handwriting", *Electronic Publishing (EP—ODD)*, vol. 8, pp. 281–294, 1995.

[14] E. H. Dooijes, "Rendition of quasi-calligraphic script defined by pen trajectory", *Raster Imaging and Digital Typography*, in: *Raster Imaging*

and Digital Typography: Proceedings of the International Conferences, Ecole Polytechnique Fédérale, Lausanne, Switzerland, October 1989 (edited by J. André and R. D. Hersch), pp. 251–260, Cambridge University Press, Cambridge, 1989.

[15] G. Farin, *Curves and Surfaces for CAGD, A Practical Guide*, Academic Press, New York, 1993.

[16] J. Gonczarowski, "A fast approach to auto-tracing (with parametric cubics)", in: *Raster Imaging and Digital Typography* (edited by R. A. Morris and J. André), vol. 2, pp. 1–15, Cambridge University Press, Cambridge, 1991.

[17] J. Gonczarowski, "Industry standard outline font formats", in: *Visual and Technical Aspects of Types* (edited by R. D. Hersch), pp. 110–125, Cambridge University Press, Cambridge, UK, 1993.

[18] J. Gonczarowski, "Curve techniques by autotracing", in: *Visual and Technical Aspects of Types* (edited by R. D. Hersch), pp. 126–147, Cambridge University Press, Cambridge, UK, 1993.

[19] Y. Haralambous, "Parametrization of PostScript fonts through METAFONT — an alternative to Adobe multiple master fonts", *Electronic Publishing (EP—ODD)*, vol. 6, pp. 145–157, 1993.

[20] Y. Haralambous, "Tour du monde des ligatures", *Cahiers GUTenberg*, vol. 22, pp. 87–100, 1995.

[21] R. D. Hersch and C. Bitrisey, "Model-based matching and hinting of fonts", *ACM Computer Graphics*, vol. 25, pp. 71–80, 1991.

[22] J. Herz and R. D. Hersch, "Towards a universal auto-hinting system for typographic shapes", *Electronic Publishing (EP—ODD)*, vol. 7, pp. 251–260, Special issue on Typography, John Wiley, 1994.

[23] J. D. Hobby, "Smooth, easy to compute interpolating splines", *Discrete Computational Geometry*, vol. 1, pp. 123–140, 1986.

[24] K. Itoh and Y. Ohno, "A curve fitting algorithm for character fonts", *Electronic Publishing (EP—ODD)*, vol. 6, pp. 195–205, 1993.

[25] P. Karow, "Automatic hinting for intelligent font scaling", in: *Raster Imaging and Digital Typography: Proceedings of the International Conferences, Ecole Polytechnique Fédérale, Lausanne, Switzerland, October* 1989 (edited by J. André and R. D. Hersch), pp. 232–241, New York, 1989.

[26] P. Karow, *Digital Typefaces*, Springer-Verlag, Berlin, 1994a.

[27] P. Karow, *Font Technology*, Springer-Verlag, Berlin, 1994b.

[28] D. Kindersley, *Optical Letter Spacing for New Printing Systems*, Wynkyn de Worde Society, distributed by Lund Humphries Publishers Ltd., 26 Litchfield St. London WC2, 1976.

[29] D. Kindersley and N. Wiseman, "Computer-Aided Letter Design", *Printing World*, pp. 12–17, 1979.

[30] D. E. Knuth, *The METAFONT book*, Addison-Wesley, Reading, MA, 1986a.

[31] D. E. Knuth, *The TEXbook*, Addison-Wesley, Reading, Mass, 1986b.

[32] D. E. Knuth, *Computer Modern Typefaces*, Addison-Wesley, Reading, Mass, 1986c.

[33] D. E. Knuth, *Digital Typography*, Cambridge University Press, 1999.

[34] S. Lejun, Z. Hao, and C. K. Wah, "FontScript — A Chinese font generation system", in: *Proceedings of the International Conference on Chinese Computing (ICC94)*, pp. 1–9, 1994.

[35] C. W. Liao and J. S. Huang, "Font generation by beta-spline curve", *Computers and Graphics*, vol. 15, pp. 527–534, 1991.

[36] J. R. Manning, "Continuity conditions for spline curves", *The Computer Journal*, vol. 17, pp. 181–186, 1974.

[37] S. Mazzucato, "Optimization of Bézier outlines and automatic font generation", M.Sc. thesis, School of Computer Science, McGill University, Montreal, Canada, 1994.

[38] C. D. McQueen III and R. G. Beausoleil, "Infinifont: a parametric font generation system", *Electronic Publishing (EP—ODD)*, vol. 6, pp. 117–132, 1993.

[39] S. Moye, *Fontographer: Type by Design*, MIS Press, 1995.

[40] M. Plass and M. Stone, "Curve-fitting with piecewise parametric cubics", *Computer Graphics*, vol. 17, pp. 229–239, 1983.

[41] P. J. Schneider, "An algorithm for automatically fitting digitized curves", in: *Graphics Gems* (edited by A. S. Glassner), pp. 612–626, Academic Press, San Diego, CA, 1990.

[42] A. Shamir and A. Rappoport, "Extraction of typographic elements from outline representations of fonts", *Computer Graphics Forum*, vol. 15(3), pp. 259–268, 1996.

[43] A. Shamir and A. Rappoport, "LiveType: a Parametric Font Model Based on Features and Constraints", Technical Report TR-97-11, Institute of Computer Science, The Hebrew University, 1997.

[44] H. Smitshuijzen AbiFarès, *Arabic Typography*, Saqi Books, London, 2001.

[45] B.-Q. Su and D.-Y. Liu, *Computational Geometry–Curve and Surface Modeling*, Academic Press, Boston, 1989.

[46] Unicode Consortium, "Unicode", `http://www.unicode.org`, 2003.

[47] URW, "Kerning on the Fly", Technical Report, URW, 1991.

[48] URW, "Phototypesetting with the URW hz-program", Technical Report, URW, 1991.

[49] URW, "Phototypesetting with the URW Kq-program", Technical Report, URW, 1991.

[50] E. van Blokland and J. van Rossum, "Different approaches to lively outlines", in: *Raster Imaging and Digital Typography II* (edited by R. A. Morris and J. André), pp. 28–33, Cambridge University Press, Cambridge, 1991.

[51] E. van Blokland and J. van Rossum, "TTX", `http://www.letterror.com/code/ttx`, 2002.

[52] H. Zapf, *Classical Typography in the Computer Age*, Oak Knoll Books, 1991.

Polices d'apprentissage de l'écriture ou «polices de cahier» : vers toujours plus de simplicité ?

Jef Tombeur
29, rue de l'Échiquier
75000 Paris
jtombeur@noos.fr

Abstract

"Teacher fonts" fall into various categories. Some are used to devise handwriting practice sheets, others to print material for young readers; some designers claim that some fonts are better suited for homework or corrective exercises for handwriting-impaired pupils — or adults, as quite a lot of "handwriting repair fonts" focus on this market. For many, "teacher fonts" evoke decorative, ornate, cursive (close to what the French call "écriture anglaise") or script (such as so-called "architect lettering") fonts. The decision of the French *Ministre de l'éducation nationale* to introduce approved handwriting fonts in schools gives us the opportunity to examine some of these teacher fonts which one might describe as "Latin writing". Designed to help children imitate the handwriting of clerks of the past, some are now supposed to help school children and students acquire a fluid and quick handwriting for note-taking. The shift from fonts meant to please to fonts made to improve handwriting (either in class or on one's own) is not a recent development. We will try to categorize these fonts and then close with this question: Could recent technology based on script writing recognition imply a future shift of focus for designers of such fonts? That is, will the perception of the "proper instruction" be influenced by technology?

Résumé

Les polices d'apprentissage de l'écriture relèvent de diverses catégories. Certaines sont destinées à élaborer des exercices scolaires, d'autres à faciliter l'acquisition de l'écriture par la lecture, et certains dessinateurs de caractères s'adressent au public des parents voulant faire progresser leurs enfants ou même aux adolescents et adultes désireux de corriger ou améliorer leur écriture manuscrite. Pour beaucoup, ces «polices de cahier» sont encore synonymes d'écritures élégante, décorative, comme l'écriture cursive dite anglaise, ou scripte, des plans d'architecture. La décision du ministère français de l'éducation de doter les établissements scolaires de polices destinées à faciliter la fluidité de l'écriture et la prise de notes donne l'occasion de s'intéresser à quelques unes de ces polices du groupe des écritures dites latines. D'abord conçues pour que les élèves puissent reproduire l'écriture des employés aux écritures ou des fonctionnaires, les polices plus récentes ont d'autres visées pédagogiques, et cherchent moins à plaire qu'à faciliter l'acquisition dirigée ou l'auto-apprentissage de l'écriture. Nous avons tenté de départager ces polices et de nous interroger : leur évolution dépendra-t-elle de celles nouveaux procédés de reconnaissance optique de l'écriture et de la création de polices optimisées pour l'affichage sur les écrans ? Le futur ductus «correct» sera-t-il influencé par l'évolution technique ?

Sur le site *The Teacher's Parking Lot* [1] de Joanne Lindgren, vous trouvez une cinquantaine de polices ou familles de polices destinées à l'apprentissage de l'écriture. Elles sont d'origines américaine ou européenne, et toutes ou presque sont des partagicielles ou des graticielles. Cette recension est loin d'être exhaustive. Car de nombreuses fonderies majeures proposent des polices encore plus élaborées, dues parfois à des créateurs renommés, et des sites spécialisés commercialisent des systèmes complets d'apprentissage comprenant certes des polices, mais aussi des logiciels, ou des macrocommandes pour MS Word, afin d'en tirer tout le parti possible … Il est donc pratiquement impossible de se livrer à un véritable état des lieux. Ces polices, trop nombreuses, sont de même très dissemblables, et certaines ne peuvent guère entrer dans les diverses classifications existantes. Nous avons cependant tenté d'en donner un aperçu …

Pourquoi s'y intéresser ? L'actualité typographique française récente nous l'a tout d'abord suggéré : l'annonce des résultats d'un «*concours public pour la création de polices d'écriture cursive*», destinées à être diffusées dans les écoles, nous y a incité. Deux modèles furent retenus, et présentés officiellement à la presse par le ministère français de l'Éducation nationale, en janvier 2002.

L'avenir proche, espère-t-on, nous dira ce qu'il en adviendra. Nous avons constaté aussi qu'aucun des nombreux ouvrages traitant de la typographie en notre possession ne faisait cas de ces polices modernes. Pourtant, la plupart des livres consacrés à la calligraphie ou à la typographie se penchent généralement sur l'évolution de l'écriture, font état, voire grand cas, des polices historiques inspirées des écritures manuscrites. Sans vouloir le moins du monde le reprocher à l'auteur, l'excellent *Lettres latines* (paru en février 2003, Éditions Alternatives, Paris), en offre un exemple. Laurent Pflughaupt, dans cet ouvrage retraçant l'histoire de chaque lettre de l'alphabet latin, fait le grand saut entre la ronde, «*apparue vers* 1650 (…) *longtemps enseignée dans les écoles*» et les *tags* ou les graffitis. Enfin, il nous semble que consacrer un tant soit peu d'attention au sujet pouvait intéresser ceux qui se consacrent à la saisie de l'écrit sur tablettes graphiques ou d'autres instruments appelés à être plus couramment employés. Et si ce furtif éclairage incite d'autres, plus qualifiés que nous, à explorer ce domaine, nous imaginons que leurs réflexions seront fécondes …

Les polices de cahier

Fig. 1 : La série des polices de la famille des Plume de Christian Verchery, et ses Seyes, incluant des réglures.

Tout d'abord, de quelles polices parle-t-on ? De celles que nous qualifierons, selon une expression ad hoc, de «polices de cahier». Soit des polices scolaires, destinées aux enseignants ou aux parents d'élèves, et non des polices d'édition ayant été conçues pour faciliter la lecture d'ouvrages réservés à un jeune public. Cette distinction est-elle valide ? Oui, s'il faut en croire certains créateurs qui ont ressenti la nécessité de concevoir deux gammes de familles, l'une destinée à faciliter l'apprentissage, l'autre réservée aux éditeurs. C'est le cas des P'tit François, d'Olivier Nineuil et Evelyn Audureau, dont les familles (scriptes et cursives) sont réparties en familles dites «Éducation», qui sont «*conformes aux réglures des cahiers scolaires (lignages «Seyes» fig.* 1 *ou «tunnels» pour l'école maternelle)*» et dites «Livre» («*avec montantes courtes* [et] *interlignage plus serré* [pour] *la composition des textes à lire*»). Chaque famille comporte quatre styles (Medium, Medium Italic, Bold, Bold Italic). Elles ont été diffusées en 1996. En revanche, la Père Castor, fig. 2, de José Mendoza, créée en 1975, et destinée à la composition des livres de la collection homonyme (éditée par Flammarion), avait tout autant vocation à servir d'exemple aux parents pour apprendre l'écriture à leurs enfants. La Père Castor, numérisée en 1996 par Thierry Puyfoulhoux, se compose, en sa version PostScript de type 1, de deux fichiers de polices, l'une rassemblant les caractères courants, l'autre des ligatures, et des formes alternatives pour un meilleur rendu de l'écriture cursive (dite aussi liée ou attachée) naturelle. Il s'agirait donc d'une police mixte.

Fig. 2 : Le «Père Castor», de José Mendoza (1975).

Les choses se compliquent avec les polices composites, pédagogiques et spéciales, vouées à faciliter la composition d'exemples réalisés par les parents ou les enseignants. Au plus simple, nous avons par exemple les polices de Jean-Marie Douteau, lesquelles se répartissent entre polices d'apprentissage de l'écriture (la gamme des polices Écolier, composé de deux polices courantes, l'une à descendantes courtes, l'autre à descendantes longues, avec trois variantes comportant aussi des réglures, une pour les classes préparatoires, deux autres reprenant la différence de longueur des descendantes), et de polices vouées à des exercices de lecture, les Obase et Odumo.

Partant du principe que la partie supérieure des caractères permet de les distinguer, ces polices comportent des lettres dont la partie inférieure n'apparaît pas.

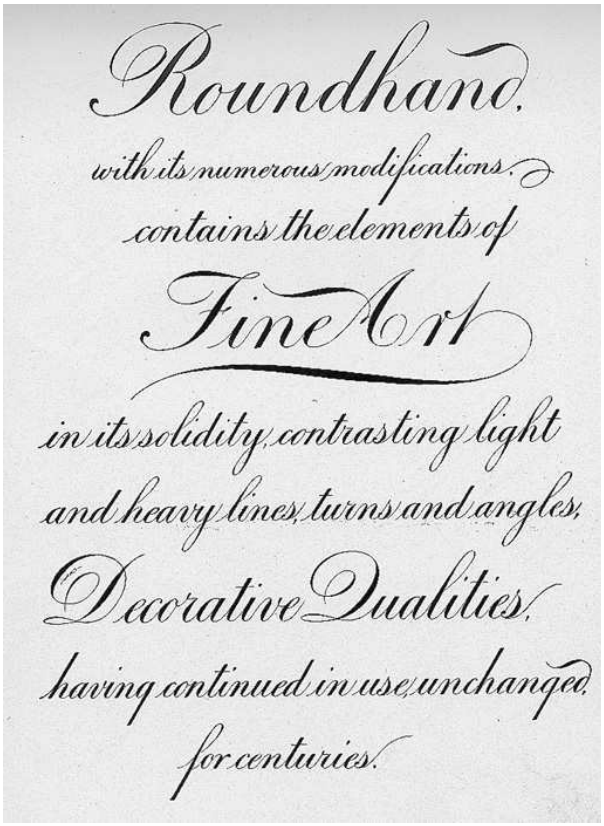


Fig. 3 : En 1888, Charles Paxton Zaner créa un établissement d'enseignement à Columbus, dans l'Ohio. En 1891, Zaner s'associa avec Elmer Ward Bloser. L'écriture commerciale, considérée être l'un des beaux arts, doit être lisible mais aussi décorative. D'autres pédagogues américains, tels Platt R. Spencer ou Austin Norman Palmer, allaient aussi établir des méthodes et faire école. La méthode D'nealian fut introduite dans les écoles américaines à partir de 1965. Sur l'histoire de l'enseignement aux États-Unis, on pourra consulter [24]. Dans les années 1740, la référence en Angleterre était *The Young Clerk's Assistant*, de George Bickham [2].

Les créateurs de telles polices «composites» déploient des trésors d'imagination pour faciliter la tâche des parents et des professeurs des écoles. Il s'agit souvent d'instituteurs, en activité ou retraités. Ainsi, celles de Bernard Vivier (BV-Ronde et autres) s'accompagnent de versions dites boîte (pour imprimer des rectangles dans lesquels l'enfant tracera la lettre). Il a aussi prévu des astuces pour reproduire trois sortes de réglures. Les lettres de ses polices, si composées après avoir utilisé

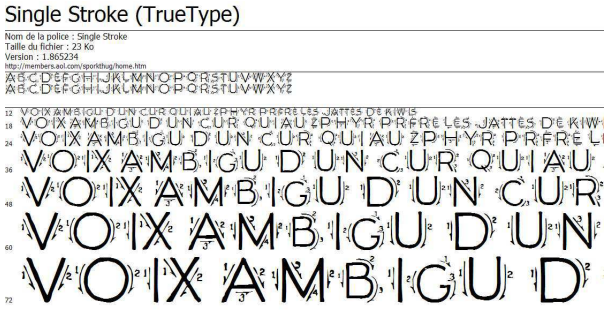


Fig. 4 : De nombreuses polices d'apprentissage, telles les Single Stroke, sont assorties d'indications du ductus (décomposant les étapes de formation du caractères et l'orientation du tracé).

des raccourcis de touches (AltGr + touche peu usitée pour le français), s'inscriront alignées dans les réglures. Vous trouverez ses polices, et d'autres, sur divers sites et pages dont [3]. Le Français Michel Bujardet ([4]) propose aussi des polices assorties des lignes des réglures mais aussi d'indications du ductus. Les contours des caractères sont donc assortis, sur leur pourtour, de petites flèches indiquant le sens du tracé. C'est le cas aussi de deux polices de la famille des New Kindergarten (voir sur [5]) de Bill et Marilyn Andersen. Ou des Single Stroke du Rev. Josh Wilhelm (de Sporkthug Typography), fig. 4, lequel indique qu'il en avait trouvé les exemples d'origine «*dans un très vieux livre de typographie*». Ces créateurs s'efforcent de tenir compte des pratiques scolaires et proposent, comme Christian Verchery, des glyphes alternatifs pour certains caractères (Q et Z capitales en particulier). Autres types de polices, les «pointillistes». Leurs glyphes sont formés de points que l'apprenant devra relier. Ces polices, pour la plupart, ne comportent que des capitales. Des logiciels permettent de leur associer les flèches des indications du ductus. Citons l'un des plus élaborés, StartWrite (commercialisé sur le site [6]), qui permet de faire apparaître ou masquer les flèches, de produire quatre positions de lignes guidant l'élève, et quatre variantes de pointillés, etc. Ce logiciel est livré avec une véritable typothèque dont voici le descriptif : «*These fonts are Manuscript (ball and stick or similar to Zaner-Bloser), Cursive, Manuscript-Simplified, Cursive-Simplified, Modern Manuscript (similar to D'Nealian), Modern Cursive, Italic (similar to Portland Italic), Italic Cursive, Palmer, Palmer Cursive, HWOT (similar to Handwriting Without Tears), HWOT Cursive, Victorian Print (Australia), Victorian Link (Australia), Queensland Print (Australia), Queensland Link (Australia), New South Wales Print (Australia), New South Wales Link (Australia), and five math fonts with assorted pictures for Manuscript, Modern Ma-*

nuscript, and Italic." Nous reviendrons, brièvement, sur les méthodes évoquées (Zaner-Bloser, D'Nealian), par la suite.

Une (très) succincte évocation historique

Fig. 5 : L'alphabet Sütterlin.

Nous ne nous intéressons ici qu'à des polices récentes, mais il convient toutefois d'évoquer rapidement le proche passé. Pour résumer en mode survol, disons que jusqu'aux débuts du xx^{e} siècle, les calligraphes formant des employés aux écritures (commerciale, administrative) font école au sens propre comme au figuré (certains enseignent leur art, forment des disciples, et créent de véritables institutions d'enseignement, et les instituteurs s'inspirent de leurs méthodes). Le souci de rentabilité (simplification des glyphes pour accroître la vélocité d'écriture) ne prédomine pas sur celui d'obtenir un rendu élégant, distingué (l'écriture de l'employé est le véhicule de l'image de marque de l'administration ou de la société). L'écriture doit rester plaisante à lire, même aux dépens de la facilité d'apprentissage. Pour la France, l'un des premiers textes préconisant une méthode très détaillée d'acquisition de l'écriture en milieu scolaire est la *Conduite des écoles chrétiennes* (manuscrit de 1706, publié en 1711 et 1720), de Jean-Baptiste de la Salle. Le chapitre quatre, qui est consacré à l'écriture, prévoit un apprentissage progressif des lettres rondes, puis des bâtardes. Les appellations des caractères (de *compte*, de *finance*, de *minute*, soit gros, moyen et plus petit) donnent une indication de ce qui est recherché : de belles lettres bien formées, élégantes, pour les actes et les livres de commerce.

Nous ne saurions affirmer que, pour les sociétés occidentales, l'Autrichien Ludwig Sütterlin (1865–1917), fig. 5, fut ou non le précurseur d'une tout autre manière d'appréhender l'apprentissage d'une écriture scolaire. L'allemand manuscrit se fondait, jusqu'à environ 1934, sur le modèle d'écriture dite Kurrent, plus ou moins uniformisé, difficile à produire (beaucoup d'angles aigus, nombreux changements de direction). Sütterlin (parfois orthographié *Suetterlin*) introduisit des formes plus arrondies et supprima nombre d'angles fermés. Son modèle fut adopté par les établissements scolaires à partir de 1915. Il commença à régresser après l'abandon des caractères dits Fraktur (par décision autoritaire le Schwabacher Judenschrift fut banni en 1941) mais il était encore enseigné en option jusqu'au milieu des années 1950. Les divers pédagogues préconisant des adaptations radicales des méthodes d'enseignement, ainsi Maria Montessori (1870–1952), qui élabore, vers 1901, des modèles pour l'apprentissage de l'écriture, ont très certainement contribué à envisager autrement la manière de former les glyphes. Johann Heinrich Pestalozzi (1746–1827), John Dewey (1859–1952), Célestin Freinet (1896–1966), lequel introduisit l'imprimerie en tant qu'activité pédagogique, valent certainement d'être cités ici. De même Rudolf Steiner (fondateur des écoles Waldorf). Mais on pourrait tout aussi bien suggérer que les créateurs de polices plus épurées, lisibles, aient pu influer, les nouveaux caractères, aux formes simplifiées, ont pu inciter les pédagogues à s'en inspirer. S'il est avéré que l'imprimerie imita très longtemps l'écriture manuscrite, le profane contemporain pourrait fort bien avancer que l'écriture manuscrite se soit récemment «calquée» sur les caractères imprimés. L'étude des influences réciproques reste encore à approfondir.

Nous avons aussi cité Maria Montessori puisqu'elle donne son nom à une police, la MontessoriScript, du suédois Stefan Hattenbach, créée en 2000 (vous la trouverez chez GarageFonts, cf. [7]). Mais aussi parce qu'elle s'est inspirée de travaux d'aliénistes, dont Jean-Marie Gaspard Itard et Édouard Seguin, et qu'une branche thérapeutique, pendant de l'orthophonie corrective, s'est développée. L'Américaine Nan Jay Barchowsky (cf. [8]) en est l'une des praticiennes les plus connues. La police Barchowsky Fluent Hand, mise au point par la fonderie Eccentrifuge, de John Butler (cf. [9]), a été, à notre connaissance, la première police d'apprentissage au format OpenType. Les fonctionnalités du format permettent d'obtenir à l'écran et l'impression deux types de rendu : écriture script et lettres attachées.

À partir des années 1950 (mise au point d'une nouvelle génération de stylos à bille par le baron Marcel Bich), l'écriture script devient la nouvelle référence. Il y a sans doute un lien entre le perfectionnement du stylo à bille (premier brevet en 1884, nouvelle méthode d'alimentation conçue par Joseph Lazlo Biro en 1938) et l'adoption des écritures script. Pourtant, en France, il ne se répandra vraiment dans les écoles qu'à partir de 1965. En 1959, dans leur *Méthode moderne d'écriture, anglaise-script* [10], les inspecteurs de l'enseignement R. Echard et F. Auxemery font l'éloge de l'écriture script. Depuis l'arrêté du 17 octobre 1945, celle-ci est facultative en

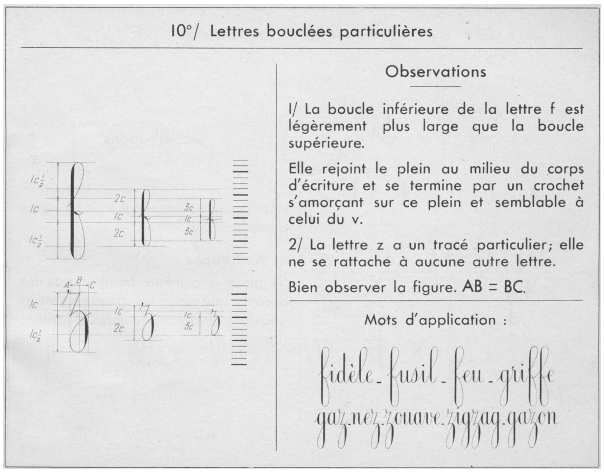


Fig. 6 : En France, l'écriture scripte commença à s'imposer à partir de 1945 dans les écoles. Les manuels faisant autorité, tels ceux des éditions Magnard, tout en la préconisant, accordent la primauté à l'anglaise jusque vers les années 1960–1970.

section préparatoire et en cours élémentaire, mais requise en cours moyen et cours supérieur. Depuis celui du 28 octobre 1947, l'écriture est notée (5 points) sur les copies de l'épreuve de rédaction du Certificat d'études primaires. Echard et Emery plaident pour la script (tracée au crayon de bois et à la plume), en raison de la «*parfaite lisibilité et netteté des caractères*» et de la «*simplicité de la technique graphique à apprendre*». Sa lisibilité serait «*persistante malgré les déformations des lettres dues au manque d'application ou à la personnalité de celui qui écrit.*» Notons que le livret des éditions Magnard (fig. 6) est composé en linéales et que les lettres sont dessinées par Pierre Borowski. Il semble évident que les scriptes sont mieux en rapport, en harmonie, avec les caractères d'imprimerie que les lettres cursives, qui semblent déjà «dater». L'essor de l'Helvetica et autres polices dites suisses, quelles que soient leurs origines, et même «internationales», nous semble avoir plaidé pour l'adoption des scriptes. Chez certains, l'idée à germé que l'apprentissage des scriptes pourrait faciliter la lecture puisque les imprimés adoptaient de plus en plus ces linéales incarnant la modernité. Ce présupposé a été depuis fortement remis en question.

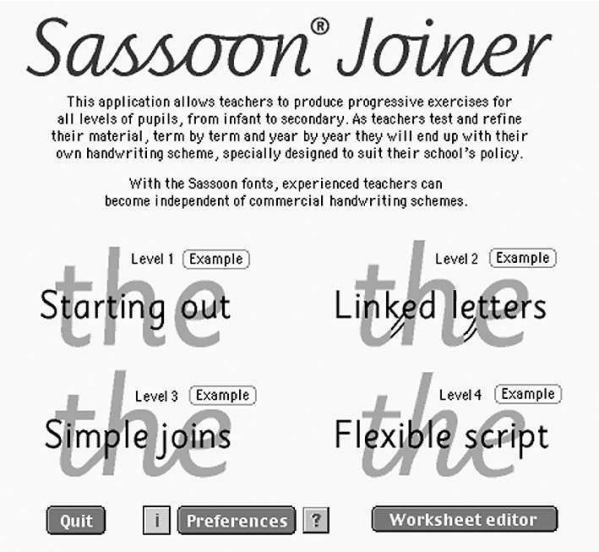


Fig. 7 : Les polices Sassoon se déclinent en familles correspondant à la diversité des préférences des enseignants et des parents.

Ceux qui voudront approfondir et compléter ce très imparfait survol elliptique de ces questions gagneront aussi à s'intéresser à l'œuvre de la typographe Rosemary Sassoon. Le site de référence [11], montre la diversité de ses polices d'apprentissage. Elle a créé des polices évidées (Sassoon Montesori Tracker), des pointillistes (Sassoon Montesori Dotted), des versions *Book* (pour les éditeurs) et *Joiner* (écriture cursive, attachée)„ fig. 7. Un applicatif est aussi fourni pour faire apparaître ou masquer les liaisons entre les lettres. Enfin, un groupe de travail d'enseignants et d'anciens élèves de l'université de Reading tente d'apporter un autre éclairage sur ces questions.

État actuel des lieux

Il serait présomptueux de notre part de tenter de dresser un état actuel des lieux. Surtout en tous lieux, et nous ne doutons pas que, du nord de la Finlande au sud de l'Argentine, une large diversité d'opinions, de méthodes, de polices, peut être trouvée. Les polices finlandaises Tikku, Takku, et Sujuva seraient le point de départ septentrional de cette exploration. Vous les trouverez sur [13]. On trouvera en ligne (cf. [14]) la reproduction d'un article intitulé *De la calligrafia a la tipografia*, de Javier González

Solas, paru dans le numéro 88 de la revue *Visual*. Yann Autret, associé d'Olivier Nineuil (éditions Bonté Divine, s'écria-t-il), dans un récent article [15] qui nous a été signalé par Thierry Bouche, relève l'existence d'une police Me Mima de José Manuel Uros et Joan Barjau (voir sur le site [16]). Sur la page de Luc Devroye consacrée aux polices dédiées aux jeunes publics ([17]), vous trouverez aussi des polices d'apprentissage de l'écriture et des polices pour l'édition de tous les pays.

Nous pouvons, en revanche, en particulier grâce à Michel Bujardet et Françoise Lagarde, apporter quelques lumières sur la manière dont ces polices sont envisagées en France et en Amérique du Nord. Relevons d'abord les similitudes entre les deux continents : une forte latitude d'appréciation est laissée aux enseignants, et il est généralement considéré que la scripte «détachée» est le prélude à l'écriture liée ; et les différences : en France, l'utilisation du stylo plume reste forte, et les éditions Magnard vendent encore bien les ouvrages des collections *Cahiers d'écriture* et *Pages d'écriture* de l'auteur favori du genre, Raymond Grosgurin, qui consacre autant de titres à l'apprentissage à la plume qu'au stylo bille. Il serait utile, pour une approche plus approfondie, d'interroger ces spécialistes que sont Danièle Dumont (rééducatrice en écriture, auteure d'ouvrages aux éditions Hatier), et les auteurs de la méthode *La boîte à lettres* ou *J'écris seul, tu écris seule*, chez Nathan, et d'autres.

Pour Michel Bujardet, typographe, en Amérique du nord, les grandes références sont «*les méthodes Zaner-Bloser, aussi dénommée* Sticks and Circles, *et D'Nealian, caractérisée par la transition bien étudiée entre les* Block Letters *pour les débutants et la cursive pour les plus grands. La méthode* Handwriting without tears *est parfois préférée. Il n'y a pas de directive unique, et les* school districts *mettent en œuvre le programme leur semblant le mieux approprié.*» Relevons aussi que les écoles Waldorf, bien implantées dans le monde anglo-saxon, ont d'autres approches (cf. [18]). Michel Bujardet remarque aussi que les polices d'exemples d'écriture d'origines françaises ou nord-américaines peuvent convenir indifféremment : «*les différences portent sur la forme des chiffres* 1 *et* 7, *et sur les lettres* Z *et* F. *Elles ne sont pas si importantes*». Christian Verchery, dans ses textes d'accompagnement de ses polices, indique d'ailleurs aussi les raccourcis clavier pour obtenir les formes alternatives du 1 et du 7.

En France, depuis longtemps, l'accent est davantage mis sur l'aisance de la production écrite plus que sur son élégance. Cela découle du constat que les résultats scolaires de ceux qui lisent et écrivent avec aisance sont meilleurs que ceux des autres élèves, qu'ils soient ou non appliqués à produire une belle écriture. Si les créateurs d'alphabets français, dont José Mendoza, ont longtemps tenté de sensibiliser l'Éducation nationale à l'opportunité de former les instituteurs à la pédagogie de

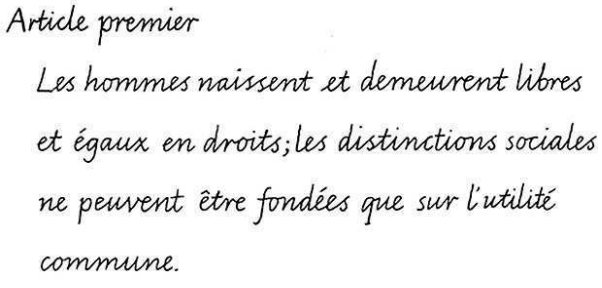


Fig. 8 : L'alphabet de Marion Andrews.

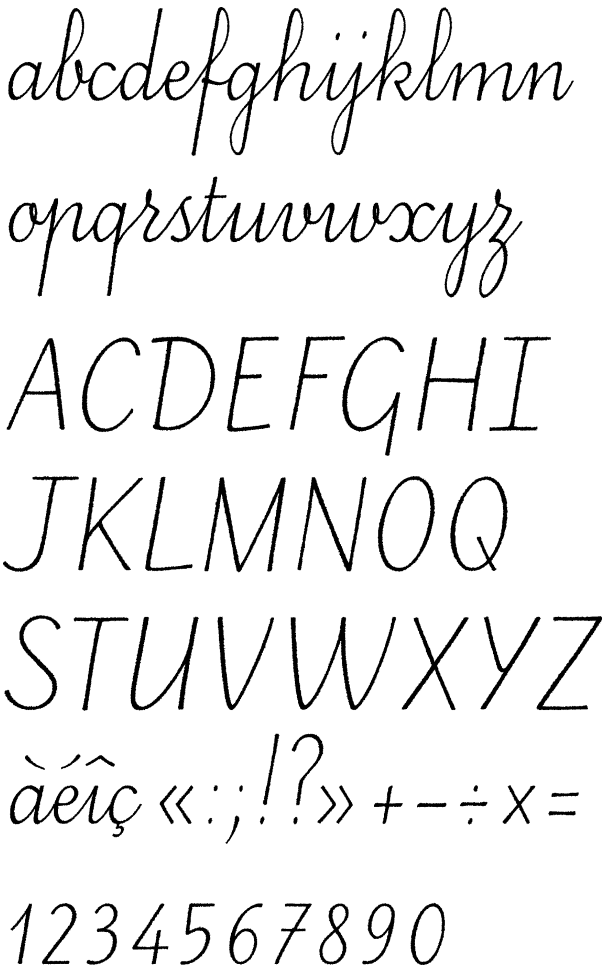


Fig. 9 : L'alphabet de Laurence Bedoin-Collard et Héloïse Tissot, en son état initial. Il a été depuis modifié pour l'harmoniser avec l'autre alphabet retenu par le jury de création du concours du ministère de l'Éducation nationale française, celui de Marion Andrews, plus proche des scriptes.

l'apprentissage de l'écriture en les dotant de polices de référence, c'est à l'initiative d'un proviseur qui connaissait le calligraphe Hassan Massoudi et conseillait le mi-

nistre Claude Allègre qu'il a pu être envisagé de lancer un concours de création de polices de ce type. Le jury, présidé par Jean-François Porchez, réunissait des spécialistes de diverses disciplines et des enseignants. Deux polices ont été primées, dues pour l'une à Marion Andrews, calligraphe, fig. 8, l'autre à Héloïse Tissot et Laurence Bedoin-Collard, fig. 9. Proposer deux polices répondait à la volonté de ne pas imposer un seul modèle (dans le but de «*fournir aux enseignants des écoles maternelles et élémentaires plusieurs exemples d'écritures*», cf. le règlement du concours, [19]) et contribuer ainsi à l'acquisition d'une meilleure dextérité. D'où aussi le choix de cursives, de l'écriture liée, plus propice à l'acquisition d'une vélocité. Jean Hébrard ([20]), inspecteur général de l'Éducation nationale, relève que «*les formes d'écritures héritées des années* 1950 [Ndlr. les scriptes] *et adaptées au crayon à bille et à des papiers rugueux exigent, avec les nouveaux instruments et les nouvelles surfaces, des efforts de freinage du geste que les enfants savent d'autant moins effectuer qu'ils n'ont plus de réelle formation à l'écriture.*» Françoise Lagrange, du Bureau des écoles du ministère, commente : «*Le concours a été motivé par le constat des difficultés observées au collège. Certains élèves peinent à prendre des notes et à rédiger. Ce qui les pénalise. La diffusion de ces polices sera assortie de directives qui ne constitueront pas une contrainte absolue. Le choix de l'une ou de l'autre sera laissé à l'enseignant mais une élève pouvant changer quatre fois de professeur avant l'entrée au collège, il a été souhaité que le passage de la pratique de l'une à l'autre soit facilité.*» Les lauréates du concours ont donc été incitées à harmoniser quelque peu leurs polices.

De possibles prolongements

Il est aussi loisible de s'interroger sur le fait que privilégier l'enseignement de l'écriture scripte répondait aussi à des impératifs administratifs. Devoir remplir de nombreux formulaires exigeant d'écrire chaque lettre dans une case est devenu moins courant : la saisie en ligne de formulaires est devenue fréquente, la reconnaissance optique des caractères tracés à la main a fait de notables progrès. Évoquant de nouvelles surfaces, de nouveaux supports, Jean Hébrard[1], qui s'est aussi intéressé au livre numérique, ne songe certainement pas qu'aux «ardoises magiques» d'hier, mais bien aussi à celles d'aujourd'hui (tablettes graphiques, écrans permettant la saisie directe manuscrite). Ces nouveaux supports devraient sans doute faire évoluer et les polices et les dispositifs (les logiciels et applicatifs incluant des polices, tels Startwrite, déjà évoqué, ou MagicAbc, de la société Mediatic, cf. [23]).

On peut tout imaginer à propos de ces évolutions. Ainsi, il peut être pensé que les progrès de la reconnaissance de la parole rendront moins crucial l'intérêt porté à l'acquisition d'une écriture manuscrite plus fluide. Ou que, puisqu'il s'agit de favoriser la prise de notes sur de nouveaux supports, on en revienne à des procédés proches de l'écriture rapide, sinon de la sténographie. Ce qui supposerait l'adoption de nouveaux caractères, qui pourraient évoquer les ligatures et signes surscrits ou souscrits des abréviations des copistes du Moyen âge. S'il devient admis que la forme écrite couramment et la forme lue aisément puissent être dissociées, que l'acquisition de l'écriture ne doive plus nécessairement être calquée sur l'apprentissage de la reproduction des formes imprimées (donc, du dessin des glyphes les plus fréquemment lus), il est possible d'envisager que les futures polices d'apprentissage de l'écriture ne présentent plus de fortes similitudes avec celles des caractères de composition. L'exercice est vain : non seulement sommes nous conscients des déconvenues que l'avenir réserva aux futurologues, mais il est naïf de fonder un raisonnement sur la seule observation de l'évolution des techniques. Les progrès techniques n'expliquent pas plus à eux seuls l'histoire de la typographie qu'ils ne peuvent, isolément, rendre compte des fluctuations et variations des modes vestimentaires ou du goût gastronomique.

Art des scribes et des lettrés, l'écriture, en se démocratisant, est devenue, comme l'orthographe, pour les beaux esprits du XIX^e^ siècle en France, la «science des ânes». De même, la typographie, autrefois apanage de compagnons et de maîtres, est-elle pratiquée quotidiennement par le plus grand nombre dans les sociétés fortement industrialisées. De même qu'il serait illusoire de croire que maîtriser l'accordéon est beaucoup plus aisé que de savoir jouer du piano, penser que ces polices d'apprentissage, ces polices de cahier, soient des polices mineures, moins dignes d'attention que d'autres, est présomptueux. Souvent considérées marginales, négligées, voire ignorées, elles pourraient, tout autant et même plus que d'autres, gagner en notoriété. Il nous a semblé, ici, adéquat de le signaler.

Références

[1] `http://my.execpc.com/~lindgren/`

[2] George Bickham, *George Bickham's Penmanship made easy, or, The young clerk's assistant*, Mineoloa, NY : Dover Publications, 1997. [1733, Londres, Imprimé pour Richard Ware, sous le titre *The young clerk's assistant, or Penmanship made easy*.]

[3] `http://pragmatice.net/polices/ecrire/mod_exos.htm`

[4] `http://www.schoolfonts.com`

[5] `http://www.kindergarten.com/fonts.html`

[6] `http://drawwrite.com`

1. Sur l'évolution de l'enseignement de l'écriture, voir notamment [21] et [22].

[7] http://garagefonts.com

[8] http://bfhhandwriting.com

[9] http://eccentrifuge.com

[10] R. Echard et F. Auxemery, *Méthode moderne d'écriture, anglaise-script*, Magnard, 1959, Paris.

[11] http://www.clubtype.co.uk/sassmont.html

[12] http://textmatters.com

[13] http://www.kolumbus.fi/jltypes

[14] http://www.ucm.es/info/cavp1/Materiales/caligrafia.pdf

[15] Y. Autret et O. Nineuil, *Aeiou, Revue de littérature pour la jeunesse*, Conseil régional de Champagne-Ardenne et Office régional culturel, n° 2, fév. 2003.

[16] http://www.type-o-tones.com

[17] http://jeff.cs.mcgill.ca/~luc/kids.html

[18] http://www.waldorfhomeschoolers.com/alphabet.htm

[19] http://www.education.gouv.fr/botexte/bo990617/SCOE9900890C.htm

[20] http://www.education.gouv.fr/presse/2002/ecriture/ecrituredp.htm#decouvrir

[21] A.-M. Chartier et J. Hébrard, «La préhistoire d'une discipline scolaire : l'écriture», *Décrire l'écrire*, sous la direction de Jacques Fijalkow, Toulouse, Presse de l'Université Toulouse-le-Mirail et CRDP, 1990.

[22] A.-M. Chartier et J. Hébrard, «Pratiques d'écriture : un éclairage historique», *Écrire et faire écrire*, Actes de l'Université d'été, ENS Saint-Cloud, Octobre 1991, coordonnés par C. Barré de Miniac et D. Bourgain, 1995.

[23] http://mediatic.fr

[24] Tamara Plakins Thornton, *Handwriting in America : A Cultural History*, Yale University Press, New Haven, 1996.

Réformes philanthropiques et réformes orthotypographiques, alphabets artificiels et synthétiques

Jef Tombeur
29, rue de l'Échiquier
75000 Paris
jtombeur@noos.fr

Abstract

It takes all types to make a typographic world. Artificially conceived alphabets fall into various categories that are generally ignored by most font classifications and even the catch-all non-roman or non-latin subdivision does not really account for them. It could be said that they are "special purpose" fonts, but the only special purpose a graphic designer could have for them would be to compose some nonsensical "jabberwocky" filler text such as the famous *lorem ipsum dolor*. Some aim to become a substitute for the latin alphabets, especially for transcribing the English language and simplifying its spelling (or even its pronunciation). Others are meant for writing artificial languages, for linguistic research or virtual worlds. Or they are the result of a purely formal, graphic, artistic endeavour. It may seem odd to pay much attention to such eccentric alphabets. But who knows? Contemporary writing forms are the result of a long evolution; who can say for sure that their current state is fixed and will no longer shift or change dramatically.

Résumé

Il faut de tout pour faire un monde. Les alphabets artificiels ou synthétiques sont multiformes et peuvent être rangés sous diverses catégories généralement ignorées des classifications typographiques ; même la sous-catégorie des polices non-latines n'en tient pas vraiment compte. Ils pourraient être qualifiés de polices «à destination spéciale» mais le seul usage qu'un graphiste pourrait en faire serait de composer du faux-texte tel l'omniprésent *lorem ipsum dolor*. Certains de ces alphabets ambitionnent de se substituer aux polices latines, en particulier pour transcrire la langue anglaise et en simplifier l'écriture, voire la prononciation. D'autres sont destinés à la recherche linguistique, ou à doter des mondes virtuels de langues écrites. Enfin, il peut s'agir encore de tentatives purement formelles, graphiques, artistiques. Il peut paraître farfelu d'accorder une telle attention à ces alphabets excentriques. Mais qui peut présager du futur ? L'écriture actuelle est la résultante d'une longue sédimentation ; qui pourrait affirmer qu'elle est définitivement figée, sans qu'aucune évolution minime ou radicale ne puisse plus jamais se produire ?

L'élaboration de nouveaux alphabets dont les glyphes, voire les caractères, seraient totalement nouveaux, s'écartant de toutes formes connues jusqu'à présent répond à des nécessités ou des volontés diverses. Elles sont de multiples natures. Ces tentatives de créer de toutes nouvelles polices peuvent être simplement ludiques ou artistiques, qu'elles soient utilitaires ou non. Mais aussi d'ordre pédagogique, de la recherche universitaire, ou relever de la propagande, de considérations aussi variées que multiples. Sous la dénomination d'alphabets artificiels ou synthétiques, nous désignons ici des polices de caractères créées de toutes pièces, ce qui exclut celles qui reproduisent les caractères des écritures courantes, ajoutant quelques caractères nouveaux (cas, par exemple, au plus simple, de nouveaux signes de ponctuation, tel le point d'ironie) ou en modifiant légèrement les glyphes pour transcrire certains sons d'une langue (cas des macrons ajoutés aux caractères latins pour transcrire la maori, d'accents adjoints à divers caractères pour d'autres langues polynésiennes, etc.). Cependant, comme on le verra, certains alphabets de réforme sont assez peu innovants en matière de création de caractères vraiment nouveaux. Sont aussi exclus les alphabets de création récente ou plus ancienne conçus pour doter d'une expression écrite des groupes humains ne disposant pas de système d'écriture, même s'il en sera cependant question. Ainsi, la transcription du tifanagh par la police Aligourane de Pierre di Scullio ne rentre pas dans notre catégorie. Les contours sont de cette catégorie sont flous et il serait bien hasardeux de tenter une classification sur le mode de celles de Thibaudeau, Vox, et ultérieures.

Nous nous sommes donc bornés, faute d'une d'avoir pu nous livrer à une recension exhaustive de tous ces alphabets, à distinguer deux ensembles, le premier assez délimité, soit celui des alphabets de réforme de l'orthographe, le second très vaste, les autres. Pour le premier ensemble, notre intérêt s'est porté essentiellement sur le, ou plutôt les, alphabets shaviens, et sur le (et

Fig. 1 : Des projets expérimentaux permettent de générer des alphabets artificiels. Le projet Alphabet Soup (de Matt Chisholm, cf. [24]), génère des glyphes à partir d'éléments segmentaires combinés grâce à un programme en langage Python.

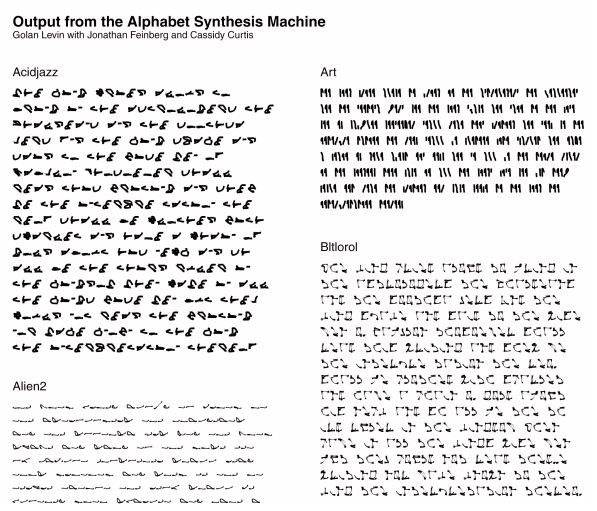


Fig. 2 : Des applets Java permettent aux visiteurs de `alphabet.tmema.org` de créer leurs propres polices artificielles en dessinant une forme que l'Alphabet Synthesis Machine va interpréter et transformer pour créer une police. Ce type d'expérimentation n'est pas sans évoquer le projet Letter Spirit, de Gary McGraw and John Rehling, deux étudiants du laboratoire de Douglas Hofstadter. En partant des formes de quelques caractères, Letter Spirit génère un alphabet complet.

les) alphabets unifon et apparentés. Nous nous y attardons aussi parce que son évocation nous semble susceptible — oserions-nous écrire, par l'absurde ? — d'apporter un autre type d'éclairage sur des problèmes qui se posent aux réformateurs de l'orthographe, et, partant, de la typographie, ou du moins, de l'orthotypographie[1].

Les alphabets de réforme

Apporter sa pierre à l'édifice d'une société plus harmonieuse, équitable et permettant l'égalité des chances est l'objectif de nombreuses initiatives philanthropiques, utopistes ou non. En ce qui concerne la société anglaise mais aussi les sociétés anglophones, et en raison de critères qui ne seront que survolés ici, la perception du progrès social passe par la promotion d'une meilleure égalité des chances liée à la maîtrise de l'expression orale et écrite … Une appréciation identique peut être formulée pour de très nombreuses autres sociétés, mais l'enjeu n'est pas perçu de la même manière.

Cet enjeu fut notamment illustré par la dotation que consentit George Bernard Shaw afin de contribuer à la création d'un nouvel alphabet, plus étendu que l'alphabet dit latin utilisé dans les pays anglophones … G.B. Shaw eut et a toujours des continuateurs et émules, l'importance de cet enjeu étant amplifiée par le phénomène actuel d'utilisation de la langue anglaise et de ses dérivés modernes (autres que le pidgin ou autres langues composites) par un nombre croissant d'allophones. Quelques remarques préliminaires s'imposent avant d'aborder le shavien et les alphabets apparentés.

Il n'existe pas de définition univoque de l'orthotypographie (ang. *orthotypography*, esp. et cat. *ortotipografía*) mais l'acception commune lie ce terme à l'activité des manuélistes (établissement des codes typographiques, marches particulières d'édition) et aux contenus de leurs ouvrages. Une définition étroite réserverait l'emploi du mot aux seules questions de transcription typographique impliquant un emploi particulier des caractères afin de se conformer aux règles et usages de composition des textes, indépendamment du respect de l'orthographe courant et des règles grammaticales usitées, sans toutefois bien sûr les négliger … Une définition beaucoup plus large, basée sur l'étymologie, impliquerait qu'elle traite des moyens et savoirs nécessaires pour composer correctement à l'aide de caractères mobiles. Mais en se référant soit au sens latin («modèle, exemple»), soit au grec («marque d'un coup» et par dérivation lettre ou caractère écrits), on peut englober toute écriture, manuscrite ou imprimée par tout moyen. De ce point de vue, la réforme la plus radicale envisageable pour une langue écrite est bien orthographique mais surtout orthotypographique. C'est le cas de systèmes innovants d'écriture (mais aussi de prononciation) de l'anglais dont nous en évoquons quelques-uns sous leurs seuls aspects, selon cette définition englobante maximale — partant, très approximative — , orthotypographiques.

Il est question ici d'alphabets le plus souvent mixtes, soit partiellement syllabiques. Les alphabets syllabiques étaient ou sont employés par exemple pour le phénicien, marqué par l'absence de voyelles, et d'autres langues et écritures postérieures. Les caractères coréens hanguls, les

1. Et aussi, comme nous y incite Ladislas Mandel dans [1] (si ce n'est aussi dans [2]), parce que nous considérons que l'évolution typographique est le reflet *des rêves*, *angoisses*, *et réalités des hommes et des sociétés* et que ces alphabets nous semblent significatifs de cette problématique.

japonais kanji hiragana et katakana, les trente-six signes de l'alphabet dit vieux perse du sumérien, ceux de l'inuttitut des Inuits (cf. leur plus récente version, l'aipainunavik, à voir sur les pages [3] et suivantes), la cinquantaine de signes du bengali, les trente-six correspondant aux syllabes déterminées par un missionnaire ayant analysé les parlers des Indiens de la baie de l'Hudson, le sequoya des Cherokees (créé par Sequoya vers 1822), témoignent de la diffusion chronologique et spatiale des systèmes syllabiques. Ils sont modernes ou obsolètes, soit périmés. En Crète, c. 1450 av. J.-C., le linéaire B apparaît, et il ne reste usité que jusqu'à 1200 av. J.-C. environ. On peut, d'un point de vue social, pour nos sociétés occidentales, tenter de caractériser la création de nouveaux alphabets phonématiques ou partiellement syllabiques en évoquant deux exemples. Leur création dérive de nécessités, qu'on ne peut que supposer altruistes, souvent liées à l'évangélisation chrétienne, toujours considérée émancipatrice par ses prosélytes. Ansi, le révérend Peck adapte pour la langue inuit un alphabet syllabique, dérivé de celui établi par le méthodiste James Evans pour les Cris, et traduit les Évangiles sur la base d'un texte en inuktituk conçu par les moraviens du Labrador. Leur création correspond aussi, tout comme celle de nombreuses langues synthétiques, esperanto et autres, à une volonté de retrouver une langue mythique des origines, d'avant la «babélisation» biblique, en vue d'une réconciliation générale, entre les humains, voire entre eux et une immanence, le «verbe» et le «Verbe» tendant à se concilier ; tout comme, pour les marxites, la transition socialiste est supposée aboutir à l'instauration du communisme qui réaliserait l'extinction du paupérisme et le progrès des peuples. Il n'est pas sûr que l'adoption et la généralisation de ces écritures constituent une régression. Mais la tentation peut parfois sembler sous-jacente. Citons ce passage de Chantal Chawaf : «*Il nous reste maintenant à travailler à apprendre mot à mot le passé de notre corps.* [...] *nous devons reconstruire* [...] *le syllabaire charnel*, *l'alphabet syllabique organique* [pour] *reproduire en grandeur naturelle la vie originale*, *la phrase infinie*, [...]» (repris de [4]). Il n'est pas question ici pour l'écrivaine d'évoquer un véritable alphabet, tout comme il n'est pas forcément dans les intentions des inventeurs ou initiateurs (G.B. Shaw, en particulier), de contribuer à recréer un Eden, d'aller vers une terre promise, par le biais d'une nouvelle écriture. Mais il serait tentant d'envisager ces alphabets aussi sous cet angle ; ce qui dépasse de très loin l'actuel propos ...

Achevons ces remarques préliminaires en évoquant rapidement Shaw. G.B. Shaw (1856–1950), "*Irish dramatist*, *essayist*, *critic and pamphleteer*", comme le définit le *Chambers Biographical Dictionary*, était le fils d'une professeure de chant, et sa vie publique débuta par un article sur les conséquences humaines de la propagande évangélique des pasteurs américains Dwight L. Moody

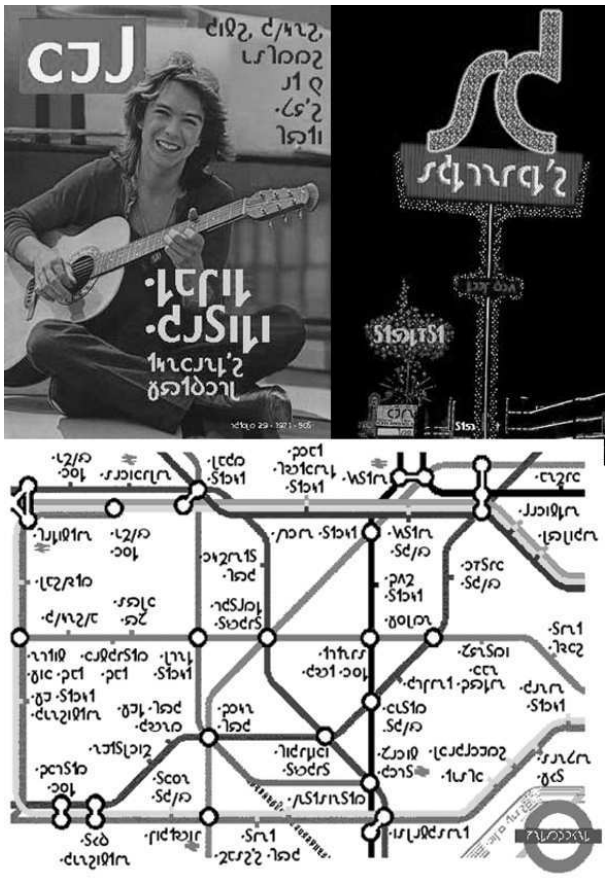


Fig. 3 : Simon Barne (cf. [27]) a utilisé les polices shaviennes disponibles (Androcles, Ghoti et Lionspaw) pour recréer des versions d'inscriptions familières : couverture du magazine *Life*, enseigne d'un restaurant McDonald, plan du métro de Londres.

et Ira D. Shankey en Grande Bretagne. Membre de la Fabian Society, il est marqué par la religion mais se proclame amoral et «héritique», proche des libres-penseurs, mais absolument pas matérialiste : il pressent une déité qu'il assimile parfois à l'Esprit Saint ... On peut le considérer féministe (cf. *Saint Joan* et d'autres pièces ou romans, essais, articles ...), et dans la lignée des réformateurs européens considérant que l'éducation concourt à l'émancipation sociale et intellectuelle (cf. *Pygmalion* et autres textes). Il allait devenir, à titre posthume, l'un des plus éminents réformateurs «orthotypographiques» et, surtout, des plus radicaux ... De son vivant, le style, la morale, l'influence de G.B. Shaw et de bien d'autres proches de lui par la pensée ou l'expression étaient qualifiées par l'épithète *shavian*. On peut donc avancer qu'il y a aujourd'hui plusieurs alphabets «shaviens».

Les alphabets shaviens G.B. Shaw a eu des prédécesseurs dans le domaine de la réforme orthotypographique et

il est assuré qu'il eut à connaître certaines de leurs propositions. Ainsi, celles de Mont Follick, professeur d'espagnol devenu membre du Parlement. Il préconisa une sorte de Spanglish (ce qui a peu à voir avec un sabir hispano-anglais sur le mode du franglais puisqu'il s'agit d'un dérivé de l'anglais supposé mieux adapté aux hispanophones et que le Splanglish est le nom d'un alphabet concurrent) et une réforme orthographique (par opposition à «orthotypographique») qui n'introduisait ou ne retranchait aucun des caractères de l'alphabet courant de l'anglais (courant, puisqu'une police anglaise étendue de base comprend aussi quelques caractères accentués, notamment pour composer des mots d'origine française, ce qui n'est guère le cas de nombreuses polices américaines de l'époque). Mont Follick, représentant travailliste de Loughborough fut conforté par le représentant conservateur de Bath, James Pitman, afin de, peu avant la mort de G.B. Shaw, proposer l'adoption d'une réforme de l'orthographe. Leur proposition de loi fut rejetée mais une polémique précéda et suivit leur initiative. Les deux alliés avaient cependant des vues divergentes. Pour résumer, l'un, Follick, se satisfaisait d'une adaptation de l'alphabet phonétique (IPA), l'autre songeait à revisiter l'alphabet phonotypique (fonotypic) inventé par son grand-père, Isaac Pitman. Sir Isaac (1813–97) avait inventé un système sténographique, le Stenographic Sound Hand, divulgué en 1837, puis fondé le *Phonetic Journal* en 1845. Sur le sujet, on peut se référer à [5][2]. Follick avait précédemment publié, en 1934, *The Influence of English* [7], et Shaw s'y était sans doute intéressé[3]. D'autant que le pacifiste Shaw devait sans doute en avoir apprécié les shaviennes visées : l'ambition désirable d'abolir la guerre ! (“*nothing less ambitious and desirable than the abolition of war*”). Il semble qu'il ait lu, comme beaucoup, le point de vue du *Times*, qui avait qualifié d'abominable la façon qu'avait Follick d'orthographier une chaise en anglais ‘ei tsheir’. C'est sans doute pourquoi, penchant pour l'approche de Pitman, et imaginant que toute transcription ayant recours à l'alphabet usuel serait taxée de ridicule et fantaisiste, Shaw en vint à envisager une solution beaucoup plus radicale : se départir des caractères latins (dits aussi romains, par rapport aux italiques) de l'anglais (un alphabet romain étendu car incluant les lettres «u», «j», «w» ; on emploie ici «latin» au sens de ‘latin de base’, par rapport au latin étendu aux caractères des écritures d'Europe centrale).

Shaw a clairement indiqué dans la préface de son Pygmalion qu'il s'était intéressé de très près aux travaux d'Henry Sweet (1845–1912), philologue et surtout auteur de *A History of English Sounds* (1874) et de *A History of Language* (1900). Sweet avait lui aussi conçu un

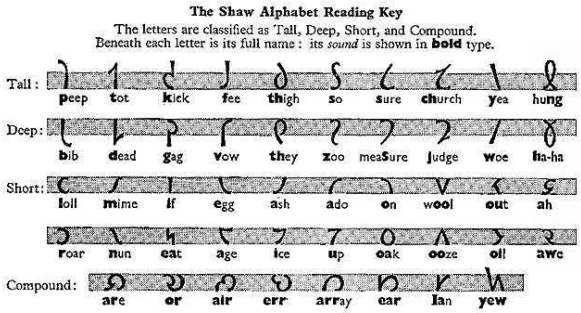


Fig. 4 : Les lettres shaviennes sont classées selon leur anatomie (*tall*, ‘à ascendantes’ ; *deep*, ‘à descendantes’ ; *short*, ‘sans hampes ou hastes’ ; et *compound*, ou ‘ligaturées’).

alphabet, le Romaic. Qu'en dit Shaw dans cette préface ? Sans doute un semi-mensonge :

Pygmalion Higgins is not a portrait of Sweet (…) still, (…), there are touches of Sweet in the play (…) Of the later generations of phoneticians I know little (…).

Cette faible connaissance énoncée est sans doute, pour le moins, une litote. Il s'était très probablement intéressé à quelques travaux des successeurs de Sweet. Toujours est-il qu'il était fort capable de piocher chez les uns ou les autres pour produire la transcription d'un paragraphe dont voici la première phrase :

Chang at leisure was superior to Lynch in his rouge, munching a lozenge at the burial in Merrion Square of Hyperion the Alien who valued his billiards so highly. Quick ! Quick !

Ce que Shaw transcrit ainsi :

/cAN at lIZD waz sMpCID t /linc in hiz rMZ, munciN a lyzanJ at H bXWl in /mXWn /skwX v /hFpCWn H ElWn hM vAlVd hiz biljDdz sO hFlI. kwik ! kwik !

On pourra lire la suite sur la page du Dr. Robert S. Richmond (M.D., F.C.A.P.) [9].

Décédé sans héritier, Shaw avait testé en prévoyant un fonds destiné à favoriser la création d'un tout nouvel alphabet comptant au moins une quarantaine de lettres. Les exécuteurs testamentaires chargés de ce fonds lancèrent un concours, dont le lauréat fut le typographe Kingsley Read, lequel avait eu auparavant des échanges épistoliers avec Shaw. Pitman supervisa le projet, et ainsi que les dernières volontés de Shaw le précisaient, sa pièce *Androcles and the Lion* fut transposée en utilisant la police créée, et diffusée largement en cette version shavienne dont l'éditeur Penguin (groupe Viking depuis) publia en 1962 une version duale (pour éviter «bilingue») sous le titre *The Shaw Alphabet Edition of Androcles and the Lion*,

2. [6] donne la version postérieure du débat par le protagoniste survivant.

3. À ce sujet, voir aussi [8, p. 98 & 137–39].

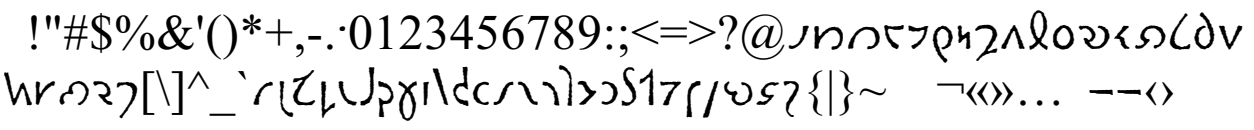


Fig. 5 : La police Androcles.

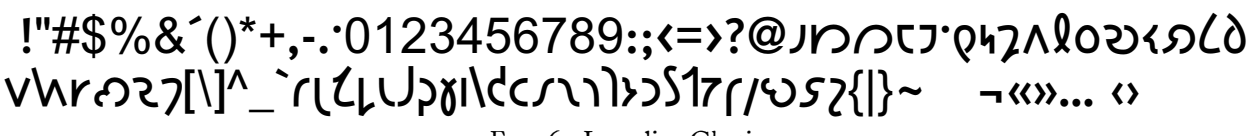


Fig. 6 : La police Ghoti.

dont la page de titre indique : *Androcles and the Lion, an Old Fable Renovated; by Bernard Shaw : with a Parallel Text in Shaw's Alphabet to be Read in Conjunction, Showing its Economies in Writing and Reading*.

Si Kingsley Read créa le dessin de la police, la fonderie Stephen Austins & Sons, de Hertford, en produisit trois graisses en corps 12 : normale (ou romaine), italique, grasse. Cette police n'a d'autre nom que celui de Shaw Alphabet, mais les polices numériques dérivées sont regroupées sous l'appellation de *shavian fonts*. La police est parfois aussi désignée sous l'appellation de Proposed British Alphabet. Sa vocation internationaliste semble ainsi affirmée (en 1950, l'anglais international est celui de l'empire britannique, et non déjà un dérivé d'un anglais surnommé «transatlantique» qui précéda un anglais plus actuel qu'on qualifiera — faute de mieux — de transnational). Nous ne nous intéresserons pas à la prononciation préconisée en termes vagues par Shaw, celle du nord de l'Angleterre, sinon pour remarquer ce qui suit …

L'Irlandais Shaw ne pouvait ignorer que, au cours de sa longue existence, la prononciation à l'irlandaise était particulièrement prisée dans certains cercles distingués (y compris dans les villes universitaires de longue tradition, sans doute beaucoup moins à Buckingham Palace, mais ce n'est qu'une supposition) voulant sans doute se différencier de la prononciation recommandée par l'exemple du corps enseignant et des multiples autres, socialement ou régionalement connotées. Le parler du nord de l'Angleterre pouvait être assimilé à un parler populaire à Londres, Bath, et dans le sud-est. Pour autant, ses propres termes sont les suivants : [une prononciation évoquant] "*that recorded of His Majesty our late King George V and sometimes described as Northern English*". George V avait inauguré, en 1931, des adresses radiodiffusées le jour de Noël. Celles d'Edouard VIII et de George VI nous étant tout aussi inconnues, on ne peut que supposer que celle du premier des Windsor sonnait plus populaire que celles de ses descendants. L'important était que, pour Shaw, l'anglais transcrit en shavien se «prononce comme il s'écrit», comme il est populairement dit.

D'un point de vue pratique, les shaviennes le sont bien peu … Shaw n'avait pas défini de machine à écrire fabriquée pour le saisir. En fait, Shaw pratiquait sa propre sténographie, très proche de celle de Pitman et non de celle de Sweet, pour rédiger ses divers écrits. Il l'indique dans sa préface à [10][4]. Il désirait qu'à chaque son corresponde une lettre et faute de pouvoir distinguer toutes les voyelles lui paraissant être perçues, il souhaitait autant de voyelles que possible. Dix-huit nouvelles voyelles (autant de lettres, caractères et glyphes de base, donc …) lui semblaient un minimum indispensable. Au total, 42 nouveaux caractères, aux glyphes typographiques (et dactylographiques) aussi différenciés que ceux du «vieil alphabet», devaient à ses yeux remplacer les 26 caractères de base. Il voulait se départir de la capitalisation mais prévisualisait de multiples marqueurs de «diacrités». S'il est ardu de s'imaginer réellement ce qu'il envisageait, on peut constater ce qu'il advint de ses intuitions … Il faut cependant croire que Kingsley Read avait prévu des emplacements de touches pour saisir son alphabet shavien (cf. "*The problem was that the first person to create a digital font did not spend much time selecting the keyboard locations for the new phonograms. Many of the key assignments are arbitrary*" [11]). En dépit des efforts des promoteurs d'une proposition d'inclusion de l'écriture shavienne dans la norme Unicode, en particulier ceux du typographe Michael Everson [12] et de John H. Jenkins [13], une telle adoption ne faciliterait guère la tâche des adeptes des polices shaviennes. La seule véritable facilité est que le point dénominatif shavien (*namer dot*), utilisé en tant que marqueur des noms propres (patronymes, toponymes), est commodément remplacé par le caractère de ponctuation générale point médian (*middle dot*). La proposition suggère de faire cohabiter l'écriture shavienne avec l'alphabet deseret, autre avatar de la phototypie de Pitman, cette fois promulgué par l'église des Saints du dernier jour (les Mormons), afin de faciliter l'écriture — et la prononciation — de l'anglais. Cohabitation

4. J.-J. Rousseau, et son livre *Essai sur l'origine des langues : ou il est traité de la mélodie et de l'imitation musicale* (cf. réédition de 1993 par Flammarion), sont évoqués par Richard A. Wilson, et il est permis d'imaginer que la réflexion de Shaw se soit nourrie aussi de cet essai.

Fig. 7 : Le graphiste Alessio Leonardi n'a pas tenté de créer un nouveau langage du genre volapük (créé par le père Johann Martin Schleyer) ou esperanto (du Dr. Ludwik-Lejzer Zamenhof), ou ido (du Dr. Louis de Beaufront, des Prs Louis Couturat, Richard Lorenz *et al.*). Partant du constat des évolutions des glyphes depuis leurs origines araméennes ou phéniciennes, il a imaginé des étapes de transformations de pictogrammes, dont ceux d'un arbre et d'une banane, pour créer son Alberobanana. Il a ensuite interprété les résultats pour créer des polices à la Bodoni, Rockwell, Frutiger et Omnia …

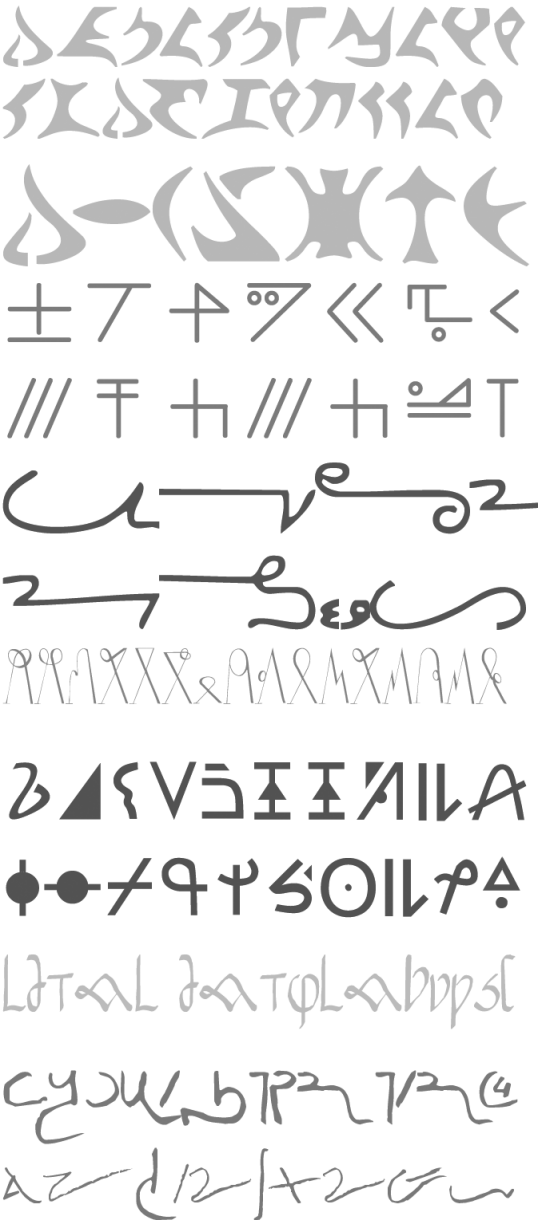


Fig. 8 : Diverses polices destinées à transcrire les langues de mondes virtuels

purement technique mais quelque peu savoureuse. Le deseret a été inclus dans la norme en perdant deux lettres (chutant de 40 à 38) et l'avenir dira si quantitativement, le shavien conservera cette prédominance numérique. Il n'est peut être exclu que d'autres alphabets artificiels ou de synthèse, possibles futurs candidats à la candidature, finissent par lui ravir l'avantage du nombre.

La praticabilité n'est guère assurée. Quant au principe d'économie de caractères à utiliser pour composer, l'un des buts recherchés par Shaw, il ne semble pas qu'il soit si flagrant. Les liens vers les fichiers images de la page [14] semblent indiquer que, même pour la composition ou le lettrage manuel des phylactères de bandes dessinées, le gain soit minime, et plutôt de l'ordre de 15 à 25%. Certes, ce n'est pas négligeable. Sa lisibilité intrinsèque (expression évoquant, faute de mieux, tel un néologisme qui pourrait être «lexabilité», l'anglais *legibility*) semble convenable. La condenser davantage[5] parait aisément envisageable, sauf pour, justement, les caractères

5. Condenser, pour une police, n'est pas étroitiser. Dans le premier cas, le concepteur réduit la chasse des caractères en les redessinant, dans le second, par interpolation, un logiciel opère une réduction de cette chasse. Une bonne condensation ne nuit guère à

«composés» ou agglomérés (*compound*). Il est bien sûr difficile de se prononcer quant à la lisibilité extrinsèque (ou *readability*, que le néologisme «lexibilité» pourrait traduire). Ce, partant d'un principe voulant que "*a legible font doesn't necessarily make a readable text*". Tant bien même, juger de la lisibilité globale (readability selon les deux acceptions, soit au niveau global de la mise en œuvre de la mise en page et du style employé) exigerait une très longue pratique. Écrire plus vite, plus long, tout en économisant de l'espace (papier, écran, autre support) n'est nullement assuré par de simples apports techniques, quelle que soit la police ou les subterfuges (réglage des approches ou réduction des espaces intermot) employés.

Au passage, on s'interrogera. Quel espace subsiste-t-il pour l'orthotypographie avec un tel système ? L'orthotypographe devra-t-il s'incliner, disparaître, tendre vers l'orthotypographisme ? Soit une approche plus graphique et, partant, bien moins large et globalisante[6] ? Il serait absurde de débattre de la ligaturisation (cf. ex. supra) des caractères shaviens. La corporation des orthotypographes est aussi insignifiante en nombre qu'elle reste informelle, mais il est aussi loisible de penser qu'outre les réticences générales, les corporatismes auraient eu toutes les raisons de s'opposer à la généralisation d'emploi des polices shaviennes si le risque n'en avait pas été limité au point d'être négligeable.

Quant à l'adoption spontanée et quasi-immédiate d'un système d'écriture, on sait que le système sténographique (puis l'écriture rapide, système d'abréviation sans modification des caractères) s'est progressivement étendu pour régresser par la suite (dictée vocale, reconnaissance assistée de la parole, reconnaissance automatique de l'écriture et tablettes ou écrans numériques y ont contribué). C'est l'un des rares cas d'adoption progressive généralisée découlant de l'adhésion des pratiquants.

L'imposition d'un système nouveau suppose un pouvoir fort, coercitif. Ainsi de la réforme de Mustapha Kemal, dit Père des Turcs (Mustapha Kemal Pasha, dit Atatürk), en 1928 (3 000 mots empruntés au français, d'autres à d'autres langues, mais transcrits à l'allemande), puis d'autres (en Asie notamment), qu'elles ressortent d'une sorte d'intégrisme laïque ou de raisons plus pragmatiques. Ainsi du fameux cas d'école hitlérien (idéali-

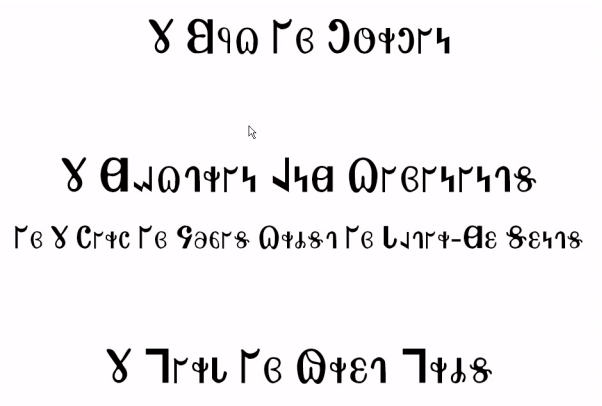


Fig. 9 : Exemple de Deseret, alphabet phonémique créé en 1850. Dans sa contribution à l'ouvrage *Unicode, Écriture du monde* (Jacques André *et al.*, Lavoisier, Paris, 2003), «Unicode, tentations et limites», Olivier Randier remarque à son égard que cet alphabet dont l'usage a été abandonné par les mormons est correctement géré par Unicode «*alors que le tifanagh, l'écriture des Berbères (plus de 30 millions de locuteurs ...), en est toujours totalement absent.*»

sation de l'écriture fraktur, revirement, démonisation — liée abusivement au génocide des juifs et à l'éradication de leur culture — pour des raisons de propagande [faciliter la lecture rapide des textes, notamment par les journalistes étrangers disposés à collaborer en véhiculant la vulgate nazie] et pratiques : signalétique dans les pays occupés qui déroute plus les autochtones que l'espionnage ennemi). L'emploi plus fréquent et géographiquement diffus de telle ou telle classe[7] de familles de caractères est lié à de multiples facteurs (quasi-hégémonie d'un système d'exploitation à partir des années 1990, forte disponibilité et diffusion antérieurement) dont les phénomènes de mode, le discours ambiant, ne sont pas les moindres. La première shavienne apparaît au moment où, comme le rappelait Gérard Blanchard[8], «*la formule réductrice* (de 1950) *du "style suisse international" à un seul caractère bâton (Akzidenz grotesk, Helvetica, Univers)* ...» tend à devenir omniprésente et se «mondialise», comme on le dit en ces années 2000. Les raisons pratiques, énoncées scientifiques (sur la base d'études de lisibilité sur divers supports et à diverses distances de lec-

la lisibilité intrinsèque, une étroitisation inconsidérée réduira tant celle-ci (*legibility*) que l'extrinsèque (*readability*).

6. Extrait d'un courriel à Jacques André : «Enfin, pour scinder longitudinalement un cheveu, je m'interroge sur la définition même de l'orthotypographie. L'orthotypographisme (apanage d'orthotypographistes) est-il envisageable ? En quoi se distinguerait-il d'une orthotypographie (labour d'orthotypographes, dont les orthotypographistes ne seraient qu'une partie glanante de l'ensemble) ? Les orthotypographes se prononçant sur la présence ou l'absence parasites ou conformes d'une espace ci et là, les orthotypographistes s'exprimant sur la valeur souhaitable, nulle ou proportionnelle à leur étalon, de la dite espace ?» (Jef T., 30 nov. 2002).

7. Par classe de familles on entendra ici de larges catégories répertoriées ou non dans les multiples classifications des familles (ensemble des polices de même nom ou de noms apparentés d'un même créateur). L'emploi pour désigner un ensemble quelconque de familles est donc ad hoc, et n'est rien de plus qu'une facilité langagière ...

8. Dans un texte qui pourrait avoir été rédigé au début du second semestre 1998 puisqu'il est tiré des *Lettres françaises*, Jean-François Porchez, dir., Paris, oct. 1998, ATypI-Adpf.

𐑞𐑲 𐑕𐑑𐑮𐑲𐑒𐑑 𐑒𐑪𐑙𐑑𐑮𐑨𐑒𐑑𐑦𐑪𐑯 𐑣𐑨𐑟 𐑤𐑧𐑕 𐑮𐑦𐑕𐑒 𐑞𐑨𐑯 𐑞𐑲 𐑒𐑮𐑲𐑟 𐑤𐑨𐑒 (Gothi c. 14, pas d'étroitisation).
𐑞𐑲 𐑕𐑑𐑮𐑲𐑒𐑑 𐑒𐑪𐑙𐑑𐑮𐑨𐑒𐑑𐑦𐑪𐑯 𐑣𐑨𐑟 𐑤𐑧𐑕 𐑮𐑦𐑕𐑒 𐑞𐑨𐑯 𐑞𐑲 𐑒𐑮𐑲𐑟 𐑤𐑨𐑒 (Gothi c. 14, étroitisation de 20%).
𐑞𐑲 𐑕𐑑𐑮𐑲𐑒𐑑 𐑒𐑪𐑙𐑑𐑮𐑨𐑒𐑑𐑦𐑪𐑯 𐑣𐑨𐑟 𐑤𐑧𐑕 𐑮𐑦𐑕𐑒 𐑞𐑨𐑯 𐑞𐑲 𐑒𐑮𐑲𐑟 𐑤𐑨𐑒 (Androcles c. 14).
𐑞𐑲 𐑕𐑑𐑮𐑲𐑒𐑑 𐑒𐑪𐑙𐑑𐑮𐑨𐑒𐑑𐑦𐑪𐑯 𐑣𐑨𐑟 𐑤𐑧𐑕 𐑮𐑦𐑕𐑒 𐑞𐑨𐑯 𐑞𐑲 𐑒𐑮𐑲𐑟 𐑤𐑨𐑒 (Androcles c. 14, étr. 20%).

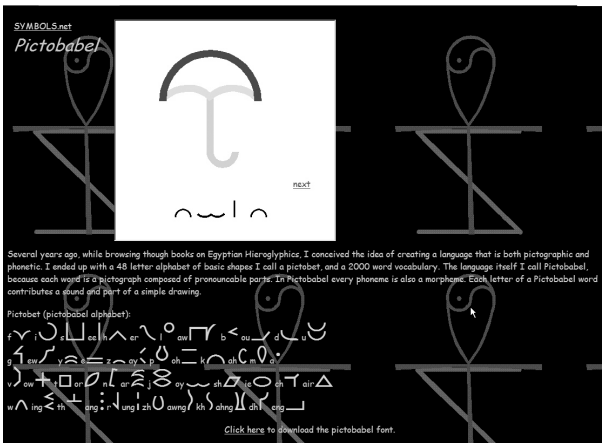


Fig. 10 : Le Pictobabel, tel que le montre la page d'accueil de [26], site créé, comme l'alphabet et sa police, par George F. Sutton. Cet alphabet, composé de 48 formes de base, inspiré des hiéroglyphes égyptiens, permet de composer environ 2 000 «mots».

ture) par la suite, n'expliquent bien sûr pas isolément cet engouement. L'Europe ressurgit de ses cendres, un nouvel ordre mondial s'ébauche, avec au moins deux prétendants se targuant d'incarner modernité, progrès, universalité. Était-ce un handicap ou un atout pour l'alphabet shavian ? Chacun pourra refaire l'histoire à sa manière, estimer que la période, propice aux expérimentations en tous domaines (dont l'éducatif) lui était favorable ou néfaste (pour d'ailleurs pratiquement les mêmes raisons, l'argument avancé par les tenants d'une hypothèse pouvant être repris par leurs contradicteurs, ainsi de ses origines dans le camp «capitaliste» à l'initiative d'un «socialiste»). Entre les phénomènes de modes et les conflits de pouvoir(s), la typographie est un enjeu souvent négligé mais qui n'est pas forcément négligeable, comme l'a mis en lumière Ladislas Mandel [1] (cf. la mise au service de la puissance impériale romaine de l'écriture lapidaire ou le rayonnement projeté du Romain du Roi, reflétant la grandeur du Roi-Soleil ou la recherche de la rentabilité par la suite …).

Énoncé plus rentable d'emploi que tout autre alphabet pour l'écriture manuelle ou la reprographie (puisqu'il économisait le papier, entre autres allégations ou arguments solides énoncés), l'alphabet shavian «avait ses chances» mais les arguments techniques ne sont pas les seuls pris en compte, qu'ils séduisent ou révulsent les novateurs ou les luddites d'une époque. Ont été surtout mis en avant, postérieurement au décès de Shaw, les avantages techniques de l'alphabet shavien. Ses promoteurs tentent principalement de convaincre de sa facilité d'apprentissage : "*Only a few hours will be needed to persuade you that the new alphabet has the potential advantages Shaw intended for it. At first you will read and write it in a plodding childlike way, as you once did Roman. Much more rapidly than a child's, your familiarity and ease will grow, until the use of Shaw's alphabet becomes as natural and automatic as your use of Roman — only faster*"[9]. (extrait d'une lettre de James Pitman, House of Commons, London, 1962). On ne cherchera pas à savoir de quelles sommes le fonds prévu pour appuyer cette évidence proclamée fut abondé (initialement, de 500£ seulement). Il est très probable qu'elles furent bien moins considérables que d'autres oeuvres, les droits d'exploitation de la comédie musicale dérivée de *Pygmalion* (*My Fair Lady*) ayant et, que l'on sache, à ce jour encore, alimenté les efforts rétribués de certains. En fait, les efforts de sociétés telles la Simplified Spelling Society, OpenRite, ou l'American Literacy Council, ont beaucoup plus contribué à focaliser l'attention sur les nouveaux alphabets. Le morse est quasiment abandonné, sauf par des amateurs, les sémaphores sont des objets muséaux, les marins communiquent de moins en moins en maniant des fanions, mais les shaviennes gardent d'actifs partisans. Elles sont concurrencées. Ainsi par John Malone, concurrent malheureux de Kinglsay Read, avec l'Unifon (actuellement en v. 2). Read simplifia, en 1960, son Proposed British Alphabet et présenta le QuickScript (sur les mêmes bases, mais en allégé, et cette fois en écriture cursive). Au nombre des concurrents, citions déjà l'englik, le spanglish déjà évoqué, et le truespel (dit système digraphique anglophonique, qui révèle les accentuations toniques). Nous avons aussi le *new follick* (nu folik), se fondant sur le *broad romic* d'Henry Sweet et Daniel Jones.

On verra sur [15] une liste d'alternatives supplémentaires (mais il s'agit davantage de simplifications orthographiques sans introduction de lettres supplémentaires, ou, du moins, très parcimonieusement).

Alphabet Unifon Si le shavien considère que 42 signes sont nécessaires pour transcrire l'anglais selon le principe

9. «Quelques heures d'essai suffiront à vous persuader (…), vous finerez par l'utiliser naturellement, tout comme l'alphabet latin, mais en gagnant en rapidité» [adaptation libre résumée].

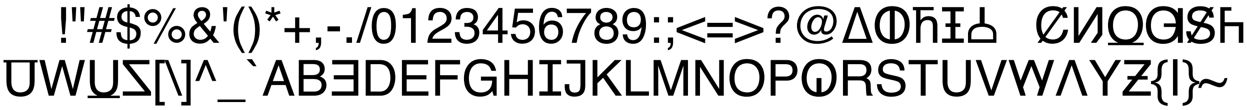


Fig. 11 : La police Unifon.

qu'à un son correspond un signe, l'unifon se satisfait de 40 signes seulement. Les graphes sont assez proches de ceux de l'alphabet latin, et le gain de place, variable selon les phrases, semble peu important. Ainsi, pour "*each letter is associated with a single sound*" (38 lettres), 32 lettres unifon suffisent. Mais certaines lettres correspondant à un son, dans l'unifon des origines, sont transcrits par une paire de caractères (ainsi un «x» est noté «KS»). Il s'agit de majuscules uniquement. Là où le Bahaus considérait que les majuscules étaient superflues (et les textes du Bahaus s'en dispensèrent à partir de 1925), les tenants de l'unifon ont pris l'option inverse. D'autres choix, basés sur le même principe, sont tout aussi valides à cette aune. Ainsi, l'ANJeL et l'UNiGRaF vont assigner soit une capitale, soit une bas de casse, à un son. Ainsi, avec l'ANJeL de Bruce Beech, *go out* devient «gX mt» («g bdc, X-maj., m bdc, t bdc»), tandis qu'avec l'UNiGRaF de Steve Bett, on aura «gO Mt» (le «X-maj.» devient «O-maj.», le «m bdc» devient «M-maj.»). Partant de l'observation que les bas de casse sont plus distinctives, discernables (*legible*) que les majuscules, une version bas de casse de l'unifon était, fin 2002, envisagée. Ce serait l'unicase. Une première épuration de l'unifon est d'ailleurs intervenue en 2001 allant déjà en ce sens.

L'évidente différence, du point de vue qui nous intéresse ici, soit la création typographique, entre cet unicase et l'unifon créé au cours des années 1950 par l'Américain Malone, est que les glyphes utilisés seraient similaires à ceux existant actuellement dans les divers alphabets latin ou grec. En cela, il s'apparenterait davantage aux diverses utilisations de l'alphabet latin courant permettant de simplifier l'orthographe sans recours à des caractères spécifiques aux glyphes inventés de toute pièce. Ainsi, on peut noter la graphie «ou» du mot *you* en doublant simplement le caractère «u», et c'est ainsi que procèdent les divers réformateurs de l'orthographe de l'anglais : soit en éliminant des lettres, soit par doublement de certaines. De nombreux systèmes employant des procédés de notations alternatives existent et nous ne les aborderont pas ici, si ce n'est qu'incidemment. Ainsi, la notation «*winglish*» (pour *World English*) dite romik consiste à transcrire 25 voyelles par six symboles dont les glyphes sont identiques aux caractères latins.

L'unifon est typographiquement moins intéressant que le shavien puisque la plupart des glyphes courants sont conservés, l'inventivité portant sur la création de 17 caractères par une légère modification des glyphes

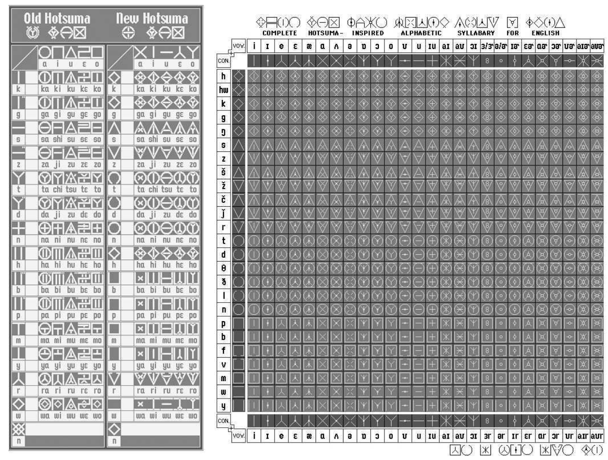


Fig. 12 : Le Hotsuma.

existants, l'emprunt de certains autres à l'alphabet grec, et l'invention de quelques-uns aux formes plus insolites. Ainsi la majuscule A subsiste, mais un «second» caractère, identique à la capitale grecque delta, transcrit un autre son, tandis qu'un troisième est obtenu par la suppression de la traverse de la capitale A. Moins d'une demi-douzaine de glyphes diffèrent réellement de ceux que nous employons couramment. Il s'agit en particulier d'un caractère évoquant le pictogramme d'un étrier (ou un y renversé barré d'un macron) dont l'usage est dévolu à rendre les sons des mots *ice* et *eye* ou *dice*. Des tableaux sont visualisables sur le site [16].

Pour le reste, les exemples ci-dessous montrent que l'utilisation de l'unifon est relativement simple lorsque les nouveaux caractères sont assimilés et qu'un gestionnaire de clavier est installé … Toutefois, les combinaisons des touches courantes et de celle(s) de(s) majuscule(s) suffit à produire l'ensemble de l'alphabet. Comme pour d'autres notations, la notation phonétique IPA, les phonétisantes (notations des manuels Berlitz et autres d'apprentissage de langues, des dictionnaires dont le Merriam-Webster Dictionary, ou celles des diverses sociétés visant à la simplification de l'orthographe), l'apprentissage de lecture semble aisé, l'écriture étant plus ardue à dominer.

AZERT YUIOP QSDFG HJKLM WΛƎVB N ⌂ ⌂

ħƎ KWIK BRQN FOKS JUMPD QVƎ ħƎ LΛZ⌂ DΛG

Et parmi d'autres … Du vieil au nouvel hotsuma … Le hotsuma, dit aussi écriture Iyo, supposée inspirée de sym-

boles tantriques, a été revisité pour transcrire les sons de l'anglais. Il s'agit du New Hotsuma, alphabet basé sur des formes simples (point, cercle, triangle, carré, losange …). Il s'apparente à un syllabaire. La combinaison de deux formes (par ex., par l'inclusion d'un trait dans un losange), et l'orientation d'un des deux éléments (trait horizontal ou vertical), forme une syllabe par combinaison des caractères des consonnes (de formes englobantes) et de ceux des voyelles (plus simples, pouvant être inscrits dans ceux des consonnes).

Autres alphabets

On ne peut tenter ici d'évoquer tous les autres alphabets s'écartant des normes traditionnelles. Certains sont des créations artistiques et le fait d'artistes ou d'étudiants en graphisme. Évoquons simplement au passage ceux de Michel et Roselyne Besnard (peintures, tapisseries, sculptures, etc., inspirées par des polices le plus souvent expérimentales). Voici ce que Jean-Claude Eymeri dit de leurs signados : «*La déconstruction alphabétique en formes élémentaires symboliques nous ramène au primitif, aux formes fondamentales qui ne pourraient être que l'habitude d'un couple, forme de demi-lune, réceptacle féminin, disque naissance, forme masculine complémentaire ; éléments symboliques qui nous font souvenir du profane, du sacré.*» Il convient aussi de remarquer, dans la classe des tentatives artistiques, l'étrange alfabetempo de Catherine Zask, graphiste et plasticienne. Cet alfabetempo qui peut se transposer aussi en trois dimensions, car ses caractères épousent les faces d'un cube, est ainsi décrit par son auteure : «*il est né de la décomposition des temps du tracé des lettres. Arbitrairement, chaque temps de tracé s'installe dans un carré.*». Il est visible en particulier sur le site Pixelcréation [17]. Bien d'autres ont été créés de la sorte en répondant à des visées esthétisantes ou symboliques.

L'anakatabase Mais nous ferons une exception pour l'étrange Anakatabase («escalier», en language courant) du maître d'art français François da Ros. Créateur des éditions homonymes, F. da Ros, apprenti composeur chez Genin Frères en 1962, puis ouvrier du Livre chez Fequet-Baudrier, puis linotypiste, a repris la composition au plomb en 1983. Extrêmement familiarisé avec les casses, il a développé une étonnante mémoire visuelle qui lui a permis de concevoir un très particulier alphabet. Celui-ci est visuel et, osera-t-on, gestuel. Sa particularité est de transcrire chaque mot, et il trace autant de caractères qu'il emploie de mots. Cet alphabet a connu deux versions. La première consistait à relier des points par des droites et se traçait de gauche à droite et de haut en bas où inversement, selon les déplacements de la main saisissant chaque caractère dans la casse. La seconde version reprend le même principe mais aboutit à des signes évoquant des idéogrammes. De part et d'autre d'une ligne verticale (cloison centrale de la casse) sont répartis les emplacements et des signes, évoquant des «c» allongés et retournés (rotation anti-horaire de 90°), marquent les «prises» (prélèvements successifs dans la casse) et le parcours de la main.

Qu'en dit-il ? Ceci : «*Je peux l'écrire couramment mais non le lire. J'arrive à quelque chose de trop difficile à décrypter … La casse est comme une portée musicale mais il est impossible de converser en anakatabasien, ce serait trop long, fastidieux.*» L'avantage de cette écriture tient notamment à la lenteur de tracé qu'elle exige. Pour paraphraser Marcel Proust, qui écrivait en substance : Pardonnez-moi d'avoir écrit trop long, le temps m'a manqué pour faire court …, rédiger lentement oblige sans doute, pour ceux animés d'une telle disposition d'esprit, à écrire «mieux», ou, du moins, plus concis. Cet alphabet est aussi une construction qui pourrait avoir un intérêt pédagogique pour réfléchir sur l'écriture idéographique en général. Nous l'évoquons surtout pour élargir le propos aux multiples «possibles» devenirs des alphabets.

Le Christian-Marc Schmidt Faute de mieux, nous nommons ce système du nom de son auteur. Christian-Marc Schmidt était étudiant au Communication Design Department de la Parsons School of Design (Greenwich Village, New York) lorsqu'il conçut, en 2002, un système graphique expérimental faisant correspondre un caractère à un mot. De A à Z, 26 lettres sont disposées en circonférence d'un cercle. L'initiale «muette» étant le centre de ce cercle, chaque lettre est reliée par des droites. Ce système présente divers désavantages relatifs à la lisibilité. Les caractères accentués sont ignorés. Les segments de droite peuvent former un angle très fermé pour les mots comportant des lettres adjacentes (par ex., pour le possessif «mon», le segment «m-o» se distingue peu du segment «o-n»). Toutefois, visuellement, cette proximité confine à la production d'un segment perçu unique, mais plus épais. Si l'anakatabasien n'a pas d'instance numérique, ce système est parfaitement et aisément gérable par un logiciel relativement simple.

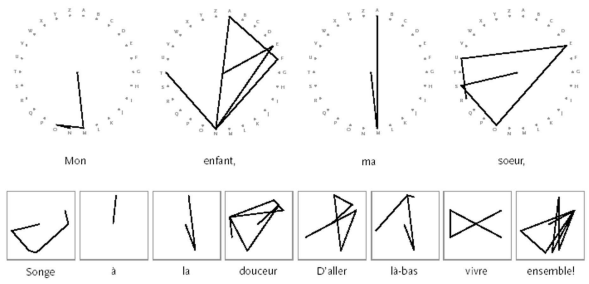


Fig. 13 : Le système graphique de Christian-Marc Schmidt.

Les inconvénients apparents pourraient trouver des solutions en ayant recours à des cercles concentriques pour les caractères accentués, les majuscules, et à des variations d'épaisseur, de tonalité (passage du gris au noir ou à une autre couleur). Pour les mots les plus longs des diverses langues, le déchiffrage semble malaisé. Ce système, affiné, perfectionné, serait-il plus recevable qu'un autre ? Tenter de répondre tient de la gageure. Limitons-nous à supposer que, comme pour le shavien ou tout autre système d'écriture, la meilleure grille d'analyse et les critères les mieux fondés ne suffiront pas à convaincre.

L'alberobanana Il en est de même de l'alphabet «arbolobananier», l'alberobanana, visible sur le site [18]. L'alberobanana, d'Alessio Leonardi, doit son nom à l'arbre et au bananier. Explication … Alphabet évoque le bœuf et la porte (Alpha, Beth). Par rotation, l'alpha devint «a» et on connaît — du moins, on a déduit — la suite. Leonardi a imaginé que les créateurs successifs d'une écriture fictive faisaient découler l'aspect de leurs caractères «a» et «b» de pictogrammes de l'«arbre» et de la «banane». Partant de 26 objets ou éléments («clou», «oreille», «parapluie», «poisson», «soleil» …, pas de raton laveur mais c'était imaginable), et par stylisations successives, il crée un alphabet contemporain plausible. Donc 26 caractères donnant la matière à des glyphes à la manière de … Bodoni, Frutiger, ou évoquant une onciale ou la Rockwell. Pour être aussi étranges que possible, ces formes nous semblent cependant plus proches de «nos» alphabets occidentaux (du grec ancien, de l'islandais, de ceux que l'on voudra) que celles du shavien. Toute considération esthétique mise à part, cette tentative de réécriture purement démonstrative de l'histoire de nos écritures n'est pas sans portée. Ce que la succession passée des usages a pu produire peut, selon des processus insoupçonnables, à l'avenir, aboutir à autre chose que ce que nous connaissons : comment l'exclure ? Que seront nos écritures des deux millénaires à venir ? Il serait vain d'en préjuger …

Les autres langages artificiels Certains étudiants des écoles de graphisme en fin d'études sont parfois enclins à créer une police de symboles, si ce n'est un alphabet imaginaire. Il s'agit de travaux purement formels. Toute autre est la démarche de prolongateurs ou de créateurs *ex nihilo* d'univers virtuels qui inventent un véritable langage transcrit par des graphies pouvant être translittérées. Ces initiatives sont surtout le fait d'amateurs de films ou de séries télévisées de science-fiction, comme en témoignent les très nombreuses polices du langage klingon de la série *Star Trek*. Mais on trouvera, notamment sur le site de Luc Devroye [19], véritable portail pour tout ce qui se rapporte à la typographie, de nombreuses polices inspirées d'autres films ou de jeux vidéo. Ainsi celles de la serie *Babylon* 5. Elles sont classées dans les catégories «runes» (en particulier des polices dérivées des langues de la trilogie de J.R.R. Tolkien) ou «science-fiction» de son site. Parmi les créateurs de mondes virtuels, Mark Rosenfelder est peut-être celui qui s'est consacré le plus opiniâtrement à créer des langages. Son verdurien et d'autres langages imaginaires sont soigneusement codifiés. L'alphabet verdurien comprend des initiales, des médiales, des finales, et se compose de versions manuscrites (police cursive) et pour l'impression. Ces types de créations sont assez similaires à celles de linguistes, tel Boudewijn Rempt, qui ont élaboré des langues artificielles. Rempt a d'ailleurs aussi créé un monde virtuel, l'univers d'Andal. On trouvera les polices afférentes sur son site [20].

Le pictobabel de George F. Sutton Sur `Symbols.net`, George F. Sutton propose son alphabet pictographique, le pictobet, qu'il utilise pour transcrire le pictobabel, langage qui comprend, selon son auteur, 2 000 mots. Le pictobet, fort de 48 signes, transcrit autant de sons de l'anglais. Sutton perpétue la mémoire du chimiste autrichien Charles Kasiel Bliss (1897–1985) qui avait inventé, en 1942, un langage pictographique basé sur une infinité de symboles, les Blissymbols, et rédigé un traité de «sémantographie» pour leur emploi. Une association internationale, Blissymbolics Communication International (BCI), basée au Canada, se targue d'être représentée dans 33 pays où les quelque 2 000 pictogrammes de ce langage seraient utilisés (au moins à l'occasion) [21]. Selon la BCI, une centaine de symboles de base seulement permettraient à des personnes, ne pouvant se comprendre autrement, de communiquer efficacement. Des marqueurs (du pluriel, du possessif, de conjugaison …) permettent des combinaisons syntaxiques.

Conclusion

Il serait présomptueux d'imaginer quels genres de polices seront utilisés, à la fin de ce siècle, par les nouveaux-nés de l'été 2003. Nos caractères sont peut-être fixés, définitivement. Ou non. Mais la technique permettant désormais d'associer un son à une forme, peut-être déjà à un geste, un muet mouvement des lèvres, si ce n'est du regard, nous conclurons … en ne concluant pas, laissant chacun songer que ce qui parait aujourd'hui immuable ne le semblera pas demain. Dans ces conditions, le shavien, l'unifon ou d'autres, pourront bénéficier (ou non, comment le prévoir ?) d'un singulier retour vers le futur ou inspireront-t-ils d'autres approches, aujourd'hui insoupçonnées : pourquoi d'emblée l'exclure ?

Comparaison n'est pas raison. Toutefois, nous sommes tentés de faire état, au final, des interrogations que l'introduction de nouveaux outils ayant recours aux techniques de la reconnaissance de la parole et de l'écriture suscite aujourd'hui chez les praticiens et spécialistes

de l'apprentissage de la lecture et de l'écriture. Des polices, des méthodes, naguère considérées novatrices, apparaissent aujourd'hui désuètes. L'accent est mis à présent sur la rapidité de la prise de notes manuscrites et leur facilité de relecture par le scripteur, tout en visant l'acquisition d'une écriture scolaire jugée convenable par les enseignants, le papier restant le seul support envisagé pour l'instant. Certains qui hier présageaient que le clavier allait vite supplanter le stylet commencent à en douter. De même, alors que l'utilisation du clavier était envisagée rapidement dominante, toutes les tentatives de généraliser l'emploi de claviers estimés plus ergonomiques, plus faciles à utiliser pour accélérer fortement la saisie, n'ont pas abouti. En fait, en considérant valides les supputations les mieux argumentées sur l'évolution passée des écritures, force est de conclure que les évolutions à venir dépendront, si ce n'est de l'accidentel (catastrophes, séismes, ravages de conflits ...), d'un subtil amalgame entre l'adhésion aux (ou le rejet des) possibilités ouvertes par l'évolution des sciences et des techniques et à celle, consentie ou forcée, au volontarisme (ou à l'attentisme) politique ... Il reste qu'on lit plus vite qu'on n'ouit (entendre, au sens de comprendre, étant tout autre chose). Une société productiviste, toujours plus soucieuse de performances, de rapidité, est toujours tentée par des gains de célérité de communication, et n'est freinée, pour l'adoption des systèmes et dispositifs les réalisant, que par l'évaluation du degré d'acceptation (garant de la rentabilité des innovations lancées sur le marché). Le devenir des types d'alphabets évoqués, qui peuvent sembler dépassés ou anecdotiques pour ceux décrits, bien aléatoire pour ceux à venir, sera fonction de choix de société(s).

Annexe

Les réformateurs anglophones de l'orthographe De nombreux réformateurs britanniques de l'orthographe de l'anglais ont tenté, soit comme leurs équivalents français du passé, de proposer une harmonisation de l'écriture, soit de faire adopter des mesures plus radicales de simplification. Pour le passé récent, on connaît l'apport de la Philological Society, fondée en 1830 (et dotée de statuts en 1842), pour la dictionnairique (projet de dictionnaire initié en 1857, publication en 1879 de ce qui allait devenir le *Oxford English Dictionary*). Mais cette société savante s'est aussi intéressée à la simplification de l'orthographe et aux divers apports extérieurs, notamment ceux de l'Américain Noah Webster (1758–1834). Ce dernier plaidait pour une simplification de l'américain et l'American Philological Association finit par considérer ses idées favorablement. En 1876, une International Convention for the Amendment of English Orthography se réunit à Philadelphie et elle devait accoucher d'une Spelling Reform Association. En 1883, les deux sociétés, l'américaine et la britannique, publièrent un manifeste commun. Lequel bénéficia de l'appui de personnalités tels Charles Darwin, Sirs John Lubbock et J.A.H. Murray, Lord Tennyson. Mais les Américains furent par la suite plus volontaristes, même si leurs amendements, passés dans l'usage courant, peuvent paraître cosmétiques au regard de réformes plus radicales. Les partisans d'une simplification modérée furent par la suite confortés par l'action, à partir de 1906, du Simplified Spelling Board, soutenu par Andrew Carnegie et des personnalités de l'époque. Le président Roosevelt, la même année, décida que leurs préconisations soient suivies par l'office gouvernemental des publications officielles américaines (Government Printing Office). Mais cette décision ne fut pas suivie d'effets autres que mineurs.

Il faudra attendre 1920 pour que l'idée d'introduire une réforme radicale germe dans les esprits de novateurs. Wilfrid Perrett publie *Peetickay : An Essay Toward the Abolition of Spelling* (Cambridge, Heffer & Sons). Dénonçant l'absurdité de se contenter de 26 lettres seulement pour transcrire une quarantaine de sons, Perrett proposera d'utiliser des ponctuations pour transcrire les voyelles mais ne suggérera pas un tout nouvel alphabet.

L'initiative de G.B. Shaw, et son échec, dérivent possiblement de cette radicalité : créer un tout nouvel alphabet. Cependant, elle a connu des prolongateurs qui ont tenté soit de conserver la plupart des caractères existants, et d'en créer quelques nouveaux, soit de simplifier l'orthographe. La consultation de la liste des nombreux liens de la page ad hoc du site `spellingsociety.org` montre que l'intérêt pour une réforme radicale reste vivace. Des sociétés telles RITE (*rite* transcrit *write*) ou le forum SaundSpel (pour *sound spell*) , celui de l'«euro-english», montrent que l'intérêt pour des réformes ne s'est pas éteint.

[22] de H.L. Mencken (1880–1956) est le plus complet sur les évolutions de l'anglais depuis les années 1910.

S'il fallait porter jugement sur l'avenir des alphabets de réforme ou artificiels, soit sur leurs chances d'adoption large et durable, il est bon de rappeler que le mouvement du Bahaus, en dépit de sa notoriété, du rayonnement de l'œuvre d'Herbert Bayer, n'a pu obtenir l'adhésion à cette simple mesure : abolir l'emploi des lettres majuscules. L'argument était que les majuscules et les bas de casse se prononçaient à l'identique, que des machines à écrire ne comportant pas de majuscules seraient plus faciles à produire (et à moindre coût) et à utiliser (tout en permettant une meilleure vélocité de frappe), et que des gains en temps de composition et de matière première (de papier) seraient réalisés. Le Bauhaus cessa d'employer les majuscules dans ses publications en 1925.

Une bibliographie d'ouvrages traitant des réformes de l'orthographe et de la prononciation de l'anglais peut être consultée sur [23].

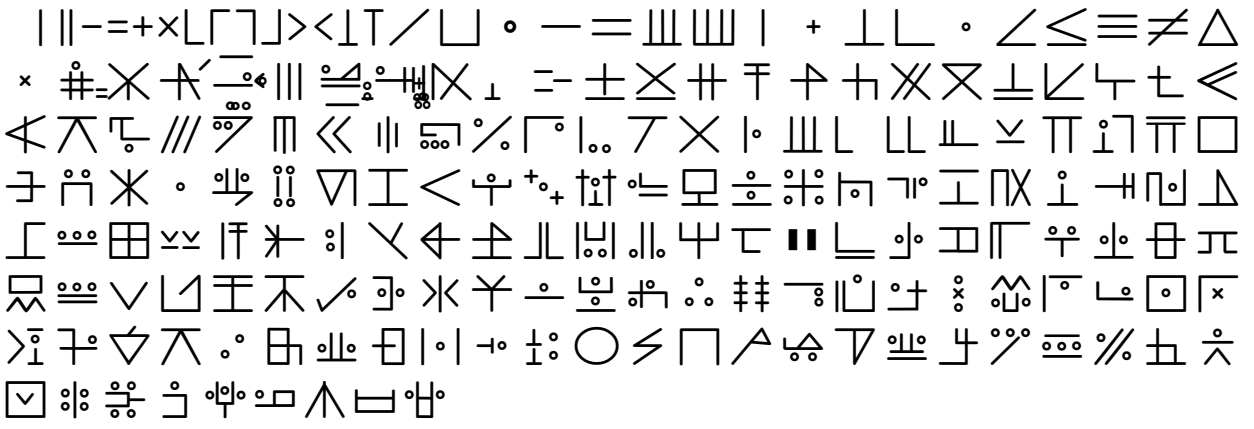


Fig. 14 : La police Axunashin.

Fig. 15 : La police Eqalar 3.

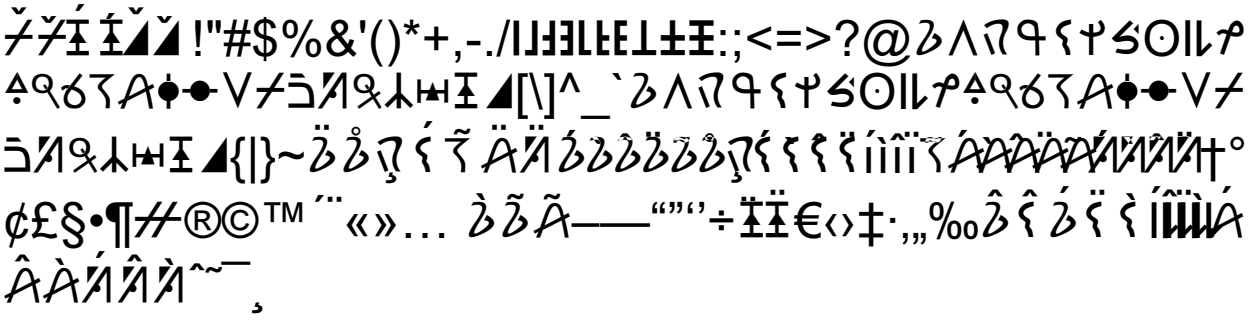


Fig. 16 : La police Ffenoc.

Fig. 17 : La police Selang.

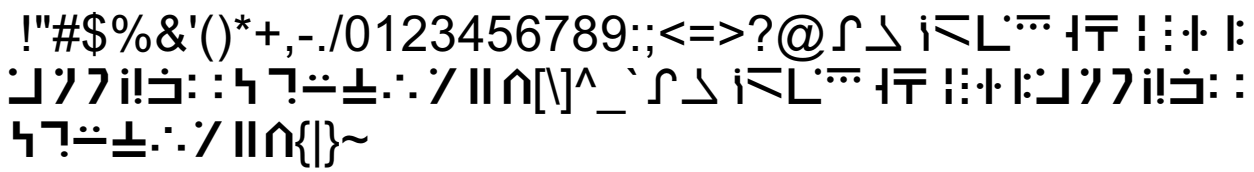


Fig. 18 : La police Standard Galactic Alphabet.

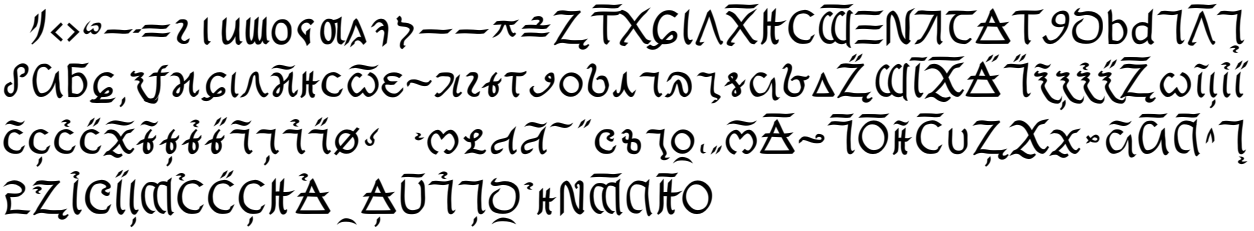


Fig. 19 : La police Verdurian.

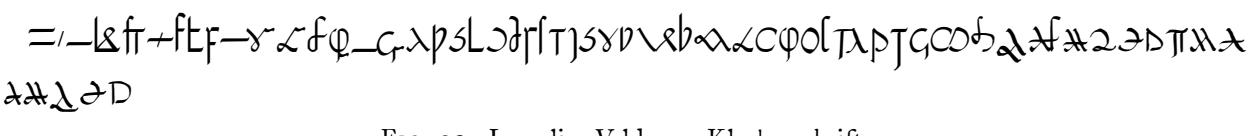


Fig. 20 : La police Valdyaans Klerkenschrift.

Références

[1] Mandel, Ladislas. *Écritures, miroir des hommes et des sociétés*. Atelier Perrousseaux Éditeur, Reillane, 1998.

[2] ——. *Du pouvoir de l'écriture*. À paraître au printemps. Atelier Perrousseaux Éditeur, Reillane, 2004.

[3] http://www.aipainunavik.com/about/f_project.html

[4] Mekta, Zupan, «Corps intégré, corps sacralisé dans la littérature contemporaine des femmes». *Religiologiques*, n° 12, Montréal, automne 1995 [UQÀM].

[5] Follick, Mont, *The Case for Spelling Reform*, Sir Isaac Pitman & Sons Ltd, Londres, 1965.

[6] Pitman, James. *The Late Dr. Mont Follick : An Appraisal. The Assault of the Conventional Alphabets on Spelling*. 1964, Bath, Pitman Press.

[7] Follick, Mont. *The Influence of English*. Williams & Norgate, Londres, 1934.

[8] Tauber, Abraham, ed. *George Bernard Shaw on Language*. Philosophical Library, New York, 1963.

[9] http://members.aol.com/rsrichmond/shawtest.html

[10] Wilson, Richard Albert. *The Miraculous Birth of Language*, Philosophical Library, New York, 1941, réédition de *The Birth of Language*, publié en 1937 à Londres par Dent pour British Publishers Guild.

[11] http://www.unifon.org/shaw-alfa.html#keyboard

[12] http://www.evertype.com/standards/csur/shavian.html

[13] http://www.unicode.org/pending/shavian/proposal/Shavian.html

[14] http://www.simonbarne.com/shavian/images.html

[15] http://victorian.fortunecity.com/vangogh/555/Spell/map-IPA.html

[16] http://www.unifon.org

[17] http://www.pixelcreation.fr

[18] http://www.leowol.de/alessio/alberobanana/albero_03.html

[19] http://cgm.cs.mcgill.ca/~luc/

[20] http://www.valdyas.org/conlang.html

[21] http://home.istar.ca/~bci/intro.htm

[22] Mencken, H.L. *The American Language : An Inquiry into the Development of English in the United States*, seconde édition, 1921.

[23] http://victorian.fortunecity.com/vangogh/555/Spell/splbib.htm

[24] http://www.theory.org/artprojects/alphabetsoup

[25] http://www.cogsci.indiana.edu/farg/mcgrawg/lspirit.html

[26] http://www.symbols.net/pictobabel

[27] http://www.simonbarne.com

Calendar

2004–2005

Oct 30 – Apr 17 "Belles Lettres: The Art of Typography", exhibition at the San Francisco Museum of Modern Art. For information, visit `http://www.sfmoma.org/exhibitions/`.

2005

Jan 3 – 7 Rare Book School, University of Virginia, January Sessions in New York City. Two one-week courses: The printed book in the West since 1800, and Book illustration processes to 1890. For information, visit `http://www.virginia.edu/oldbooks`.

Jan 12 – Feb 17 Hans Schmoller: the Penguin Years. An exhibition at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Jan 18 – Feb 25 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Scripps College, Claremont, California. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

Feb 23 – 25 Seybold Seminars, New York. For information, visit `http://www.seybold365.com/2005/`.

EuroTEX 2005
Abbaye des Prémontrés (Pont-à-Mousson, France).

Mar 7 – 11 The 15th Annual Meeting of the European TEX Users, and the $2^{2^{2^{2^{-\infty}}}}$ anniversary of both DANTE and GUTenberg, "Let's TEX Together". For information, visit `http://www.gutenberg.eu.org/eurotex2005/`.

Mar 10 – Apr 22 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Rochester Institute of Technology, Rochester, New York. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

Apr 6 – 8 27th Internationalization and Unicode Conference, "Unicode, Cultural Diversity, and Multilingual Computing". Berlin, Germany. For information, visit `http://www.unicode.org/iuc/iuc27/`.

Apr 14 – 16 TYPO.GRAPHIC.BEIRUT 2005 Conference, Lebanese American University, Beirut, Lebanon. For information visit `http://www.atypi.org/` and look for the entry under "News from members".

Apr 25 – 28 Book History Workshop, Institute d'histoire du livre, Lyon, France. For information, visit `http://ihl.enssib.fr/`.

Apr 26 Harry Carter, man of type: lecture by Martyn Thomas at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Apr 30 – May 3 BachoTEX 2005, 13th annual meeting of the Polish TEX Users' Group (GUST), "The Art of TEX Programming", Bachotek, Brodnica Lake District, Poland. For information, visit `http://www.gust.org.pl/BachoTeX/2005/`.

May 10 – Jul 17 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. University of Texas, Austin, Texas. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

May 11 – Jun 16 From chisel to pen: inscriptional letterforms from early Christian Wales. An exhibition at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Status as of 1 January 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`.

May 22 – 27 Book History at A&M: The Fourth Annual Texas A&M Workshop on the History of Books and Printing. Texas A&M University, College Station, Texas. For information, visit `http://lib-oldweb.tamu.edu/cushing/bookhistory/2005.html`.

May 24 – 27 XTech Conference, "XML, the Web and Beyond", Amsterdam RAI Centre, Netherlands. For information, visit `http://www.xtech-conference.org/`.

May 25 – 28 CIDE.8, Conférence Internationale sur le Document Electronique, "Multilingualism", Beirut, Lebanon. For information, visit `http://www.certic.unicaen.fr/cide8/`.

Jun 1 – 3 Society for Scholarly Publishing, 27th annual meeting, "Expanding the World of Scholarly Publishing", Boston, Massachusetts. For information, visit `http://www.sspnet.org`.

Jun 6 – 9 Seybold Seminars, Amsterdam. For information, visit `http://www.seybold365.com/2005/`.

Jun 6 – Jul 29 Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on topics concerning typography, bookbinding, calligraphy, printing, electronic texts, and more. For information, visit `http://www.virginia.edu/oldbooks`.

Practical TEX 2005
Friday Center for Continuing Education, Chapel Hill, North Carolina.

Jun 14 – 17 Workshops and presentations on LaTeX, TeX, ConTeXt, and more. For information, visit `http://www.tug.org/practicaltex2005/`.

Jun 15 – 18 ALLC/ACH-2005, Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "The International Conference on Humanities Computing and Digital Scholarship", University of Victoria, British Columbia. For information, visit `http://web.uvic.ca/hrd/achallc2005/` or the organization web site at `http://www.ach.org`.

Jun 20 – 23 Seybold Seminars Amsterdam 2005, Netherlands. For information, visit `http://www.seybold365.com/2005/`.

Jun 24 – 26 NTG 35th meeting, Terschelling, Netherlands. For information, visit `http://www.ntg.nl/bijeen/bijeen35.html`.

Jul 14 – 17 SHARP Conference (Society for the History of Authorship, Reading and Publishing), "Navigating Texts and Contexts". Dalhousie University, Halifax, Canada For information, visit `http://sharpweb.org/` or `http://www.dal.ca/~sharp05/`.

Jul 20 – 24 TypeCon2005, Type Directors Club, New York City. For information, visit `http://www.tdc.org/news/2004typecon2005.html`.

Jul 31 – Aug 4 SIGGRAPH 2005, Los Angeles, California. For information, visit `http://www.siggraph.org/s2005/`.

Aug 1 – 5 *Extreme* Markup Languages 2005, Montréal, Québec. For information, visit `http://www.extrememarkup.com/extreme/`.

TUG 2005
Wuhan, China.

Aug 23 – 25 The 26th annual meeting of the TeX Users Group. For information, visit `http://www.tug.org/tug2005/`.

Sep 7 – Oct 6 The Graven Image Press: lettercutting and visual metaphor in the work of Stan Greer. An exhibition at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Sep 11 – 14 Seybold Seminars, Chicago. For information, visit `http://www.seybold365.com/2005/`.

Sep 15 – 18 Association Typographique Internationale (ATypI) annual conference, Helsinki, Finland. For information, visit `http://www.atypi.org/`.

Sep 22 – 23 American Printing History Association conference, "[r]Evolution in Print: New Work in Printing History & Practice", Mills College, Oakland, California. For information, visit `http://www.printinghistory.org/htm/conference/`.

Oct 10 – 12 Fourth Annual St. Bride Conference, "Temporary Type", London, England. For information, visit `http://www.stbride.org/conference.html`.

Nov 2 – 4 ACM Symposium on Document Engineering, Bristol, UK. For information, visit `http://www.documentengineering.org/`.

Nov 29 – Dec 2 Seybold Seminars, San Francisco. For information, visit `http://www.seybold365.com/2005/`.

TUG Election Report

The deadline for nominations for the TUG Board of Directors and for TUG President in the 2005 election passed on February 1, 2005.

There are six candidates for positions on the Board for terms ending in 2009: *Steve Grathwohl*, *Jim Hefferon*, Klaus Höppner, *Arthur Ogawa*, *Steve Peter*, and David Walden. Those whose names appear in *italic* are incumbent. (Continuing directors, with terms ending in 2007, are: Barbara Beeton, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, Samuel Rhoads, and Philip Taylor.)

There are two candidates for TUG President, for a term ending in 2007: Karl Berry and Lance Carnes.

Because there are more open positions for TUG Director than there are nominees (15 Directors are allowed under the TUG bylaws), the seats on the Board are deemed uncontested, and the candidates are considered duly elected.

However, because the office of TUG President is contested, there will be an election ballot, in accordance with the TUG election procedures (available online at `http://tug.org/elecproc.html`).

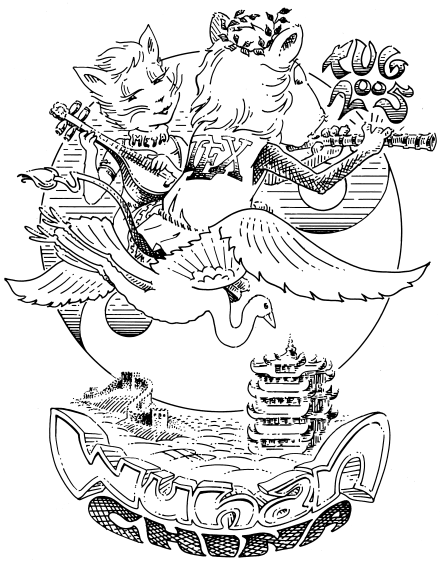
In years past, these procedures would have ballots sent out by mail. However, there has been interest in allowing for an electronic ballot. Accordingly, this Committee is preparing a proposed change to TUG Bylaws and TUG Election Procedures to provide for such a ballot. The date for mailing out the ballot and the deadline for its return have not been finalized at this time. Please see `http://tug.org/election` for up-to-date details.

To participate in the ballot, you must be a 2005 TUG member in good standing. To avoid last-minute problems, please join or renew your membership as soon as possible. See `http://tug.org/join.html` for online and printable membership forms, or contact the TUG office.

This year, two directors are retiring: Mike Sofka and Gerree Pecht; we thank them very much for their service and hope and expect to continue to work with them on TEX and TUG projects.

The election statements and other information provided by all candidates are also available online at `http://tug.org/election`. They will also be printed in a future issue of *TUGboat*.

◇ TUG Election Committee
`election@tug.org`



TUG 2005 International Typesetting Conference Announcement and Call for Papers

TUG 2005 will be held in Wuhan, China from August 23–25, 2005. CTUG (Chinese TEX User Group) has committed to undertake the conference affairs.

Wuhan is close to the birthplace of Taoism and the Three Gorges Reservoir. China is also the birthplace of typography in ancient times, and is simply a very interesting place to go.

For more information, see the conference web page at `http://tug.org/tug2005`, or email `tug2005@tug.org`.

Call for papers

Please submit a title and abstract for papers or presentations by April 1, 2005, via email to `tug2005@tug.org`. Any TEX-related topic will be considered.

Conference fees

The conference fees and deadlines for members of any TEX user group (in US dollars):

Early registration	May 20, 2005	$100
Normal registration	July 1, 2005	$220
Late registration	August 1, 2005	$380

In all cases, non-user group members add $20.

Conference bursary

Some financial assistance is available. The application deadline is March 25, 2005. Please see `http://tug.org/bursary` for details.

Hope to see you there!

Practical TEX 2005
Workshops and Presentations:
LATEX, TEX, ConTEXt,
and more

June 14–17, 2005
Friday Center for Continuing Education
Chapel Hill, North Carolina, USA

http://tug.org/practicaltex2005
conferences@tug.org

Keynote address: *Nelson Beebe, University of Utah*

Who should attend?

Mathematicians ∗ University & corporate (LA)TEX documentation staff ∗ Students ∗ Publishing company production staff ∗ Scientists ∗ Researchers

... and anyone who uses or is considering using the LATEX and TEX technical documentation system.

Further information

This four-day conference focuses on practical techniques for document production using LATEX, TEX, ConTEXt, MetaPost, and friends. It includes one day of classes and tutorials, followed by three days of presentations and workshops.

Conference attendees will enjoy an opening night reception and an (optional) banquet one evening. Coffee and lunch will be served each day of the meeting. Located in historic Chapel Hill, North Carolina.

Conference fee, hotel, and other information is available on the web site.

Pre-conference classes

On the first day, June 14, courses will be offered focusing on specific areas: Intermediate LATEX, Introduction to ConTEXt, and TEX on the Web.

- **Call for papers:** If you'd like to make a presentation, on any TEX-related topic, please email us by **March 1, 2005**.
- **Registration** forms and hotel reservation information are on the web site. Early bird discount for registrations before **March 18, 2005**.
- **Sponsorship:** If you'd like to promote your TEX products and services, or otherwise support the conference, see the web site for donation, sponsorship, advertising options. We are very grateful to Duke University for major support.

Hope to see you there!

Sponsored by the TEX Users Group.

Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Banca d'Italia,
Roma, *Italy*

Center for Computing Science,
Bowie, *Maryland*

CNRS - IDRIS,
Orsay, *France*

CSTUG, *Praha*, *Czech Republic*

Duke University, Vesic Library,
Durham, *North Carolina*

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

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

Institute for Advanced Study,
Princeton, *New Jersey*

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

KTH Royal Institute of
Technology, *Stockholm*, *Sweden*

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

Max Planck Institut
für Mathematik,
Bonn, *Germany*

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

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

Siemens Corporate Research,
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 LATEX2ε 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

EASY TABLE

KHANH HA

A TEX Table Macro Package

IRREGULAR TABLE

Sources of output	Quantity, pounds	Per pound: Selling price, cents	Per pound: Cost* about, cents	Profit about, cents	Total profit
Copper produced by mine	80,000,000	13.25	12	1.25	$1,000,000
Copper produced by reclamation plant . .	28,600,000	12.24	8	5.24	1,500,000
					$2,500,000
Earned per share from mining operation					$0.50
Earned per share from reclamation plant					0.75
					$1.24

*After depreciation, but before depletion.

CODES:

```
\aligncen\tplain{1}{IRREGULAR TABLE}
\toprul\tab{&&\bspan[3-4]Per pound\et}
\xbspan[3-4]\prul[3-4,3pt,1]
\tab{\xrow{\RC Sources of output}&\xrow{Quantity, pounds}&Selling price, cents&
Cost* about, cents&\xrow{Profit about, cents}&\xrow{Total profit}\et}
\hrul{6pt}{0}\alignbot\hgwid=1em
\tab{Copper produced by mine\dl& 80,000,000& 13.25& 12& 1.25&\$1,000,000\et}
\tab{Copper produced by reclamation plant\dl& 28,600,000& 12.24&\en 8&5.24&\en
1,500,000\et}
\prul[6-6,3pt,1]\tab{&&&&&\$2,500,000\et}
\hrul{6pt}{0}
\tab{\bspan[1-6]\em\em Earned per share from mining operation\dl\$0.50\kern3em\et}
\tab{\bspan[1-6]\em\em Earned per share from reclamation plant\dl\UL{\en 0.75}\kern3em\et}
\tab{\bspan[1-6]\RL\$1.24\kern3em\et}
\hrul{6pt}{0}\normalfnt{*After depreciation, but before depletion.}
```

◊ ◊ ◊

This TEX table macro package rivals the best of the commercial typesetting systems.

Cost? $49.95

- Dynamic table setting by template control
- Multiple mixed column spanners, subspanners, and row spanners
- Easy routines to split table footnotes and break extremely long tables
- Partial hrules, floating hrules anywhere, any length on exact baselineskip
- Automatic decimal alignment, or any special character, in irregular tables
- End columns with \et command anywhere and all vrules are automatically drawn
- Old article: http://tug.org/TUGboat/Articles/tb11-2/tb28ha.pdf
- Version10.04, far superior to the original 1989 version, is now available.

Inquire about EASY TABLE at:

www.authorkhanhha.com/EZ

301-523-4242